In-Place Algorithm for Image Rotation

Tetsuo Asano[1], Shinnya Bitou[1], Mitsuo Motoki[1], and Nobuaki Usui[2]

[1] School of Information Science, JAIST, Japan
[2] Imaging Engineering Div., Products Group, PFU Limited, Japan

Abstract. This paper presents an algorithm for rotating a subimage in place without using any extra working array. Due to this constraint, we have to overwrite pixel values by interpolated values. Key ideas are *local reliability test* which determines whether interpolation at a pixel is carried out correctly without using interpolated values, and *lazy interpolation* which stores interpolated values in a region which is never used for output images and then fills in interpolated values after safety is guaranteed. It is shown that linear interpolation is always safely implemented. An extension to cubic interpolation is also discussed.

1 Introduction

Demand for scanners is growing toward paper-less society. There are a number of problems to be resolved in the current scanner technology. One of them is to detect the direction of a document scanned, i.e., which side is the top of the document. One way is to use OCR technology to read characters which is now common to scanners. Of course, we want to avoid using OCR since it takes time. Another common problem which we address in this paper is correction of rotated documents. If the document contains only characters, then OCR is definitely a solution. Since it is costly, a geometric algorithm for such correction is required. It consists of two phases. In the first phase we detect rotation angle. Some scanners are equipped with a sensor to detect rotation angle. If no such sensor is available, we could rely on another algorithm called Hough transform [1, 2] for finding line components to detect rotation angle. To simplify the discussion, we assume a hardware sensor to detect rotation angle.

Once rotation angle is obtained, the succeeding process is rather easy if sufficient working storage is provided. Suppose input intensity values are stored in a two-dimensional array $a[.,.]$ and another array $b[.,.]$ of the same size is available. Then, at each lattice point (pixel) in the rotated coordinate system we compute an intensity value using appropriate interpolation (linear or cubic) using intensity values around the lattice point (pixel) in the input array and then store the computed interpolation value at the corresponding element in the array $b[\,]$. Finally, we output intensity values stored in the array $b[\,]$. It is quite easy. This method, however, requires too much working storage, which is a serious drawback for devices such as scanners in which saving memory is a serious demand for their built-in softwares and their costs. Is it possible to implement the interpolations without using any extra working storage? This is our question in this paper.

We propose an efficient algorithm for correcting rotation of a document without using any extra working storage. A simple way of doing this is to compute an interpolation value at each pixel in the rotated coordinate system and store the computed value somewhere in the input array $a[\,]$ near the point in the original coordinate system. Once we store an interpolation value at some element of the array, the original intensity value is lost and it is replaced by the interpolation value. Thus, if the neighborhood of the pixel in the rotated coordinate system includes interpolated values then the interpolation at that point is not correct or reliable. One of keys is a condition to determine whether interpolation at a given pixel is reliable or not, that is, whether any interpolated value is included in the neighborhood or not. Using the condition, we first classify pixels in the rotated coordinate system into reliable and unreliable ones. In the first phase we compute interpolation at each unreliable pixel and keep the interpolation value in a queue, which consists of array elements outside the rotated subimage. Then, in the second phase we compute interpolation at every pixel (x, y) in the rotated coordinate system and store the computed value at the (x, y)-element in the array. Finally, in the third phase for each unreliable pixel (x, y) we move its interpolation value stored in the queue back to the (x, y)-element in the array.

There are increasing demands for such memory-efficient algorithms. The work in this paper would open a great number of possibilities in such directions in applications to computer vision, computer graphics, and build-in software design. Image rotation is one of the most important topics for devices such as scanners. In fact there are a number of patents such as [3] proposing a method for rotating images so that the number of disc accesses is minimized and [4] using JPEG compression. Unfortunately, as far as the authors know, there are no theoretical results on this topics.

This paper is organized as follows. In Section 2 we give a mathematical description of our problem after preparing necessary notations and definitions. Then, in Section 3 we present a condition to determine whether interpolation at a given pixel is reliable or not only using local geometric information. Using the condition, we give an in-place algorithm for correcting a rotated subimage without using any extra working storage. We conclude the paper together with some open problems.

2 Problem Definition

In this section we formulate a problem mathematically. An input is an image which contains a subimage rotated by some angle θ. We assume that the rotation angle is a part of our input. Furthermore, for simplicity of argument we assume that the document is rotated in a counter-clockwise way. Rotation in the opposite direction can be dealt with in a symmetric manner.

Refer to Figure 1. The leftmost one is an image taken by a scanner. A document part in the figure is rotated. Given such a rotated image, we want to correct the rotation. The right figures ullustrate our strategy in this paper. We first execute interpolation at each pixel in the rotated subimage and store those

interpolated values over the input image. Finally, we shift the subimage to the center position.

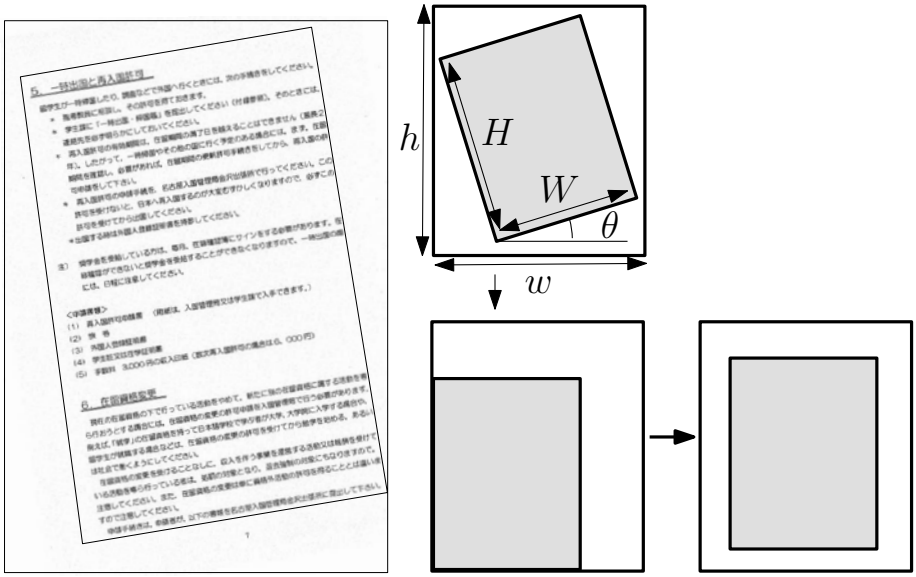


Fig. 1. An image containing a rotated subimage. A schematic illustration for correcting rotation is given in the right figures.

2.1 Input image and rotated subimages: G_{wh} and R_{WH}

Input image G consists of $h \times w$ pixels. Each pixel (x, y) is associated with an intensity level. The set of all those pixels (or lattice points in the xy-coordinate system) is denoted by $G^{\#}_{wh}$ and its bounding rectangle by G_{wh}.

Rotated subimage R consists of $h \times W$ pixels, which form a set $R^{\#}_{WH}$ of pixels (or lattice points in the XY-coordinate system). Intensity levels at each pixel (X, Y) is calculated by interpolation using intensity levels in the neighborhood.

2.2 Output image and location function

An interpolation value calculated at a pixel $(X, Y) \in R^{\#}_{WH}$ in the rotated subimage is stored (or overwritten) at some pixel $s(X, Y) \in G^{\#}_{wh}$ in the original input image. The function $s(\ )$ determining the location is referred to as a location function. A simple function is $s(X, Y) = (X, Y)$ which maps a pixel (X, Y) in $R^{\#}_{WH}$ to a pixel (X, Y) in $G^{\#}_{wh}$. We could use different location functions, but this simple function seems best for row-major and column-major raster scans. So, we implicitly fix the function.

Then, an output image after correcting rotation is a range of the function. It is rather easy to move the output image to the center position of the original rectangle G_{wh} in an in-place manner.

2.3 Correspondence between two coordinate systems

Let (x_0, y_0) be the xy-coordinates of the lower left corner of the rotated document (more exactly, the lower left corner of the bounding box of the rotated subimage). Now, a pixel (X, Y) in $R^{\#}_{WH}$ is a point (x, y) in the rectangle G_{wh} with

$$x = x_0 + X\cos\theta - Y\sin\theta,$$
$$y = y_0 + X\sin\theta + Y\cos\theta.$$

The corresponding point (x, y) defined above is denoted by $p(X, Y)$.

2.4 Scan order $\sigma(X, Y)$

Let σ be a scanning order over the pixels in $R^{\#}_{WH}$. It is a mapping from $R^{\#}_{WH}$ to a set of integers $\{0, 1, \ldots, WH-1\}$, that is, $\sigma(X, Y) = i$ means that the pixel (X, Y) is scanned in the i-th order. If σ is a row-major raster scan, $\sigma(X, Y) = X + Y \times W$ where $X = 0, \ldots, W-1$ and $Y = 0, \ldots, H-1$. A column-major raster order is symmetrically characterized by $\sigma(X, Y) = Y + X \times H$.

2.5 Window $N_d(x, y)$ for interpolation

Following the scan order σ, we take pixels in the rotated image and for each pixel (X, Y) we compute an intensity value at (X, Y) by interpolation using intensity values of pixels in the neighborhood of the corresponding point $(x, y) = p(X, Y)$. There are a number of algorithms for interpolation. The simplest one called the nearest neighbor algorithm copies an intensity level from the nearest pixel. Linear interpolation performs interpolation by linear combination of intensity values at the four immediate neighbors. An algorithm using cubic polynomials for interpolation is called a cubic interpolation. Window used for the interpolation is denoted by $N_d(x, y)$, where d is a parameter to determine the size of the window. The value of d is 1 for linear interpolation and 2 for cubic interpolation. The window size of the nearest neighbor algorithm is at most 1, but only one point is used for interpolation. The window $N_d(x, y)$ is defined by

$$N_d(x, y) = \{(x', y') \in G^{\#}_{wh} | x' = \lfloor x \rfloor - d + 1, \ldots, \lfloor x \rfloor + d, y' = \lfloor y \rfloor - d + 1, \ldots, \lfloor y \rfloor + d\}.$$

The set $N_d(x, y)$ consists of at most $4d^2$ elements. We do not describe how linear or cubic interpolation is calculated.

2.6 Basic interpolation algorithm and its problem

The following is a basic algorithm for interpolation with a scan order σ and location function $s(\ )$.

Basic interpolation algorithm
(1) Scanning

for each $(X, Y) \in R^{\#}_{WH}$ in the scan order σ do
· Calculate the location $p(X, Y) = (x, y)$ in the xy-coordinate system.
· Execute interpolation at (x, y) using intensity levelsin the window $N_d(x, y)$.
· Store the interpolation value at a pixel $s(X, Y) \in G^{\#}_{wh}$ specified by the location function.

(2) Clear the margin
for each $(x, y) \in G^{\#}_{wh}$ do
if no interpolation value is stored at (x, y)
then the intensity level at (x, y) is set to *white*.

The basic algoritm above is simple and efficient. Unfortunately, it may lead to incorrect interpolations since to calculate an interpolation value at some pixel it may reuse intensity levels resulting from past interpolations. More precise description follows:

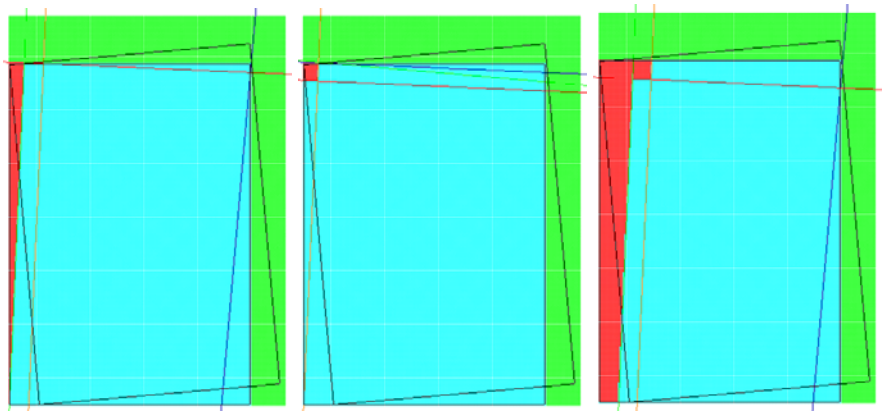
We say interpolation at $(X, Y) \in R^{\#}_{WH}$ is *reliable* if and only if none of the pixels in the window $N_d(x, y)$ keeps interpolation value. Otherwise, the interpolation is *unreliable*. "Unreliable" does not mean that the interpolation value at the point is incorrect. Consider an image of the same intensity level. Then, interpolation does not cause any change in the intensity value anywhere. Otherwise, if we use interpolated value for interpolation, the resulting value is different from the true interpolation value. We use the terminology "unreliable" in this sense. A pixel (X, Y) is called *reliable* if interpolation at (X, Y) is reliable and *unreliable* otherwise.

Figure 2 shows how frequently and where unreliable interpolations occur. In these figures those internal images are rotated counterclockwisely by degrees 5 without any left or bottom margins. When we scan the images by a usual row-major raster order with $d = 1$ (window size), those unreliable interpolations occur consecutively near the left boundary. If we use a column-major raster order instead, then we have mush less unreliable pixels as shown in (b) in the figure. If we make the window size d larger then the more unreliable pixels we have. The figure in (c) shows the case for $d = 2$.

3 Lazy Interpolation and Local Reliability Test

An idea to avoid such incorrect interpolation is to find all unreliable pixels and keep their interpolation values somewhere in a region which is not used for output image. In the following algorithm we use a queue to keep such interpolation values.

[**Lazy Interpolation**]
Q: a queue to keep interpolation values at unreliable pixels.
for each pixel $(X, Y) \in R^{\#}_{WH}$ in the order σ do
if (X, Y) is unreliable
then push the interpolation value at (X, Y) into the queue Q.
for each pixel $(X, Y) \in R^{\#}_{WH}$ in the order σ do



(a) (b) (c)

Fig. 2. Distribution of unreliable pixels colored red (dark, if no color is available): image size $= 234 \times 170$, rotation angle $= 5$ degrees counterclockwisely. (a) Row-major raster with $d = 1$, (b) column-major raster with $d = 1$, and (c) row-major raster with $d = 2$.

Calculate the interpolation value at (X, Y) and store the value at the pixel $s(X, Y) \in G^{\#}_{wh}$.
for each pixel $(X, Y) \in R^{\#}_{WH}$ in the order σ do
if (X, Y) is unreliable then
pop a value up from the queue Q and store the value at the pixel $s(X, Y)$.

Here are two problems. One is how to implement the queue. The other is how to check unreliability of a pixel. It should be remarked that both of them must be done without using any extra working storage.

Suppose we scan pixels in a rotated subimage $R^{\#}_{WH}$ according to a scan order σ and interpolation using a window of size d around each point (X, Y) is calculated and stored at an array element $s(X, Y)$ specified by the location function. Now we can define another sequence τ to determine an order of all pixels in $G^{\#}_{wh}$ to receive interpolated values. That is, the function τ is defined so that

$$\tau(s(X, Y)) = \sigma(X, Y)$$

holds for any $(X, Y) \in R^{\#}_{WH}$. Since rotated subimage is smaller than the original image, some pixels in the original image are not used for output image. That is, there are pixels (x, y) in $G^{\#}_{wh}$ such that there is no (X, Y) in $R^{\#}_{WH}$ with $(x, y) = s(X, Y)$. For such pixels (x, y) we define $\tau(x, y) = WH$. More precisely, τ is a mapping from $G^{\#}_{wh}$ to $\{0, 1, \ldots, WH\}$ such that

$\tau(x, y) = i < WH$ if i-th computed interpolation value is stored at (x, y) in $G^{\#}_{wh}$,
$\tau(x, y) = WH$ if no interpolation value is stored at (x, y).

Then, interpolation at (X, Y) is reliable in the sense defined in the previous section if none of the pixels in the window does not keep interpolated value, that is,

$$\tau(x, y) \geq \sigma(X, Y) \text{ for each } (x, y) \in N_d(p(X, Y)).$$

This condition referred to as the reliability condition.

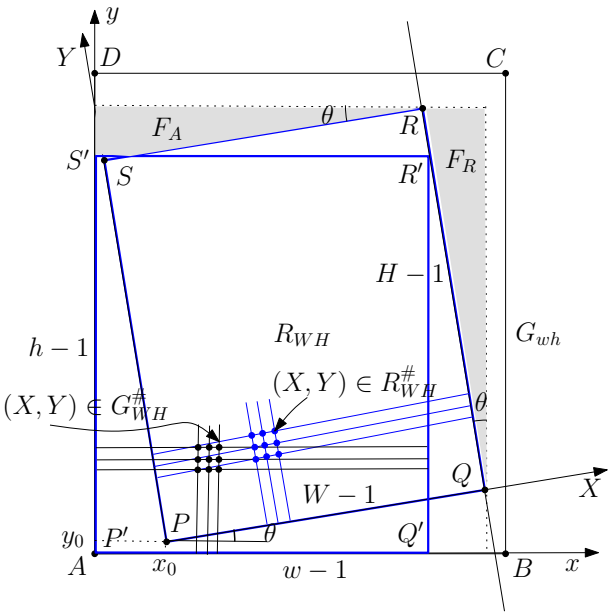


Fig. 3. Two rectangles G_{wh} and R_{WH}.

Figure 3 illustrates two rectangles, $ABCD$ for G_{wh} and $PQRS$ for R_{WH}.

3.1 Row-major raster scan for counterclockwise rotation

Consider a simple case where σ is a row-major raster scan. Let $(x, y) = p(X, Y)$, that is,

$x = x_0 + X\cos\theta - Y\sin\theta,\ y = y_0 + X\sin\theta + Y\cos\theta.$

If we order those pixels in the interpolation window of size d around (x, y) in the order of receiving interpolation values, then the first point is $(\lfloor x \rfloor - d + 1, \lfloor y \rfloor - d - 1)$ because interpolation values are also filled in $G^{\#}_{wh}$ in the same row-major raster order (restricted to the part $0 \leq x < W$ and $0 \leq y < H$). If the first part has not received any interpolation value, that is, if $\tau(\lfloor x \rfloor - d + 1, \lfloor y \rfloor - d - 1) \geq \sigma(X, Y)$, then the pixel (X, Y) is reliable. Otherwise, it is unreliable. By the definition of σ and τ, we have a simpler expression of the condition.

Lemma 1. [Local Reliability Condition] *Assuming a row-major raster order for σ and τ, pixel $(X,Y) \in R^{\#}_{WH}$ is unreliable if and only if*
(1) $x_0 + X\cos\theta - Y\sin\theta - d + 1 < X$ and $y_0 + X\sin\theta + Y\cos\theta - d < Y$, or
(2) $x_0 + X\cos\theta - Y\sin\theta - d + 1 < W$ and $y_0 + X\sin\theta + Y\cos\theta - d + 1 < Y$.

Proof By the condition stated above, a pixel (X,Y) is unreliable if and only if (1) $\lfloor x_0 + X\cos\theta - Y\sin\theta \rfloor - d + 1 \leq X - 1$ and $\lfloor y_0 + X\sin\theta + Y\cos\theta \rfloor - d + 1 \leq Y$, or
(2) $\lfloor x_0 + X\cos\theta - Y\sin\theta \rfloor - d + 1 \leq W - 1$ and $\lfloor y_0 + X\sin\theta + Y\cos\theta \rfloor - d + 1 \leq Y - 1$.

Let a and b be two arbitrary positive real numbers. Then, $\lfloor a \rfloor \geq \lfloor b \rfloor$ holds if and only if $a \geq \lfloor b \rfloor$. Also, $\lfloor a \rfloor \leq \lfloor b \rfloor$ holds if and only if $a < \lfloor b \rfloor + 1$. Using these inequalities, the above condition can be restated as in the lemma. □

An importance of Lemma 1 is that it suggests a way of testing reliability of interpolation at each pixel without using any working array. That is, it suffices to check the two conditions in the lemma.

By Lemma 1, a pixel (X,Y) is unreliable if and only if

$$\begin{aligned} &(1)\ Y > -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d+1}{\sin\theta} \text{ and } Y > \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d}{1-\cos\theta} \text{ or} \\ &(2)\quad Y > \tfrac{\cos\theta}{\sin\theta}X - \tfrac{W-x_0+d-1}{\sin\theta} \text{ and } Y > \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d+1}{1-\cos\theta}. \end{aligned}$$

By L_1, L_2, L_3 and L_4 we denote the four lines above:

$$\begin{aligned} &L_1:\ Y = -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d+1}{\sin\theta}, \quad L_2:\ Y = \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d}{1-\cos\theta}, \\ &L_3:\ Y = \tfrac{\cos\theta}{\sin\theta}X - \tfrac{W-x_0+d-1}{\sin\theta}, \quad L_4:\ Y = \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d+1}{1-\cos\theta}. \end{aligned}$$

Then, a pixel (X,Y) is unreliable if and only if the point (X,Y) is above the two lines L_1 and L_2 or above the two lines L_3 and L_4.

3.2 Column-major raster scan for counterclockwise rotation

How about a column-major raster order instead of row-major order? By similar arguments we have a similar observation.

Lemma 2. *Assuming a column-major raster order for σ and τ, a pixel $(X,Y) \in R^{\#}_{WH}$ is unreliable if and only if*
(1') $x_0 + X\cos\theta - Y\sin\theta - d < X$ and $y_0 + X\sin\theta + Y\cos\theta - d + 1 < Y$, or
(2') $x_0 + X\cos\theta - Y\sin\theta - d + 1 < X$ and $H > y_0 + X\sin\theta + Y\cos\theta - d + 1 \geq Y$.

By Lemma 2, a pixel (X,Y) is unreliable if and only if

$$\begin{aligned} &(1')\ Y > -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d}{\sin\theta} \text{ and } Y > \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d+1}{1-\cos\theta} \text{ or} \\ &(2')\ Y > -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d+1}{\sin\theta} \text{ and } Y < -\tfrac{\sin\theta}{\cos\theta}X + \tfrac{H-y_0+d-1}{\cos\theta}. \end{aligned}$$

By L'_1, L'_2, L'_3 and L'_4 we denote the four lines above:

$$\begin{aligned} &L'_1:\quad Y = -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d}{\sin\theta}, \quad L'_2:\ Y = \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d+1}{1-\cos\theta}, \\ &L'_3:\ Y = -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d+1}{\sin\theta}, \quad L'_4:\ Y = -\tfrac{\sin\theta}{\cos\theta}X + \tfrac{H-y_0+d-1}{\cos\theta}. \end{aligned}$$

Figures 4 (a) and (b) depict the four lines and the region of unreliable pixels bounded by them for each of row-major and column-major raster orders.

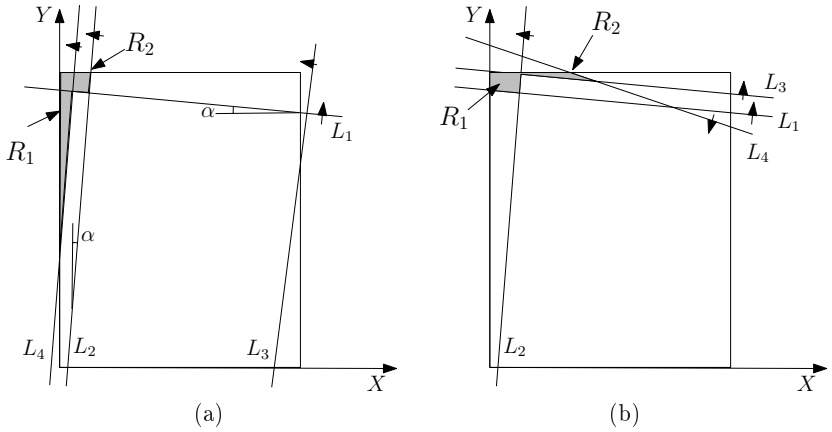


Fig. 4. Regions of unreliable pixels, (a) for row-major raster order, and (b) for column-major raster order.

3.3 Lazy interpolation for $d = 1$

Now we know how to detect possibility of unreliable pixel each in constant time. If each pixel is reliable, we just perform interpolation. Actually, if the bottom margin y_0 is large enough, then the location $s(X, Y)$ to keep interpolation value is far from a point (X, Y) and thus it does not affect interpolation around the point. Of course, if the window size d is large, then interpolations become more frequently unreliable.

Here we present an in-place algorithm for correcting rotation. For the time being we shall concentrate ourselves in the simpler case $d = 1$. A key to our algorithm is the local test on reliability. In our algorithm we scan $R^{\#}_{WH}$ three times. In the first scan, we check whether (X, Y) is a reliable pixel or not each in constant time. If it is not reliable, we calculate interpolation value and store it somewhere in $G^{\#}_{wh}$ using pixels outside the rectangle determining the output image. Such a region is called a *refuge*.

In-place algorithm for correcting rotation

Phase 1: For each $(X, Y) \in R^{\#}_{WH}$ check whether a pixel (X, Y) is reliable or not. If it is not, then calculate interpolation there and store the value in the refuge F.

Phase 2: For each $(X, Y) \in R^{\#}_{WH}$ calculate interpolation there and store the value at $(X, Y) \in G^{\#}_{wh}$.

Phase 3: For each $(X, Y) \in R^{\#}_{WH}$ check whether interpolation at (X, Y) is reliable or not. If it is not, then update the value at $(X, Y) \in G^{\#}_{wh}$ by the interpolation value stored in the refuge F.

The algorithm above works correctly when $d = 1$. The most important is that the total area of refuge available is always greater than the total number of unreliable pixels.

Theorem 1. *The algorithm above correctly computes interpolations for row-major and column-major raster scans with the location function* $s(X, Y) = (X, Y)$.

Proof We do not prove correctness of the algorithm due to space limit. We only prove that we can always find a refuge F sufficiently large. Because of similarity we only prove the theorem for the row-major raster scan.

As described earlier, the region of unreliable pixels is divided into two regions, one bounded by the two lines L_1 and L_2, and the other by L_4 and the left boundary of R_{WH}. The two regions are denoted by R_1 and R_2 in this order, as shown in Figure 4.

We have two rectangles G_{wh} corresponding to an input image and R_{WH} to a rotated subimage. With the location function $s(X, Y) = (X, Y)$, the output image is determined by rotating R_{WH} clockwisely by the angle θ and translating it so that the lower left corner coincides with the lower left corner of G_{wh}. Drawing the horizontal line through the upper right corner and vertical line through the lower right corner of R_{WH}, we have two regions F_L and F_A, as shown in Figure 3, which can be used as refuge. In other words, we can store any values there without affecting correct interpolations to be output.

To ease the proof we assume that there is no margin between the two rectangles G_{wh} and R_{WH}, that is, the four corners of R_{WH} all lie on the boundary of G_{wh}. In this case we have $x_0 = (H-1)\sin\theta$ and $y_0 = 0$. Since $d = 1$, the line L_1 passes through $(0, H-1)$ and L_4 does $(0, 0)$. The angle α between the line L_4 and the vertical line is smaller than θ because

$$\tan(\alpha) = \frac{1 - \cos\theta}{\sin\theta} < \tan\theta.$$

Thus, the area of the region (R_1 in Figure 4 (a)) bounded by L_4 and the left boundary is smaller than the refuge F_R bounded by the line RQ and the right boundary of G_{wh} (see Figure 3).

By the same reason we can also prove that the area of the region R_2 bounded by L_1 and L_2 is smaller than that of the region F_A above the line SR in Figure 3. This completes the proof. □

3.4 Lazy interpolation for $d = 2$

With a larger window of size $d \geq 2$ the algorithm above does not work due to insufficient area of the refuge. Fortunately, if the lower margin, y_0, is at least

$d-1$, then the lazy interpolation for the column-major raster works correctly. When $y_0 = d-1$ and $d \geq 2$, the unreliable region is the union of the two regions R_1 above L'_1 and L'_2 and R_2 above L'_3 and below L'_4. The line L'_2 passes through the origin, we can use the right refuge F_R as before for R_1.

What about the region R_2 bounded by L'_3 and L'_4? The line L'_4 is parallel to the horizontal side of the rectangle G_{wh} and the line L'_3 has smaller slope than the upper side of the rotated rectangle. Hence, the angle between L'_3 and L'_4 is smaller than θ. This implies that the region R_2 bounded by L'_3 and L'_4 has smaller area than the upper refuge F_A. See Figure 5 for illustration.

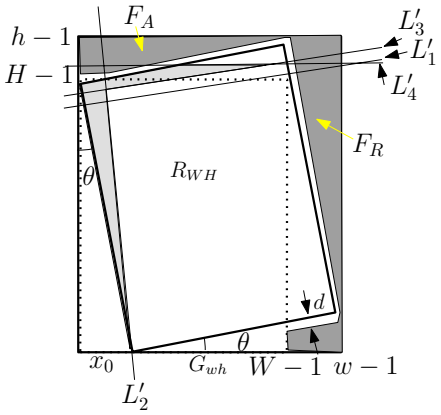


Fig. 5. The region of unreliable pixels and right and top refuges F_R and F_A.

Unfortunately we cannot use the algorithm above for a larger window, $d = 2$ since we have so many unreliable pixels even in the case. The idea here is to use a queue to store interpolation values at unreliable pixels and pop them up whenever storing them does not cause any harm for interpolations. The region outside the rotated image and the output image, shown in Figure 5, can be used for the purpose.

Assume a row-major raster order. Suppose we are going to calculate interpolation at pixels in a row Y. Then, the pixel values below the row $\lfloor y_0 + Y \cos\theta \rfloor - d$ (including the row) are never used for interpolations. Let us call the row the high limit for Y. If it is greater than the previous high limit, i.e., $\lfloor y_0 + (Y-1)\cos\theta \rfloor - d$, then we can safely store interpolation values at the row. This observation leads to the following algorithm.

In-place algorithm 2 for correcting rotation

Q = a queue containing interpolated values, using the region in the refuge.
for each row $Y = 0$ to $H-1$ do
  for each $X = 0$ to $W-1$
    if (X, Y) is unreliable

then push the interpolation value at (X, Y) into the queue Q.
if $\lfloor y_0 + Y\cos\theta \rfloor - 2 > \lfloor y_0 + (Y-1)\cos\theta \rfloor - 2$
then $Y' = \lfloor y_0 + Y\cos\theta \rfloor - 2$.
for each $X = 0$ to $W - 1$
if (X, Y) is unreliable
then store the value popped from Q at $s(X, Y)$.
else calculate interpolation value at (X, Y) and store it at $s(X, Y)$.

Unfortunately, no formal proof has not been obtained for correctness of the algorithm above. However, it has caused no problem for practical applications.

4 Concluding Remarks and Future Works

In this paper we have presented in-place algorithms for correcting rotation of a subimage contained in an image using interpolation. We have shown that as long as interpolation is implemented by linear interpolation algorithm we can always correct any rotation without using any extra working array. Correctness proof for a larger window used for cubic interpolation has been left as an open problem.

In this paper we considered two scan orders, row-major and column-major raster orders. Many other scan orders are possible. In addition to row- and column major raster scans we could scan an image at any angle. One of promising scans is the following: First, find a rotation angle θ. Then, round it to an anlge θ' defined by two pixels in a rotated subimage. Using this approximate angle, we can scan all of pixels in the rotated subimage without any extra working storage.

It is interesting to evaluate and compare those scan orders by the number of unreliable pixels. The best scan order may depend on margins. In our experience, if the bottom margin is greater than the left margin then the row-major raster is better than the column-major one. If the left margin is larger than the bottom margin, the column-major raster outperforms row-major raster. But there is no formal proof.

References

1. T. Asano and N. Katoh: "Variants for Hough Transform for Line Detection," Computational Geometry: Theory and Applications, vol. 6, pp.231-252, 1996.
2. Duda, R. O. and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," Comm. ACM, Vol. 15, pp. 11-15, 1972.
3. D. Kermisch, "Rotation of digital images," United States Patent, 4545069, 1985.
4. F.A. Micco and M.E. Banton, "Method and apparatus for image rotation with reduced memory using JPEG compression," United States Patent, 5751865, 1998.