Techniques for Area-Time Efficient Image Rotation

R. K. Satzoda, S. Suchitra and T. Srikanthan
Centre for High Performance Embedded Systems
Nanyang Technological University, Singapore
Email: {rksatzoda, assuchitra, astsrikan}@ntu.edu.sg

***Abstract*— Real-time image rotation is a necessary task in many vision based applications. Most available image rotation architectures are serial in nature, while the existing parallel techniques suffer from computational bottlenecks, limiting their performance. In this paper, we propose techniques that not only speed up the rotation process but also reduce area, yielding an area-time efficient architecture. They exploit properties of symmetry in image coordinates for parallel image rotation. We show that the number of computations, the total computation time and area cost of the proposed architecture are lesser by as much as 80%, 55% and 67% respectively when compared to the most efficient parallel architecture in recent literature.**

I. Introduction

Rotation of images in real-time is an important step employed in various vision based applications like registration of geo-sat images [1] and rectification of medical endoscopic images [2]. Conventionally, image rotation by an angle θ involves applying the following equations on every pixel coordinate (x', y') to get its rotated coordinate (x, y).

$$\begin{aligned} x &= x' \, cos\theta - y' \, sin\theta \\ y &= y' \, cos\theta + x' \, sin\theta. \end{aligned} \tag{1}$$

Although limited in number, application specific hardware implementations to rotate images in real-time can be found in [3]–[6]. These use either CORDIC [7] or look-up tables (LUTs) to implement the rotation equations in (1). In [5], rotation is performed by mapping pixels along a skew raster scan line in the source image to a horizontal scan line in the rotated image space. It uses an LUT to store the sine and cosine values and the computations are performed serially as the image is read. Tsuchida et Al. proposed a new method of high-speed image rotation using twice skew transformations [3]. Although the rotation process is parallelized to an extent, the algorithm requires a preprocessing step to prepare several data tables for every input angle. Moreover, this approach is valid only for small angles of rotation.

Ghosh et Al. [4] proposed a parallel image rotation engine which gives orders of magnitude higher throughput as compared to serial methods in [3], [5]. In this algorithm, the image is divided into blocks using a grid and all the blocks are processed in parallel. As an initialization step, the block centers are first rotated using the CORDIC engine and stored for a given input angle. Subsequently, pixels from all the blocks having the same relative position with respect to the pre-computed block centers are rotated in parallel by adding a fixed offset. This offset is shown to be the same for all the blocks. This way, k^2 pixels are rotated in parallel when the image is divided using a $k \times k$ grid, resulting in high throughputs. However, this architecture suffers from a high initialization latency to rotate the block centers using CORDIC, especially for large grid sizes. In addition, the offset computation also uses CORDIC which adds to the overall latency.

The hierarchical rotation technique in [6] largely overcomes the limitations of [4] by introducing a hierarchy parameter h. The image is divided into h layers and the number of rotated centers to be computed using CORDIC is limited to h only, which is drastically lesser than [4]. These rotated centers from each hierarchy layer are then used to determine the rotated block centers using simple additions. The offset generation in [4] is also further optimized tremendously in [6] by eliminating the use of CORDIC for offset computation.

Though the latency is reduced by orders of magnitude in [6] as compared to [4], the benefits of speed-up is nullified by the increased initialization latency for high values of hierarchy h. In this paper, we introduce novel techniques based on symmetry of image coordinates, that drastically reduce this initialization latency. In addition, it also reduces the number of computations to get the overall rotated coordinates without compromising on the area cost. In fact the proposed architecture is shown to be more efficient than [6]. The rest of the paper is organized as follows. In Section II, we propose and verify techniques that lead to an efficient rotation architecture with high performance and low area costs. Section III describes the block diagram of the proposed rotation architecture. In Section IV, the proposed techniques are compared against [6] in terms of number of computations, area cost, total computation time and the area-time tradeoffs. The paper is concluded in Section V.

II. Proposed Techniques for Area-Time Efficiency

Since the motivation of this work is to circumvent the bottleneck of the hierarchical rotation technique in [6] (henceforth referred to as HRT), a brief discussion of the steps in the existing HRT has been included before presenting the proposed improvements for the corresponding steps.

A. Initialization Process in HRT

The initialization stage in the HRT involves division of a square sized $m \times m$ image into hierarchical layers as shown in Fig. 1 for a hierarchy $h = 3$. The size of ith layer is a quarter of the $(i-1)$th layer, i.e. the first layer is $m \times m$, the second layer is $m/2 \times m/2$ and so on. Each layer is further divided into

978-1-4244-3828-0/09/$25.00 ©2009 IEEE

four quadrants. In Fig. 1, the rotated quadrant centers (rqc) of ith layer and jth quadrant are given by (x_{ij}, y_{ij}). In this paper, the top-right quadrant is considered as first quadrant and the other quadrants are numbered in the anti-clockwise direction. The rqc of the first quadrant in each layer are computed using CORDIC and the rqc of the rest of the quadrants in the same layer are inferred using symmetry. This can be summarized using the following relations:

$$\begin{array}{lll} (x_{i2}, y_{i2}) = (-y_{i1}, x_{i1}) & ; & (x_{i1}, y_{i1}) : CORDIC \\ (x_{i3}, y_{i3}) = (-x_{i1}, -y_{i1}) & ; & (x_{i4}, y_{i4}) = (y_{i1}, -x_{i1}) \end{array} \quad (2)$$

Once the rqc of each layer are obtained, they are added in different combinations to give the rotated block centers (rbc) of all the 4^h blocks in the image. For example, in Fig. 1, x_{11} and x_{21} are added to the four rqc of layer 3, resulting in the rbc of the four blocks on the top-right corner, as shown. The rbc of the rest of the blocks in Fig. 1 can be obtained similarly. $4^h(h-1)$ additions are required to generate these 4^h centers [6].

Thus, the parameter h directly affects the total computational time of HRT. Increase in h implies more number of blocks, which means more parallelism and lower computation time. However, a higher h results in more layers and rqc, leading to an exponential increase in the number of addition operations in the initialization process. This push-pull effect is shown in [6] where the benefits of speed up due to parallelism start to get nullified beyond a certain optimal hierarchy h. In the next few sections, effective ways are proposed to reduce the initialization latency as well as the area to develop an area-time efficient hierarchical rotation architecture.

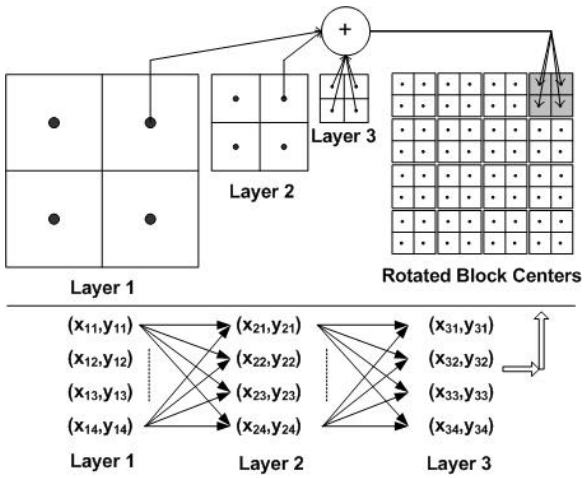

Fig. 1. Hierarchy layers for $h = 3$ and generation of rotated block centers in [6]

B. Proposed Bottom-up Initialization Process

Instead of adding the rqc of different layers in all combinations as shown in Fig. 1, the proposed techniques exploit symmetry in the rotation equations, which is facilitated by employing a systematic bottom-up approach. The first step in the proposed method is to express the initialization equations in a bottom-up fashion and the second is to optimize the computations based on the properties of image symmetry. The two steps are explained below:

1) Step 1: Expressing as Bottom-up Addition: The bottom-up approach refers to the flow of data from the deepest layers upward to the bigger layers. That is, the rqc of hierarchy layers h and $h-1$ are first used for generating intermediate results, which are further added with rqc of the next layer $h-2$ and so on till the first layer is reached. The addition equations shown in Fig. 1 can be rewritten as (3) based on this bottom-up approach, where X_i of size $2^{h-i+1} \times 2^{h-i+1}$ denotes the intermediate results for layer i i.e. summation of x-coordinates of rqc of all layers till that ith layer. When $i = 1$, the $2^h \times 2^h$ sized matrix X_1 contains the rotated block centers, rbc.

$$X_i = \begin{bmatrix} X_{i2} & X_{i1} \\ X_{i3} & X_{i4} \end{bmatrix} = \begin{bmatrix} x_{i2} + X_{i+1} & x_{i1} + X_{i+1} \\ x_{i3} + X_{i+1} & x_{i4} + X_{i+1} \end{bmatrix} \quad (3)$$

In the above equation, X_{i1}, X_{i2}, X_{i3} and X_{i4} represent the sub-matrices corresponding to the first, second, third and fourth quadrants of X_i respectively.

2) Step 2: Exploiting Symmetry for Optimization: We now propose techniques based on symmetry to reduce these additions to compute X_i at every layer in the bottom-up approach. The proposed method computes only two quadrants of X_i, top-right and bottom-right, i.e. X_{i1} and X_{i4}. These are then used to infer the rotated centers in the other two quadrants, resulting in 50% reduction in the number of computations at every layer. This is based on the observation that at any hierarchy layer i, if X_{i1} and X_{i4} are given, then X_{i2} and X_{i3} can be derived as follows:

$$X_{i2} = \psi_h\psi_v(-X_{i4}) \quad ; \quad X_{i3} = \psi_h\psi_v(-X_{i1}) \quad (4)$$

where $\psi_h\psi_v()$ on a matrix refers to a vertical flip operation along a vertical axis followed by a horizontal flip operation along a horizontal axis. Mathematically $\psi_h\psi_v(M) = M'$ such that $M'(u,v) = M(s_M - u + 1, s_M - v + 1)$, where $s_M \times s_M$ is the size of M. This is illustrated with an example for $i = 2$. Let $h = 3$. For the deepest layer $i = 3$, X_3 contains the four x-coordinates of the rotated quadrant centers only. Using the symmetry relationships in (2), X_3 can be defined as:

$$X_3 = \begin{bmatrix} x_{32} & x_{31} \\ x_{33} & x_{34} \end{bmatrix} = \begin{bmatrix} -y_{31} & x_{31} \\ -x_{31} & y_{31} \end{bmatrix} \quad (5)$$

Using the definition of X_i for $i < h$ in (3), X_2 can be determined as:

$$X_2 = \begin{bmatrix} X_{22} & X_{21} \\ X_{23} & X_{24} \end{bmatrix} = \begin{bmatrix} x_{22} + X_3 & x_{21} + X_3 \\ x_{23} + X_3 & x_{24} + X_3 \end{bmatrix}$$

$$= \begin{bmatrix} x_{22} - y_{31} & x_{22} + x_{31} & x_{21} - y_{31} & x_{21} + x_{31} \\ x_{22} - x_{31} & x_{22} + y_{31} & x_{21} - x_{31} & x_{21} + y_{31} \\ x_{23} - y_{31} & x_{23} + x_{31} & x_{24} - y_{31} & x_{24} + x_{31} \\ x_{23} - x_{31} & x_{23} + y_{31} & x_{24} - x_{31} & x_{24} + y_{31} \end{bmatrix}$$

Applying symmetry conditions $x_{22} = -y_{21}$, $x_{23} = -x_{21}$ and $x_{24} = y_{21}$ from (2)

$$= \begin{bmatrix} -y_{21} - y_{31} & -y_{21} + x_{31} & x_{21} - y_{31} & x_{21} + x_{31} \\ -y_{21} - x_{31} & -y_{21} + y_{31} & x_{21} - x_{31} & x_{21} + y_{31} \\ -x_{21} - y_{31} & -x_{21} + x_{31} & y_{21} - y_{31} & y_{21} + x_{31} \\ -x_{21} - x_{31} & -x_{21} + y_{31} & y_{21} - x_{31} & y_{21} + y_{31} \end{bmatrix} \quad (6)$$

From X_2 above, it can be seen that when the elements in the first quadrant of X_2, i.e. X_{21}, are negated and flipped vertically and horizontally, the third quadrant elements X_{23} are obtained. Similarly, X_{22} can be derived from X_{24}. This proof can be extended to any layer i in a similar fashion. Thus, at every layer i, only half of X_i is computed using a simple adder-subtractor engine. This effectively halves the computations at every layer resulting in tremendous computation savings to obtain the rbc in X_1.

C. Rotated Coordinates' Generation

With the rotated block centers determined, we now look at the generation of the rotated coordinates of the pixels. In [6], an offset is computed and added to all the rotate block centers in parallel. This results in rotated coordinates for $2^h \times 2^h$ pixel coordinates in parallel which are located at the same relative positions in all the blocks. This would require $2^h \times 2^h$ parallel adder array for x-coordinates and an equal number for y-coordinates.

We optimize the rotated coordinates generation by employing similar properties that were introduced in the initialization step in Section II-B. Let the rotated x-coordinates for all $m \times m$ coordinate positions in the input image be given by R_x. A careful study of R_x shows the following property: *If* $R_x = \left[R_x^l \; R_x^r\right]$, *then*

$$R_x^l \quad = \quad \psi_h \psi_v(-R_x^r). \tag{7}$$

where R_x^l *and* R_x^r *denote the left and the right halves of* R_x. This property can be derived from (4) and the fact that a constant offset is added to all the block centers in X_1 in order to generate the rotated pixel coordinates. This has the following direct implications on the computational and storage costs:

- Rotation needs to be done for only one half of the image, which implies that offset is added to only half the number of rbc in X_1. The rotated coordinates of the other half can be inferred by simple negation and flipping. This reduces the adder cost.
- The rbc of only half the number of blocks need to be stored. In other words, for a hierarchy h, we do not store all the $2^h \times 2^h$ elements of X_1. Instead, only the right half of X_1 need to be stored to generate R_x^r coordinates resulting in 50% savings to store the rbc.

D. Rotated y-coordinate Generation

The generation of rotated x-coordinates has been discussed till now. These rotated x-coordinates that are generated using the proposed techniques can be directly used to infer the rotated y-coordinates. The relationship between the rotated y-coordinates R_y and x-coordinates R_x is given by:

$$R_y \quad = \quad \psi_h(R_x^T) \tag{8}$$

where R_x^T is the transpose of R_x and $\psi_h()$ flips R_x^T along the horizontal axis. This relationship further reduces the computational complexity and memory by an additional 50%.

III. Proposed Improved Architecture

The block diagram representation of the proposed improved architecture for image rotation is shown in Fig. 2. The first-quadrant centers of h layers, $(x'_{11}, y'_{11}), (x'_{21}, y'_{21}), \ldots, (x'_{h1}, y'_{h1})$, are sent as input to the CORDIC engine, which generates rqc of the first quadrant of all layers, i.e. $(x_{11}, y_{11}), (x_{21}, y_{21}), \ldots, (x_{h1}, y_{h1})$. These first quadrant rqc are then sent through the symmetry operation block which performs negation and flipping operations to generate the x-coordinates of the rqc of the remaining quadrants using the relationships in (2). The rotated x-coordinates of all the quadrants for each layer are then added using the proposed bottom-up approach from the deepest hierarchy layer to the topmost hierarchy layer, to generate the rbc in the first and fourth quadrants of X_1, i.e. X_{11} and X_{14}. This results in $2^h \times 2^{(h-1)}$ x-coordinates. These are rotated block centers for rotating the right half of the image. The offset computed by the offset generation block is added to $2^h \times 2^{(h-1)}$ centers to give the rotated coordinates of the pixels in the right half of the image. These are then used to infer the rotated coordinates of the pixels in the left image half as well as the y-coordinates.

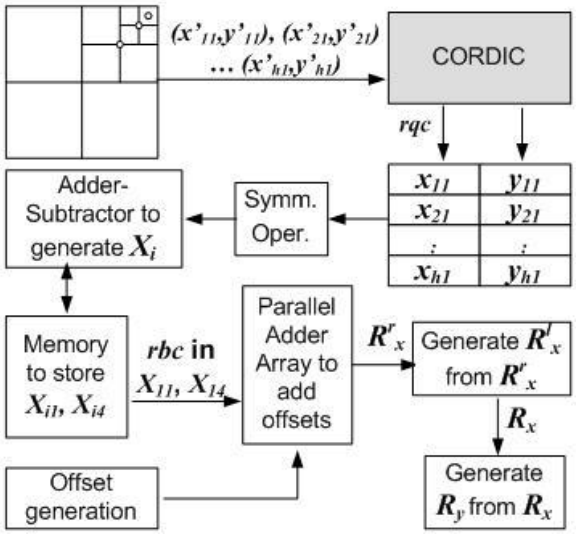

Fig. 2. Block diagram of the proposed architecture

IV. Comparative Analysis

In this section, the proposed improved hierarchical rotation engine is compared with the architecture in [6] in terms of computational complexity, area cost and total computation time followed by a discussion on the area-time tradeoffs.

A. Computational Complexity of Initialization Stage

As shown previously, [6] adds all the four rqc of each of the h layers in all combinations to generate $2^h \times 2^h$ block centers. This involves $4^h(h-1) \times 2$ additions to determine both x and y coordinates of the rbc. In the proposed bottom-up approach, at every hierarchy layer i, $2^{(2h-2i+1)}$ addition operations are

performed for x-coordinates only. This results in a total of $\sum_{i=2}^{h} 2^{2i-1}$ additions to generate rbc in X_1 matrix. For $h = 3$, the proposed architecture reduces the computations by 84% as compared to [6] for computing the rbc. The savings increases to 89% as h increases to 5.

B. Area Savings

The architecture in [6] stores the x and y coordinates of all the rbc. This requires $2^h \times 2^h \times 2 = 2^{2h+1}$ w-bit registers. The proposed techniques eliminate all memory for y-coordinates. Furthermore, we store the rotated x-coordinates of only half the block centers requiring $2^h \times 2^{h-1} = 2^{2h-1}$ w-bit registers. This results in a total memory reduction of 75% for any hierarchy. In [6], a total of $2^h \times 2^h \times 2$ w-bit adders are required to add the offset to all the x and y coordinates of the block centers to generate the rotated coordinates in parallel. In the proposed method, we add the offset to only the x-coordinates of half the rbc in X_1 resulting in $2^h \times 2^{h-1} = 2^{2h-1}$ w-bit adders. The coordinates in the other half are generated by negation and flipping as shown in Section II-C. Negation is a simplified addition operation and flipping does not involve any extra cost. If the cost of negation is taken as half of normal two-operand addition, then we require another $2^{2h-1}/2$ w-bit adders. The total number of adders is now equal to $2^{2h-1} + 2^{2h-2}$ resulting in 62.5% adder cost reduction as compared to [6] for any given hierarchy.

C. Computation Time and Area-Time Tradeoff

As explained earlier in Section II-A, the plot of computation time T versus h for HRT in [6] shows an optimal hierarchy where T is minimum. Table I lists the minimum computation times, T_{min}, corresponding to the optimal hierarchies in [6] for different image sizes $m \times m$. For the same hierarchy settings yielding T_{min} in HRT, computation time T for the proposed improved architecture is computed and listed in Table I. It can be seen that the proposed architecture shows a speedup of atleast 1.2 times for all m. Table I also compares the area cost of the proposed architecture against HRT. The area costs are computed for the same hierarchy settings in both approaches (corresponding to the optimal h in HRT). As in HRT, for area cost computation, a 16-iteration CORDIC engine is taken for generating the rqc of hierarchy layers. It is assumed to have ripple carry adders and barrel shifters (made of $log_2 16$ layers of 2-to-1 multiplexers) for shifting and addition in one cycle [7]. All registers and adders are assumed to be 16-bit wide. The areas of a 2-to-1 multiplexer and a one-bit register are assumed to be 0.38 and 0.80 times the area of a full-adder [8]. Table I shows that the proposed architecture occupies about 67% lesser area as compared to HRT for all m. The reduction in area and timing reduces the area-time product metric (AT) also by at least 70% as shown in Table I.

It should be noted that the values of T for the proposed architecture in Table I are not the least possible computation times. The computation time of the proposed architecture can be further decreased to 296, 488, 1016, 1784 and 3848 cycles for $m = 128, 256, 512, 1024$ and 2048 respectively. This is possible because of the increased parallelism that is introduced by the proposed techniques, which allow for higher hierarchies. Although this incurs an additional area cost, it was found that the AT measure of the proposed solution designed for the least computation time is still lesser by at least 12% than that of HRT designed for its optimal h.

TABLE I
COMPARISON OF COMPUTATION TIME AND AT SAVINGS

	T_{min} (cycles)	T (cycles)	Area (# full adders		AT
$m \times m$	[6]	Proposed	[6]	Proposed	% Savings
128×128	432	344	3792.64	1283.84	73.04
256×256	1088	488	14851.84	4816.64	85.45
512×512	1856	1256	14851.84	4816.64	78.05
1024×1024	4928	4328	14851.84	4816.64	71.52
2048×2048	8272	4856	59088.64	18947.84	81.18

V. CONCLUSION

In this paper, we have proposed efficient techniques towards realizing an area-time efficient image rotation architecture. The initialization latency to generate the block centers in the hierarchical rotation technique in [6] was significantly reduced by employing the proposed bottom-up approach and symmetry based techniques. The proposed solution for the optimal hierarchies listed in [6] is shown to be atleast 1.2× faster and 67% less area intensive. Furthermore, we have shown that proposed architecture can be designed at higher hierarchy levels as compared to [6] to increase the performance and yet give lower area-time product metrics.

REFERENCES

[1] J. LeMoigne, "Parallel registration of multisensor remotely sensed imagery using wavelet coefficients," *Proc. of SPIE Wavelet Applications Conference*, pp. 432 – 443, 1994.

[2] D. Koppel, Y. F. Wang, and H. Lee, "Automated image rectification in video-endoscopy," *Proc. of International Conference on Medical Image Computing and Computer-assisted Intervention*, pp. 1412 – 1414, 2001.

[3] N. Tsuchida, Y. Yamada, and M. Weda, "Hardware for image rotation by twice skew transformation," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 10, pp. 1350 – 1359, 1989.

[4] I. Ghosh and B. Majumdar, "VLSI implementation of an efficient asic architecture for real-time rotation of digital images," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 9, pp. 449 – 462, 1995.

[5] S. M. Bhandarkar and H. Yu, "A VLSI implementation of real-time image rotation," *Proceedings of International Conference on Image Processing*, vol. 2, pp. 1015 – 1018, 1996.

[6] S. Suchitra, S. K. Lam, C. T. Clarke, and T. Srikanthan, "Accelerating rotation of high-resolution images," *IET Proc. on Vision Image Signal Processing*, vol. 153, no. 6, pp. 815 – 824, December 2006.

[7] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," *Proceedings of 1998 ACM/SIGDA Sixth International Symposium on FPGA*, pp. 191 – 200, February 1998.

[8] "TSMC 0.18 μm process 1.8-Volt Sage-X standard cell library databook," October 2001.