Accelerating rotation of high-resolution images

S. Suchitra, S.K. Lam, C.T. Clarke and T. Srikanthan

Abstract: Real-time image rotation is an essential operation in many application areas such as image processing, computer graphics and pattern recognition. Existing architectures that rely on CORDIC computations for trigonometric operations cause a severe bottleneck in high-throughput applications, especially where high-resolution images are involved. A novel hierarchical method that exploits the symmetrical characteristics of the image to accelerate the rotation of high-resolution images is presented. Investigations based on a 512×512 image show that the proposed method yields a speedup of $\sim 20\times$ for a mere 3% increase in area cost when compared with existing techniques. Moreover, the effect of hierarchy on the computational efficiency has been evaluated to provide for area–time flexibility. The proposed technique is highly scalable and significant performance gains are evident for very high-resolution images.

1 Introduction

The emergence of time-critical applications for medical image processing, computer vision and pattern recognition has created the need for real-time image processing solutions. Realisation of high-throughput image processing solutions has been made feasible with the recent advancements in high-speed camera systems [1, 2], and the advent of state-of-the-art CAD tools that lead to rapid hardware implementations. A fundamental problem that is commonly found in these time-critical applications is rotation of images, often of high resolution. Real-time rectification of medical images during endoscopy [3] and registration of astronomical images [4, 5] are typical examples of applications that require high-throughput image rotation.

Conventional rotation of images involves performing trigonometric computations on each image pixel. Previously reported hardware implementations for image rotation rely on a LUT (look-up table) or a CORDIC (coordinate rotation digital computer)-based method to simplify the complex trigonometric operations. CORDIC [6] is capable of performing a series of micro-rotations on a vector lying on the $X–Y$ plane over a desired input angle using simple add-shift operations. However, the iterative nature of CORDIC can lead to performance degradation, especially for images of high resolution. An alternative technique to overcome the inefficiency of the trigonometric computations is by employing an LUT to store all the possible sine and cosine values. However, as the size of the LUT is governed by the resolution of angles that can be rotated, the hardware cost incurred becomes undesirable for applications that require high rotation resolutions.

In this paper, we propose a novel hardware implementation for rotation of high-resolution images, which replaces the computer intensive CORDIC computations with simple additions. The proposed method is based on three simple techniques: (1) hierarchical strategies, (2) exploitation of image symmetry and (3) exploitation of the pixel order. In addition, we introduce a parameter called Hierarchy and show how it affects the overall performance of the architecture. It is demonstrated that an optimal choice of this parameter can lead to very high-throughput computations. It is also noteworthy that the proposed implementation can be integrated with various pixel interpolation schemes to map the rotated pixel positions onto the original pixel positions. We will provide a discussion on the various pixel interpolation schemes and the corresponding hardware cost and accuracy implications on the proposed system. Finally, we show that the proposed technique can achieve significant speedup at a marginal increase in hardware cost, while providing the same degree of accuracy with an existing technique.

In the next section, we begin by describing the two existing hardware implementations for image rotation that employ LUT and CORDIC-based methods, respectively. The limitations of these methods for very high-throughput and low-cost realisations will be highlighted. We then describe the proposed strategies for area–time efficient image rotation. This is followed by an accuracy analysis of the proposed system and performance–area comparison with an existing technique. Finally, we show the hardware implications of the proposed technique for higher resolution images.

2 Existing techniques

To date, there has been very limited reported work on hardware implementations for image rotation. There are, however, two notable contributions by Ghosh and Majumdar [7] and Bhandarkar and Yu [8], which employ CORDIC-based and LUT-based methods, respectively.

2.1 LUT-based implementation

Bhandarkar and Yu [8] proposed an image rotation engine that is based on the mapping of pixels along a skew raster scan line in the source image to a horizontal scan line in the rotated image space. The skew lines are parallel to

S. Suchitra, S.K. Lam and T. Srikanthan are with the Centre for High Performance Embedded Systems, Nanyang Technological University, Singapore 637553, Singapore

C.T. Clarke is with the Department of Electronic and Electrical Engineering, University of Bath, Bath, BA2 7AY, UK

E-mail: assuchitra@ntu.edu.sg

each other in the source image and inclined at an angle $-\theta$ to the horizontal, where θ is the specified rotation angle. The design makes use of a LUT to store the sine and cosine values that are required for computation of the initial rotated position. Owing to the backward mapping of the pixels onto the rotated image, the holes or measles that are normally introduced in the rotated image have been eliminated.

However, the throughput is still limited, as there is no parallelism exploited in this architecture. Although this technique has scope for parallelism, it would incur a computational overhead and a significant increase in hardware cost. It has been shown how the bulk of the operations can be handled by just additions. However, it becomes inevitable to perform at least one set of multiplication operations to obtain the address of the first pixel in the first scan line. This is because only sine and cosine values are stored and computations resembling (1) and (2) are mandatory to obtain the starting address of the first skew line, before generating the remaining addresses by simple additions. This overhead becomes more prominent when the architecture is parallelised. Moreover, it does not scale well for applications requiring high rotation resolutions. For example, if the resolution of the rotation is made finer (e.g. 0.1° instead of 1°), the memory required to store the sine/cosine values will increase by ten times.

2.2 CORDIC-based implementation

2.2.1 Pixel-by-pixel CORDIC-based rotation: The CORDIC engine can be employed to improve the hardware efficiency of image rotation systems that are based on computing transformations on each image pixel [9]. Let (x', y') be the coordinates of a pixel, which has been rotated by an angle of ϕ where (x_0, y_0) is the original position of the pixel. Equations (1) and (2) show the relationship between (x', y') and (x_0, y_0), where $\delta = +1/-1$, when ϕ is +ve/−ve

$$x' = \cos\phi(x_0 - \delta y_0 \tan\phi) \quad (1)$$

$$y' = \cos\phi(y_0 + \delta x_0 \tan\phi) \quad (2)$$

The CORDIC algorithm, which was developed by Volder [6], can be used to reduce the complexity of the trigonometric computations in (1) and (2). The CORDIC computation performs a series of micro-rotations on a vector lying on the X–Y plane over a desired input angle using simple add-shift operations. The algorithm is based on the principle that any angle can be approximated as a summation of n micro angles of the form $\arctan(2^{-i})$ (i.e. $\phi \simeq \sum_{i-1}^{n} \pm \arctan(2^{-i})$). The equations for the CORDIC micro-rotations shown in (3) and (4) can be derived from (1) and (2) by replacing the multiplication with simple shift operations by i positions

$$x_{i+1} = k(x_i - \delta_i y_i 2^{-i}) \quad (3)$$

$$y_{i+1} = k(y_i + \delta_i x_i 2^{-i}) \quad (4)$$

Image rotation architectures that rely on the pixel-by-pixel transformations are intensive in CORDIC computations. In the recent past, there have been many attempts to improve the performance of the CORDIC engine, which included the use of redundant number systems [10, 11] and architectural improvements such as pipelining and parallelism [12]. Although these advancements have shown tremendous improvement in the CORDIC algorithm, the number of CORDIC operations will still remain as the bottleneck that restricts the throughput of the architecture. This is exacerbated in systems that process high-resolution images, as the number of CORDIC computations increases proportionally with the number of image pixels.

2.2.2 Pipelined-parallel architecture: Ghosh and Majumdar [7, 13] have proposed schemes to reduce the number of CORDIC computations in an image rotation system. A pipelined-parallel architecture for image rotation using CORDIC was proposed where a 512×512 image is first divided into an 8×8 window grid [7]. The rotated positions of the centres of the 64 window grids are first calculated using the CORDIC engine and stored in the initialisation stage as shown in Fig. 1. In Pipeline Stage A, the relative rotated offset of each pixel with respect to its window centre is computed using the CORDIC engine. Only the pixels of a single window are considered for the offset computation, as the relative offsets of the corresponding pixels in each window are the same. In Pipeline Stage B, this offset is added in parallel to the 64 rotated window centres using local adders to generate the rotated positions of the 64 corresponding pixels simultaneously. The CORDIC engine employed for a 512×512 image [7] performs 12 CORDIC iterations with 20-bit data path width.

The bottleneck of this approach arises from the iterative CORDIC micro-rotations. First, the approach has high initialisation latency, as 64 CORDIC computations are required to obtain the rotated window centres. In addition, the CORDIC computations are required in Pipeline Stage A for each pixel in a single window. This incurs an additional $64 \times 64 = 4096$ CORDIC calculations, which contributes to high overall latency.

3 Proposed technique

The image rotation scheme proposed in this paper addresses the computational bottleneck of the approach in Ghosh and Majumdar [7], without the need for large LUT, which is a potential problem in the work of Bhandarkar and Yu [8]. The following schemes have been proposed to increase the throughput of the rotation process:

(i) Employing a hierarchical approach to generate the rotated centres in order to significantly reduce the initialisation latency.
(ii) Exploiting the symmetrical characteristics of the image coordinate system to further reduce the number of CORDIC computations for calculating the rotated centres.
(iii) Replacing the CORDIC computations in the offset calculation with additions by exploiting the order of the pixel coordinates.

3.1 Hierarchical approach for generating rotated centres

This method aims to reduce the number of CORDIC computations in the initialisation stage by recursively partitioning the image into hierarchical quadrant layers. To explain this method, we take the example of an 8×8-window grid and its three hierarchical quadrant layers as shown in Fig. 2. A window centre lies in a single quadrant in each of the three quadrant layers. In the example shown, centre P has {A, B, C} as its set of identifying quadrants in layers 1, 2 and 3, respectively. The rotated position of any centre P (with respect to the image centre) is equivalent to the sum of the rotated positions of the centres of the identifying quadrants with respect to the layer centres. Thus, the number of CORDIC operations to obtain the 8×8 window centres can be reduced by computing only the

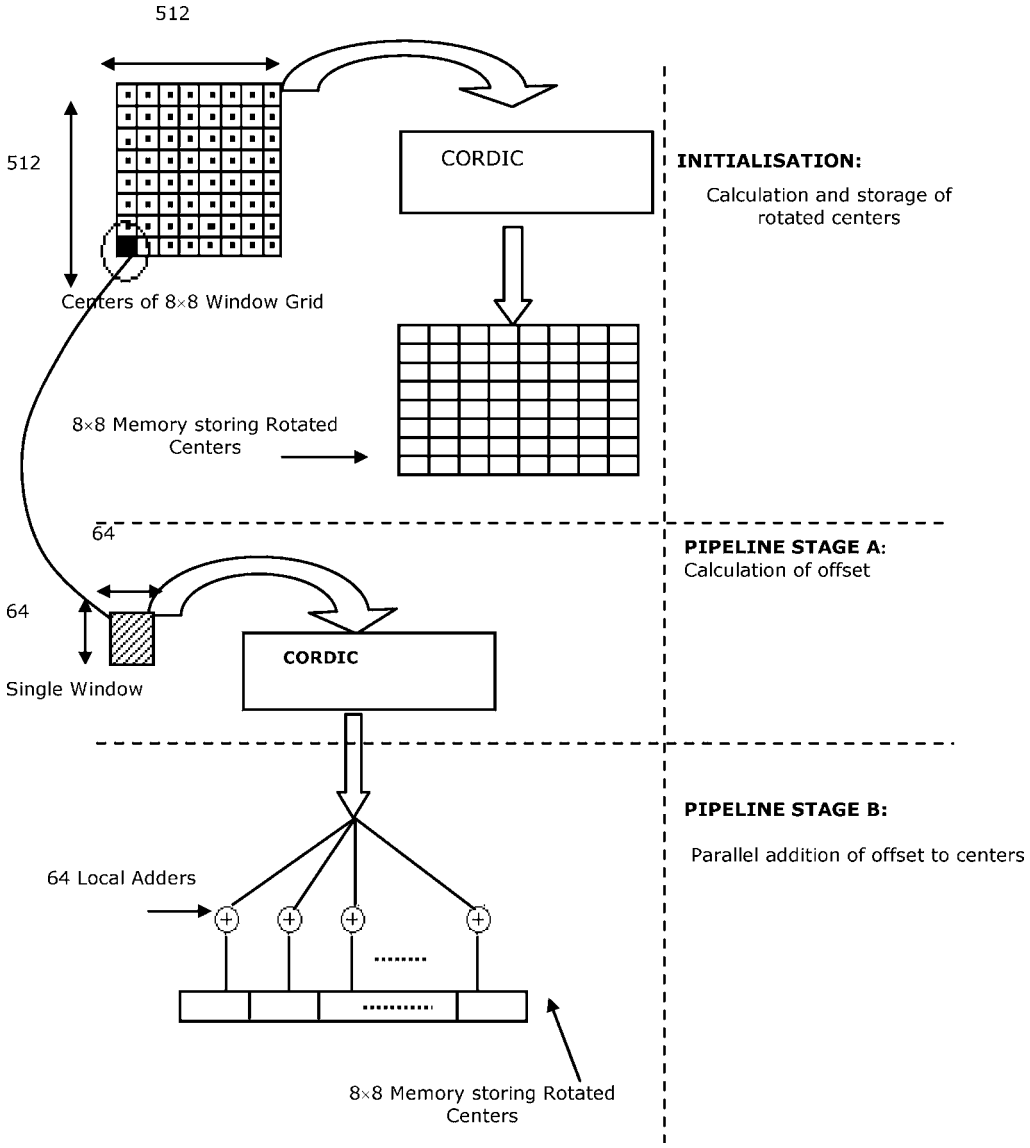

Fig. 1 *Illustration of rotation method [7]*

rotated positions of the four centres of each layer, and subsequently generating the rotated positions of all the centres through various addition combinations of the 12 centres. This has effectively reduced the number of CORDIC computations from 64 to 12.

As described [7], having a larger dimension of window grid can lead to increased performance, as there will be a lesser number of offset computations in Pipeline Stage A (Fig. 1). This implies that the proposed method can employ a larger number of hierarchical layers to improve the performance. However, increasing the number of hierarchical layers necessitates a larger number of additions to obtain all the rotated window centres in the initialisation stage. Hence, the choice of the number of hierarchical layers affects the overall performance. This will be discussed in more detail in Section 6.1.

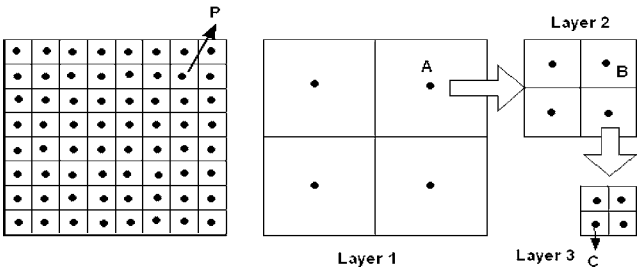

Fig. 2 *Hierarchical rotation of centres*

3.2 Exploiting symmetrical characteristics to reduce CORDIC computations

We can further reduce the number of CORDIC computations in the initialisation stage by exploiting the fact that the coordinates of the window centres in the hierarchical layers are symmetrical about the coordinate axes. Hence, in hierarchical layers with the same number of horizontal and vertical pixels, we only need to perform the CORDIC computations on a single representative centre of the layer. The remaining three image centres can be inferred by exploiting the symmetrical characteristics of the coordinate system. In the case of hierarchical layers with a different number of horizontal and vertical pixels (rectangular images), two CORDIC computations are required. It is noteworthy that the inference of the remaining window centres requires at most a simple negation. Fig. 3 illustrates this notion of inferring the rotated window centres based on the symmetrical characteristics of the image. For example, by computing (x', y'), the rotated position of (x, y), we can easily infer the rotated positions of $(x, -y)$, $(-x, -y)$

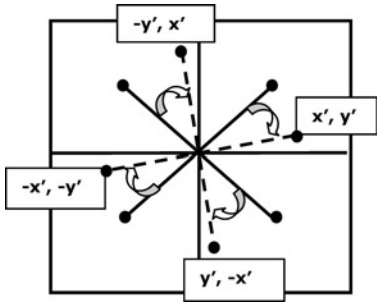

Fig. 3 *Inferring the rotated pixel values based on the image symmetry*

and $(-x, y)$. This has led to a further reduction in the number of CORDIC computations from 12 to 3.

3.3 Generation of the pixel offsets

In this section, we propose a hardware efficient technique to reduce the computational complexity of Pipeline Stage A (Fig. 1). The offsets of each pixel in the 64×64 window are computed using the iterative CORDIC computations [7]. However, if the pixel coordinates are provided in a consecutive order for the offset computations, it is sufficient to calculate the offset of just one pixel, and increment the result with a regular value (of $\sin \phi$ or $\cos \phi$) to obtain the offsets of the neighbouring pixels. The explanation to this can be obtained by analysing (3) and (4) for input values (x, y), $(x+1, y)$ and $(x, y+1)$, as neighbouring pixels differ by exactly 1 unit. The flowchart shown in Fig. 4 elucidates the process. This reduces the 64^2 CORDIC computations to mere additions.

4 System overview

The overall rotation process can be broadly divided into the generation of rotated centres and generation of offsets, running on similar lines [7]. For illustration purposes, we fix the number of hierarchical layers to three, and we assume that the hierarchical layers have an equal number of horizontal and vertical pixels. Fig. 5 depicts the main computations involved. The rotated position of a single window centre from each hierarchical layer is first computed using CORDIC. Subsequently, the rotated values of the remaining window centres in the hierarchical layers are inferred through the 'Symmetry-based Inference Unit'. As discussed earlier, this requires at most a simple negation operation. The outputs of this unit are added in different combinations to generate the 8×8 rotated-positions of the window grid centres. These values are stored in the memory to be used in the next stage.

To generate the pixel offsets, the sine and cosine value of the angle are obtained using the CORDIC engine. Then, rotation is performed on the first pixel of the window using the CORDIC engine to generate X' and Y'. Subsequent offsets are obtained by incrementally adding the sine and cosine values to the previously computed X' and Y' as described in Fig. 4. As these offsets are

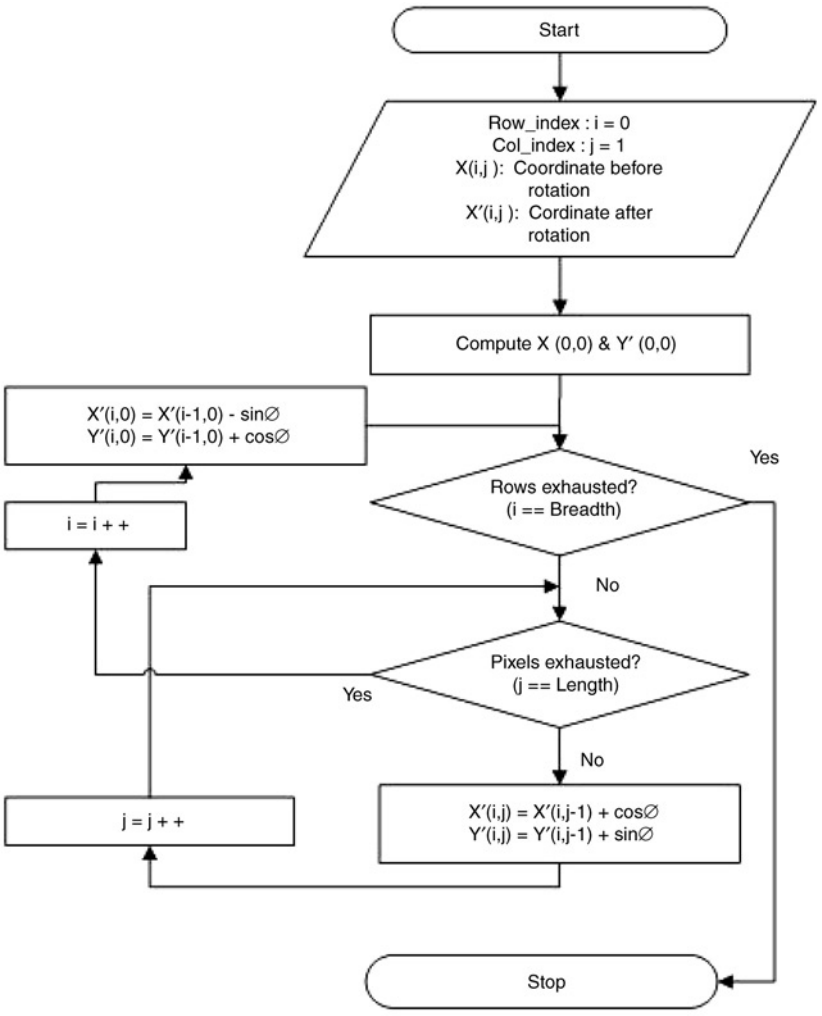

Fig. 4 *Offset computation*

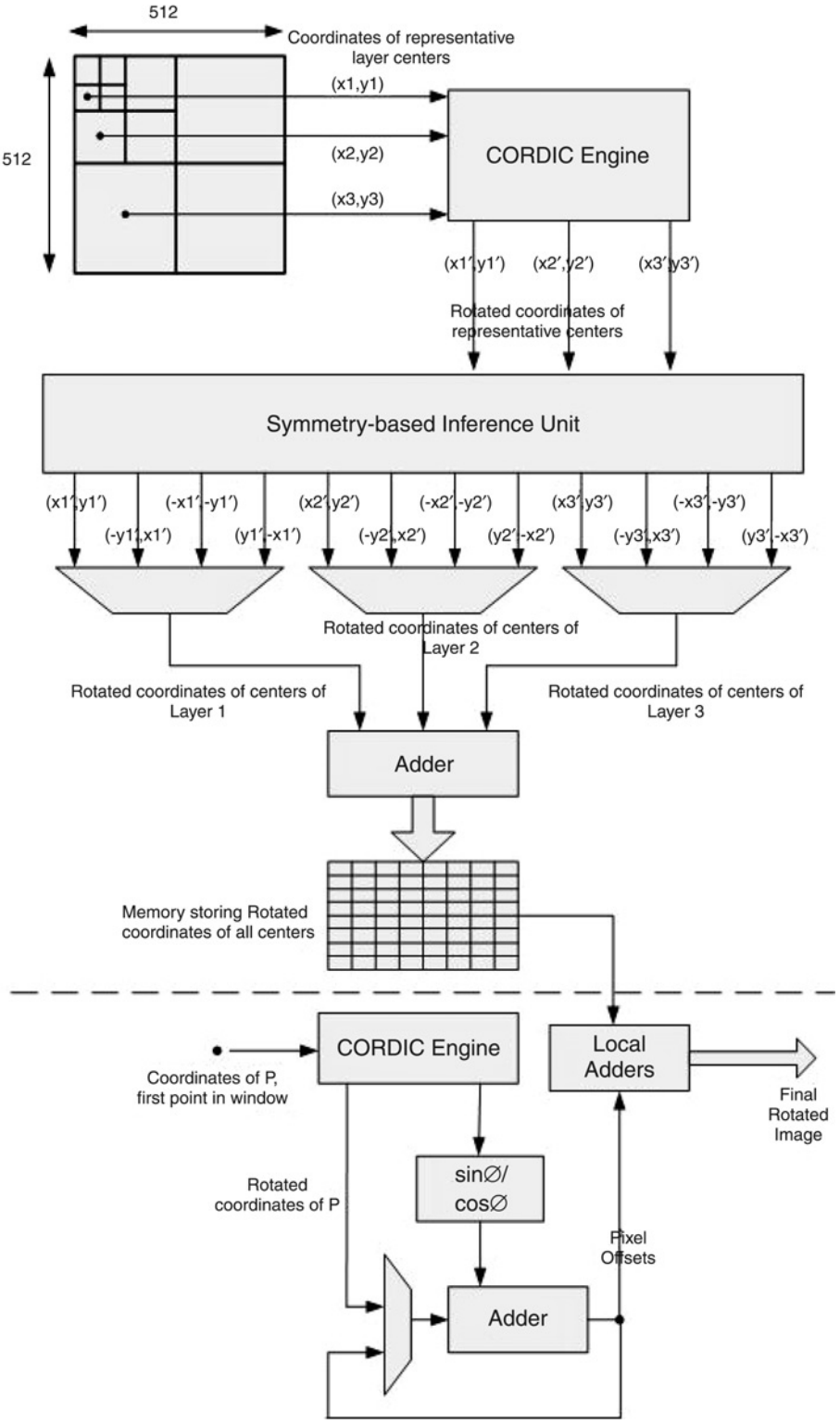

Fig. 5 *System overview of proposed image rotation engine*

generated, they are simultaneously added to all centres using the local adder array to obtain the final rotated pixel-coordinates.

4.1 Accuracy analysis of the proposed system

The accuracy of the proposed image rotation system is governed by two factors: (i) the interpolation scheme employed in the image rotation system and (ii) the error of the rotated pixel position, which are elaborated upon in the sections to follow. A certain degree of accuracy of the final rotated position is often needed in order to choose certain interpolation schemes. It is worth mentioning that the proposed system can employ various interpolation schemes based on the application's requirement by utilising a CORDIC implementation with sufficient accuracy.

4.1.1 Interpolation schemes: Conceptually, an image rotation requires two steps: a coordinate system transformation followed by pixel interpolation. Interpolation is necessary when the transformed pixel positions do not coincide with the original pixel positions. New pixel values are obtained by interpolating the original pixels in the neighbourhood of the transformed pixel position.

There are many popular interpolation functions that include nearest neighbour, bilinear, bicubic polynomial, cubic spline and Gaussian [14]. The proposed system can easily accommodate any of these interpolation schemes. The choice of interpolation scheme involves three main considerations: (i) computational cost, (ii) required quality of the final image and (iii) accuracy of the rotated pixel position, each of which is discussed in what follows.

There is a direct cost consideration when choosing among these interpolation kernels. For example, the nearest neighbour requires the least number of computations as no arithmetic operations are needed. The interpolated output data point is assigned the value of the nearest sample point from the original data. As mentioned in the work of Di Bella *et al.* [15], cubic interpolation requires 40% fewer multiplies than bicubic interpolation.

The other consideration when choosing the interpolation methods is the desired accuracy of the final image. Each method has a capacity to reconstruct the sampled image with a certain degree of exactness. It can be seen that of all the interpolation methods considered, the nearest neighbour introduces noticeable noise and jagged edges [15].

Another important issue that accompanies the choice of the interpolation method is the accuracy of the rotated pixel position, which has to match the size of the interpolation kernel. In other words, if the rotated coordinate is of 1-pixel accuracy, then only the nearest neighbour method would be feasible. The other two methods require sub-pixel accuracies of the coordinate position. For example [14], to implement the Gaussian interpolation, each destination pixel is treated as an array of 65 × 65 subpixels.

4.1.2 Accuracy of rotated pixel position: In this section, we describe the quantisation errors that affect the accuracy of the final rotated coordinate position in our system. As described [15], the CORDIC algorithm inherently suffers from two types of quantisation errors, which are due to:

(i) approximation of the angle arising from finite number of micro-rotations;
(ii) usage of finite length registers in the CORDIC data path.

The first type of error is inherent in the CORDIC algorithm and is dependent on the number of iterations that are chosen. However, the second type is purely a design issue, as it is affected by the size of the finite-length registers. In the proposed approach, error would be introduced because of the large number of additions during the offset computations. The proposed technique provides the means for design exploration on the basis of accuracy and area trade-offs. Simulations for register lengths varying from 12 to 32 were performed for a 512 × 512 image. The maximum error is defined as $[\max(\text{error}_x) + \max(\text{error}_y)]/2$ where error_x and error_y are the maximum errors in the x and y coordinates, respectively. For each data-path length, this error is recorded for all angles, and the maximum of these errors is plotted on a logarithmic scale as shown in Fig. 6. It can be observed that the error decreases exponentially with increasing operand length and becomes negligible for very high operand lengths. As mentioned earlier, the choice of interpolation method is application specific and the rotation schemes proposed in this paper are independent of it. The various interpolation schemes can be adopted by altering the register length to provide for sufficient accuracy. In the remaining sections, we adopt the same interpolation scheme [7], which is the nearest neighbour interpolation method, in order to compare between the two systems.

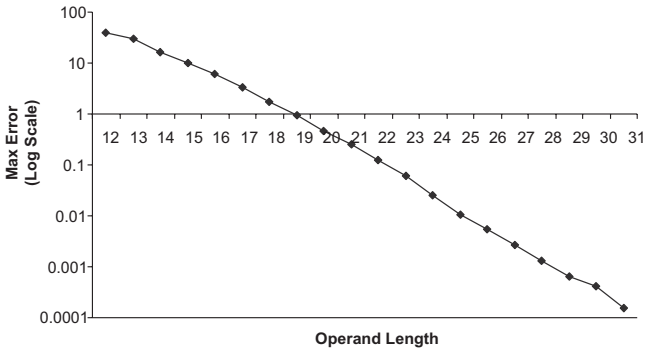

Fig. 6 *Rotation position accuracy against operand length*

5 Results and comparisons

5.1 Choice of data width

We show that any additional error that is induced due to fixed register sizes can be offset completely by extending the data width. To illustrate this, we choose the image size as 512 × 512, the interpolation scheme as nearest neighbour and to keep the error comparable with that in Ghosh and Majumdar [7], we replaced the registers and adders of length 20 [7] with those of length 25. As mentioned earlier, the number of hierarchical layers was chosen to be three to keep the window sizes comparable. Fig. 7 shows the plot of average error per pixel for the proposed method with the register length of 20 and 25. The error is obtained by comparing the results with the conventional rotation method that performs the CORDIC operations on each pixel in the 512 × 512 image. The outputs (rotated x and y coordinates) were compared with the pixel-by-pixel rotation method for all angles from 0° to 45° with 1° resolution.

5.2 Comparative analysis

In this section, the proposed rotation engine is compared with the design proposed by Ghosh and Majumdar [7] in terms of design considerations, computational complexity, latency and area of overall chip for a 512 × 512 image. A 12-iteration CORDIC engine is used. The data width of the adders-registers used in the CORDIC engine for the method proposed by Ghosh and Majumdar [7] and the proposed method are 20 bit, and 25 bit, respectively. The outputs of CORDIC in Pipeline Stage A (Fig. 1) are truncated to 10 bits prior to the additions in Pipeline Stage B. So, the data widths of the local-adders in both methods are 10-bits wide. An 8 × 8 window grid is employed, which gives rise to 64 window grid centres and window size of 64 × 64. To obtain this configuration, a hierarchy of three was chosen for the proposed method. The latency estimates are obtained from the Passport 0.35 Micron standard-cell library [16].

5.2.1 Computational complexity: The entire rotation process essentially comprises of additions and CORDIC functions and these will be analysed separately for both the methods (Figs. 1 and 5). In the work of Ghosh and Majumdar [7], 64 centres are rotated during the initialisation phase and this accounts to 64 CORDIC operations. In the proposed method, three centres are rotated and

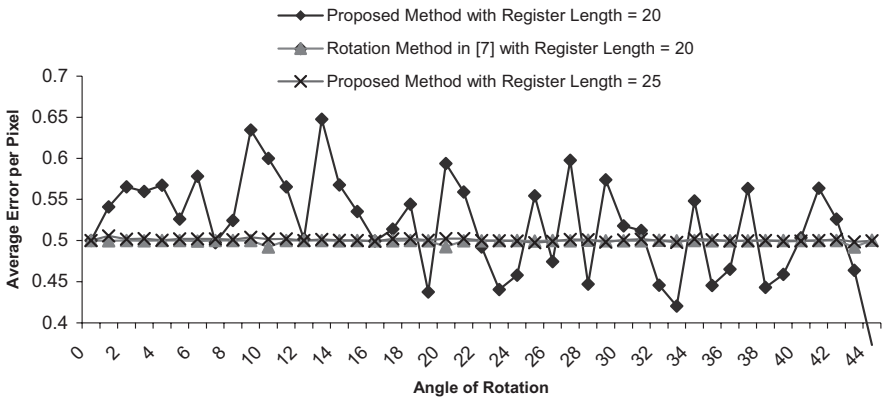

Fig. 7 *Higher data width to hide extra quantisation errors*

subsequently two additions are performed for every rotated centre. Hence, a total of three CORDIC computations and 128 additions are performed in the initialisation stage. For the generation of the offsets, the technique proposed by Ghosh and Majumdar [7] performs a rotation for each of the 64×64 pixels in the offset-window, accounting to $64^2 = 4096$ CORDIC operations. In our proposed method, one CORDIC computation is performed to rotate the first pixel of the offset-window and another CORDIC computation to obtain the sine and cosine values of the input angle. Subsequently, the remaining 4095 offsets are obtained through additions. Once the offsets and the rotated centres are generated, the final phase is similar for both methods, where each offset is added to all the rotated centres using local-adders to generate 64 rotated pixels simultaneously. The number of different computations involved in each stage are summarised in Table 1 and it can be seen that there are fewer CORDIC operations in the proposed method.

5.2.2 Latency comparison: We begin by estimating the latency of the CORDIC computations for the method of Ghosh and Majumdar [7] and the proposed method. It is assumed that all additions are implemented using carry look ahead (CLA) adders and the latency estimates are in terms of full adder latency (T_{FA}). The estimated latency for an n-bit CLA addition is $(\log_2 n)T_{FA}$. Hence, a 10-bit add-operation is estimated to take $3.32T_{FA}$, 20-bit add about $4.32T_{FA}$ and a 25-bit addition takes $4.64T_{FA}$. A CORDIC engine comprises addition and shift operations. Assuming that the shift operations are carried out through MUX-based barrel shifters, a maximum shift of 12-positions warrants four levels of MUX. So the latency attributed to the shift operation is $4 \times T_{MUX}$, where T_{MUX} is the latency of a $2-1$ multiplexer. A 12-iteration CORDIC operation involves 12 additions and 12 shifts and delay due to use of registers. We assume that T_{F-F} is the latency of a register. We can also assume that $T_{F-F} \simeq T_{FA}$ and $T_{MUX} \simeq 0.65T_{FA}$ [17]. Based on these values, the CORDIC computational latency for the method in Ghosh and Majumdar [7] is $96T_{FA}(T_{CORDIC-[7]} = [12 \times (4.32T_{FA} + T_{F-F} + 4 \times T_{MUX})])$ whereas that of the proposed method is $99T_{FA}(T_{CORDIC-NEW} = [12 \times (4.64T_{FA} + T_{F-F} + 4 \times T_{MUX})])$.

Hence, the initialisation latency for the method in Ghosh and Majumdar [7] is $I_{[7]} = 64 \times 96 = 6144T_{FA}$, whereas that of the proposed method is $I_{new} = 3 \times 99 + (64 \times 2 \times 4.64) = 891T_{FA}$. The initialisation stage of the proposed method is about seven times faster now, as the number of CORDIC operations is reduced. Referring back to the pipeline shown in Fig. 1, the overall latency [7] is $I_{[7]} + (642 \times T_{CORDIC-[7]}) = 399\,360T_{FA}$. For the proposed method, the overall latency is given by $I_{new} + [(642 - 1) \times 4.64] + (2 \times 99) = 19\,199T_{FA}$.

The latency estimates take into account the pipelined processing of the two methods. For the method in Ghosh and Majumdar [7], the CORDIC computations in Pipeline Stage A are done in parallel with the additions in Pipeline Stage B (Fig. 1). The latency of CORDIC in Pipeline Stage A is more time-consuming than the latency of the additions in Pipeline Stage B. Hence, the throughput of the pipelined implementation in Ghosh and Majumdar [7] is governed by the latency of CORDIC, and this is used for the pipeline latency estimates. Similarly, for the proposed method, the addition with 25-bit operands to generate offsets is done in parallel with the addition of offsets and rotated centres using 10-bit operands. The former latency is therefore considered in the performance evaluations.

Table 1: Area, latency, computational complexity comparisons

	Method in Ghosh and Majumdar [7]		Proposed method	
Computational latency (T_{FA})	399 360		19 199	
Area (A_{FA})	3817		3951	
Computational complexity	CORDIC	Addition	CORDIC	Addition
Initialisation	64	—	3	64×2
Pipeline A	64^2	—	2	$64^2 - 1$
Pipeline B	—	64^3	—	64^3

Table 1 summarises the performance findings and it is evident that an overall speedup of ~20× is possible over the technique proposed by Ghosh and Majumdar [7].

5.2.3 Area comparison: The area of the CORDIC engine used in Ghosh and Majumdar [7], which has three 20-bit adders and 20-bit registers (for each X, Y and Z datapath), a pair of 20-bit wide barrel shifters and a memory table, is estimated to be $(20 \times 3 \times (2A_{FA} + A_{F-F})) + (20 \times 2 \times 4 \times A_{MUX}) + (20 \times 12A_{F-F})$, where A_{FA}, A_{F-F} and A_{MUX} correspond to the area of a 1-bit full adder, flip-flop and $2-1$ MUX, respectively. A 1-bit CLA is estimated to occupy twice the area of a 1-bit serial adder. The memory is assumed to be implemented as flip-flops and for a 12 iteration CORDIC, 12 arctan values are stored. This requires an area of $12 \times 20A_{F-F}$. It can also be inferred that $A_{F-F} \simeq 1.13A_{FA}$ and $A_{MUX} \simeq 0.47A_{FA}$ [16]. Hence, the CORDIC engine in Ghosh and Majumdar [7] is estimated to occupy about $534A_{FA}$. Similarly, the area of the CORDIC engine used in the proposed method is about $668A_{FA}$.

The architecture in Ghosh and Majumdar [7] primarily comprises a CORDIC engine and 64^2 pairs of local-adders. In the proposed method, there is a CORDIC engine, a pair of adder-units to generate the rotated centres and the offsets and the 64^2 sets of local-adders. The memory requirement to store the rotated centres is estimated to be about $64 \times 10 \times A_{FF} = 723A_{FA}$. All the local-adders in both methods are implemented as 10-bit CLAs. Assuming there are 64 pairs of them (one for each centre's x and y coordinates), the area is estimated to be $64 \times 10 \times 2 \times 2A_{FA}$. The adder-unit for generating the offsets in the proposed technique comprises 25-bit CLAs (area $= 100A_{FA}$).

The area estimates are also summarised in Table 1. It can be seen that the areas of the two architectures are comparable with the proposed method showing a marginal increase of about 3%.

6 Extending to higher-resolution images

In this section, we introduce a new parameter called Hierarchy (or h) that enables us to study the proposed method across a wide range of area–time measures. A very high-level analysis is done to derive expressions for latency and area of the rotation engines in terms of h that would help in making an informed choice of the hierarchy. This is followed by a comparative study with the reference model, keeping the focus on high-resolution images.

6.1 Choice of hierarchy

The parameter h refers to the number of layers that the image is broken into before the proposed techniques are applied. The extreme cases are $h = 0$ and $h = \log_2 m$ where m refers to the length of the square image.

(i) $h = 0$: With no hierarchy, the initialisation phase is eliminated, as the image centre is the only centre to be rotated (with respect to itself and hence redundant) and the entire image becomes the window for offset generation. Using the proposed engine, the computations involved would be only additions with 2 CORDIC operations. One of the main limitations of this set-up is that it cannot be parallelised, because pixels need to be fed in successive order (Fig. 4). In addition, larger window sizes will lead to higher quantisation error.
(ii) Maximum possible h: With the maximum hierarchy of $\log_2 m$, each pixel becomes a centre and this implies the entire computation is done in the initialisation phase. Pipeline Stages A and B now become redundant.

The choice of this parameter exerts a push–pull effect on the initialisation latency and the offset generation. For example, lower levels of hierarchy will lead to larger window sizes, and this increases the latency of the offset computation. Alternatively, higher levels of hierarchy lead to more additions of the layer centres to obtain the rotated positions of the centres in the initialisation phase. It is evident that the optimal hierarchy lies somewhere in between these two extremes.

In order to obtain the optimal choice of hierarchy given an image of size $m \times m$, we first represent the overall rotation latency estimate in terms of number of additions for various hierarchies. In order to derive these figures, first, the number of CORDIC and addition computations contributing to the latency of the engine are listed. Then, the latency of the CORDIC is estimated to be k times that of an add operation.

Derivation of k: To derive k, we assume the requirement of 1-pixel accuracy of the rotated position after truncation, as mapping is done onto a single pixel and no interpolation is performed. For this, $\log_2 m$ iterations would suffice within a CORDIC operation, each of which mainly comprise one add one shift operation, apart from the register delays. Assuming that barrel shifters are used for shifting, implemented as a tree of multiplexers and by obtaining the relative delay of a $2-1$ MUX with respect to a 1-bit full adder, we can estimate the latency of the CORDIC engine to be $1.6 \log_2 m$ times that of an add operation. To obtain this value, we also assume that the size of registers used for addition is $n + \log_2 n$, where n is the number of iterations.

Table 2 shows the expression for latency of rotation (in number of adds) for the proposed method in terms of hierarchy h and image size $m \times m$. It also shows the latency for the method in Ghosh and Majumdar [7] with fixed window size $a \times a$.

Table 2: Latency expression as function of hierarchy

	Method in Ghosh and Majumdar [7]		Proposed method	
	CORDIC	Addition	CORDIC	Addition
Initialisation	$(m/a)^2$	—	$h+2$	$4^h(h-1)$
max (Pipeline Stage A, Pipeline Stage B)	a^2	—	—	$(m/2^h)^2$
Total latency (number of adds)	$[(m/a)^2 + a^2]k$		$(h+2)k + 4^h(h-1) + (m/2^h)^2$	

$m \times m$: image size; k: $1.6 \log_2 m$; $a \times a$: window size; h: hierarchy

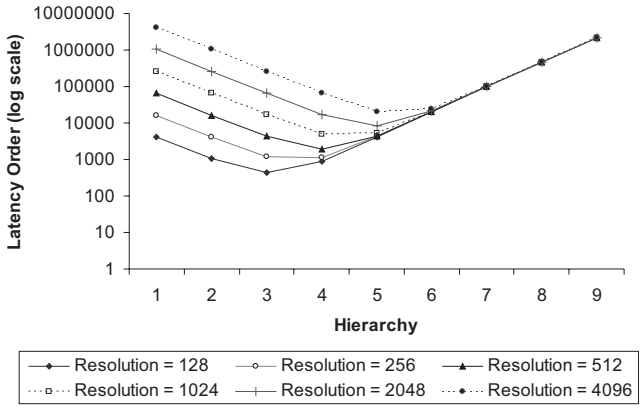

Fig. 8 *Choice of hierarchy*

Table 3: Optimal hierarchy for rotation of an $m \times m$ image

Image resolution, m	Optimal hierarchy
128	3
256	4
512	4
1024	4
2048	5
4096	5
8192	6
16 384	6
32 768	7
65 526	7
131 072	8

Fig. 8 shows a plot of the latency (in number of adds) in logarithmic scale plotted against varying hierarchies. It illustrates that extreme hierarchies do not lead to the most time-efficient implementation. Table 3 shows the optimal hierarchy for images of varying image resolution. This corresponds to the hierarchy with the minimum latency for various resolutions in Fig. 8.

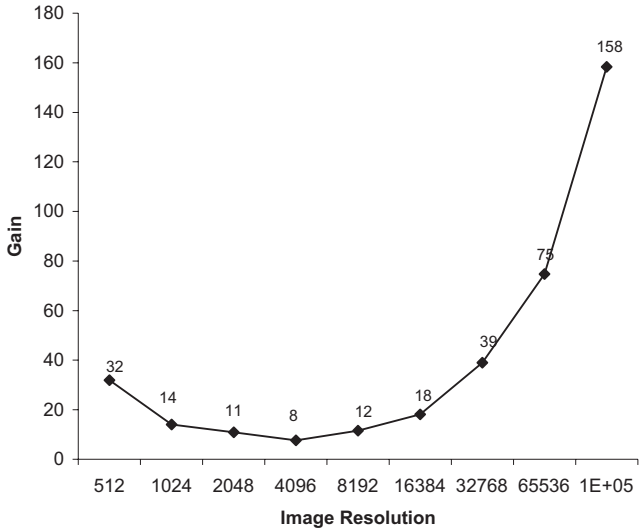

Fig. 9 *Gain of proposed method over existing method*

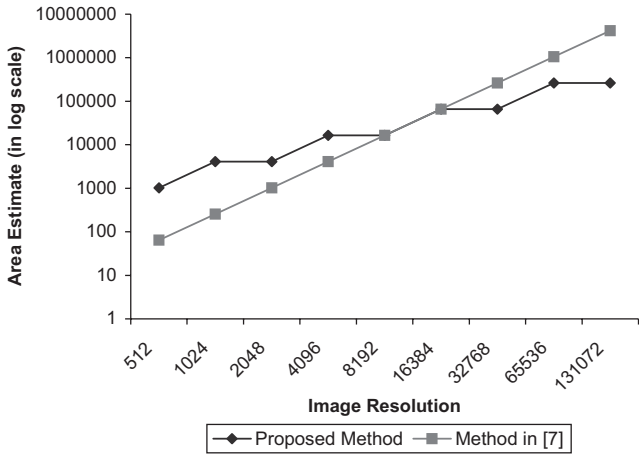

Fig. 10 *Area estimates for varying image resolution*

6.2 Latency estimate comparison

'Gain' is defined as, the ratio of the latency of the method in Ghosh and Majumdar [7] to the optimal latency of the proposed method. The latency measures are obtained by referring to Table 3. Fig. 9 shows a graph of gain plotted against varying image resolutions. It can be observed that there is a definite gain for all resolutions. The gain becomes more pronounced for images of higher resolutions as it increases in an exponential fashion. It is worth noting that in the initialisation phase, the additions for obtaining the rotated centres are assumed to be sequential. In order to further increase the gain, the local adders can be employed in the initialisation stage to perform the additions for obtaining the rotated centres in parallel. Also, the proposed system can employ improved CORDIC implementations to further increase the performance gain. However, as the majority of the operations in the proposed system consist of additions, the improvements in CORDIC will not lead to a notable performance gain.

6.3 Effect of changing hierarchy on area

The memory area to store the centres in the initialisation stage increases with the number of hierarchies. Furthermore, as the proposed architecture requires every centre to have a local adder for addition of the offset in Pipeline Stage B (Fig. 1), the adder area also grows proportionally. The number of such adders is $(m/a)^2$ and $(m/2^h)^2$ for the method in Ghosh and Majumdar [7] and the proposed method, respectively, which is the same as the number of memory elements to store the centres. Treating the area of one memory element with its local adder together as a unit α, we plot the area in α for both methods in Fig. 10. For the proposed method, the area corresponding to the configuration giving optimal performance is considered. Both methods use one CORDIC engine each and this area is negligible compared with the overall area consumed by the memory and the local adders.

As can be observed in Fig. 10, the proposed engine is more favourable in terms of area and latency for high image resolutions. It is noteworthy that this graph takes into account the optimal hierarchy and hence the areas look unattractive for lower resolutions. For cost sensitive solutions, a different hierarchy that leads to a lesser optimal solution can be chosen to satisfy the area constraint and at the same time sustain a considerable gain in

performance. For example, the rotation engine described in Section 5 is 20× faster with almost no increase in area. Further reduction of the area can be achieved by reusing a smaller set of adders for the computations.

7 Conclusion

We have proposed novel schemes to notably minimise the number of CORDIC operations to accelerate the process of image rotation. We have employed hierarchical techniques to better manage the required number of additions and CORDIC operations, thereby providing for area–time flexibility in the hardware realisation. It has been demonstrated that this approach provides for substantial speedup with an acceptable increase in area cost when compared with existing techniques. We have also analysed the effect of hierarchy on the computational efficiency so as to identify the optimal hierarchy for a given image resolution. In the case of the 512×512 image, the hierarchy of four was shown to be optimal although the maximum possible hierarchy is nine. We have shown that the increase in the data width to accommodate the total number of additions incurs a marginal increase in the overall hardware cost. The proposed technique is highly scalable and lends well for high-speed rotation of high-resolution images.

8 References

1 Krymski, A., Van Blerkom, D., Andersson, A., Bock, N., Mansoorian, B., and Fossum, E.R.: 'A high speed, 500 frames/s, 1024×1024 CMOS active pixel sensor'. Symp. on VLSI Circuits, Digest of Technical Papers, 1999, pp. 137–138

2 Zarandy, A., Dominguez-Castro, R., and Espejo, S.: 'Ultra-high frame rate focal plane image sensor and processor', *IEEE Sens. J.*, 2002, **2**, (6), pp. 559–565

3 Koppel, D., Wang, Y.F., and Lee, H.: 'Automated image rectification in video-endoscopy'. Proc. Int. Conf. on Medical Image Computing and Computer-Assisted Intervention, Utrecht, The Netherlands, 2001

4 LeMoigne, J.: 'Parallel registration of multisensor remotely sensed imagery using wavelet coefficients'. Proc. 1994 SPIE Wavelet Applications Conference, Orlando, 1994, pp. 432–443

5 Chalermwat, P., and El-Ghazawi, T.: 'Multi-resolution image registration using genetics'. IEEE Int. Conf. on Image Processing, 1999, vol. 2, pp. 24–28

6 Volder, J.E.: 'The CORDIC trigonometric computing technique', *IRE Trans. Electron. Comput.*, 1959, **EC-8**, pp. 330–334

7 Ghosh, I., and Majumdar, B.: 'VLSI implementation of an efficient ASIC architecture for real-time rotation of digital images', *Int. J. Pattern Recognit. Artif. Intell.*, 1995, **9**, pp. 449–462

8 Bhandarkar, S.M., and Yu, H.: 'A VLSI implementation of real-time image rotation'. Proc. Int. Conf. on Image Processing, 1996, vol. 2, pp. 1015–1018

9 Eggers, D.D., and Ackerman, E.: 'High speed image rotation in embedded systems', *Comput. Vis. Image Underst.*, 1995, **61**, pp. 270–277

10 Takagi, N., Asada, T., and Yajima, S.: 'Redundant CORDIC methods with a constant scaling factor for sine and cosine computation', *IEEE Trans. Comput.*, 1991, **40**, pp. 989–995

11 Lee, J.A., and Lang, T.: 'Constant factor redundant CORDIC for angle calculation and rotation', *IEEE Trans. Comput.*, 1992, **41**, pp. 1016–1025

12 Wang, S., Piuri, V., and Schwartzlander, E.E. Jr.: 'Granularly-pipelined CORDIC processors for sine and cosine generators', *IEEE Trans. Comput.*, 1997, **46**, pp. 1202–1207

13 Ghosh, I., and Majumdar, B.: 'Design of an application specific VLSI chip for image rotation'. 7th Int. Conf. on VLSI Design, 1994, pp. 275–278

14 Wallis, J.W., and Miller, T.R.: 'An optimal rotator for iterative reconstruction', *IEEE Trans. Med. Imag.*, 1997, **16**, pp. 118–123

15 Di Bella, E.V.R., Barclay, A.B., Eisner, R.L., and Schafer, R.W.: 'A comparison of rotation-based methods for iterative reconstruction algorithms', *IEEE Trans. Nucl. Sci.*, 1996, **43**, pp. 3370–3376

16 Avant! Passport 0.35 micron, 3.3 volt SC Library CB35OS142, March 2000

17 Yu, Y.H.: 'The quantization effects of the CORDIC algorithm', *IEEE Trans. Signal Process.*, 1992, **40**, pp. 834–844