IEICE TRANS. INF. & SYST., VOL.E95–D, NO.11 NOVEMBER 2012

PAPER

An Improved Look-Up Table-Based FPGA Implementation of Image Warping for CMOS Image Sensors

Se-yong RO[†,††], **Lin-bo LUO**[†,††a)], *Nonmembers*, *and* **Jong-wha CHONG**[††b)], *Member*

SUMMARY Image warping is usually used to perform real-time geometric transformation of the images captured by the CMOS image sensor of video camera. Several existing look-up table (LUT)-based algorithms achieve real-time performance; however, the size of the LUT is still large, and it has to be stored in off-chip memory. To reduce latency and bandwidth due to the use of off-chip memory, this paper proposes an improved LUT (ILUT) scheme that compresses the LUT to the point that it can be stored in on-chip memory. First, a one-step transformation is adopted instead of using several on-line calculation stages. The memory size of the LUT is then reduced by utilizing the similarity of neighbor coordinates, as well as the symmetric characteristic of video camera images. Moreover, an elaborate pipeline hardware structure, cooperating with a novel 25-point interpolation algorithm, is proposed to accelerate the system and reduce further memory usage. The proposed system is implemented by a field-programmable gate array (FPGA)-based platform. Two different examples show that the proposed ILUT achieves real-time performance with small memory usage and low system requirements.

key words: *Image warping, look-up table, perspective transformation, panorama unrolling, FPGA*

1. Introduction

Image warping is the spatial transformation of an image based on a given geometric relationship [1]. Warping is usually used to perform image geometric transformations in surveillance video cameras images [2], robot vision [3], and vision-based intelligent transportation systems [4], such as lens distortion correction [5], and panorama unrolling for omnidirectional images taken by a catadioptric camera [6] or a fish-eye camera [7]. For these mobile applications, real-time performance of approximately 30 frames per second is mandatory. However, distortion correction and most other image warping include complex calculations and consequently incur a large computational cost. Therefore, the implementation of real-time image warping is a very critical design issue.

Many studies have been conducted on this issue. Several algorithms have reduced the number of division operations that exist in the software warping process, using special optimization methods to reduce or remove the division involved in perspective transformation [8]–[10]. However, all of the software algorithms have to be supported by a processor and an operating system. Therefore, they are not suitable for low-cost and small-size mobile systems, and it remains necessary to design a hardware-based method. The hardware implementations can be categorized as on-line calculation and look-up table (LUT)-based methods. The on-line calculation does not require additional memory to store the LUTs; whereas it requires a system with high-quality specification for support, such as a general processor, graph processor unit, or digital signal processor. For example, Huggett realized a perspective correction for SOC using ninth-order on-line polynomials [11], however the system required a 32-bit processor for support. For mobile applications, this method is not usually a suitable choice. The LUT-based algorithm can easily realize real-time performance, especially for video cameras, which are needed to iterate some geometric transformations using fixed parameters. However, this approach requires additional memory storage for the LUT. For example, Mattson and Oh proposed two field-programmable gate array (FPGA)-based fast image warping [12], [13], and Mohan and Chen proposed a LUT-based framework for fish eye correction [7] and panorama unrolling [3] respectively. The LUTs in these four studies are not compressed and, as a result, use off-chip memory or will use off-chip memory when image size increases. The use of off-chip memory will increase the control complexity, latency, and bandwidth consumption.

The purpose of this paper is to use image warping to perform geometric transformation for video camera images, such as distortion correction, viewpoint changes, and image unrolling for omnidirectional images. Considering that processing speed and power consumption are more important for mobile applications, especially for high resolution images, in this paper, we adopt a LUT-based approach to perform image warping.

Our contribution is that, in order to overcome the shortage of the LUT-based approach, we propose an improved look-up table (ILUT) which utilizes the characteristics of practical video sequences to compress the general LUT, making it possible to use on-chip memory. We propose four optimization algorithms to compress the LUT: one-step transformation; Δ-algorithm; symmetric principle; and 25-point interpolation. The algorithms will be described in detail later.

The remainder of this paper is organized as follows. Section 2 gives a brief background of LUT-based image warping. Section 3 introduces our optimization strategy for the existing LUT-based algorithm. Section 4 presents the

Manuscript received March 14, 2012.
Manuscript revised July 19, 2012.

[†]The authors are with the Faculty of Mech. & Elec. Inf., China University of Geosciences, Wuhan, 430074 China.
[††]The authors are with Department of Elec. & Comp. Engineering, Hanyang University, Seoul, 180–8585 Korea.
a) E-mail: luolinbo@cug.edu.cn
b) E-mail: jchong@hanyang.ac.kr

DOI: 10.1587/transinf.E95.D.2682

details of our proposed system and the implementation of each module of the system. Section 5 shows the efficiency of the ILUT-based scheme through two different examples: a top view system and panorama unrolling of an image captured by a central catadioptric camera. Section 6 concludes the study.

2. Background of LUT-Based Image Warping

The warped image is the result of resampling the input image based on a given geometric relationship between the coordinate system of the input image (u, v) and the output (x, y) [1]. Distinguished by the manner in which the data flows during transformation, image warping is classified as either forward or inverse mapping (FM or IM), which yields two different forms of expression as follows:

$$\text{FM:} \quad (u, v) = (F_u(x, y), F_v(x, y)) \tag{1}$$

$$\text{IM:} \quad (x, y) = (F_x(u, v), F_y(u, v)) \tag{2}$$

where corresponding locations on the input and output images have coordinates (x, y) and (u, v), respectively, defined by the forward mapping functions F_u and F_v or the inverse-mapping functions F_x and F_y.

2.1 Forward Mapping and Inverse Mapping

The forward mapping algorithm has a shortcoming in that the mapping output results in holes. Given the example of perspective transformation, as shown in Fig. 1, holes exist in the output image, round which no corresponding mappings are present. Therefore, it is difficult to interpolate the pixel values.

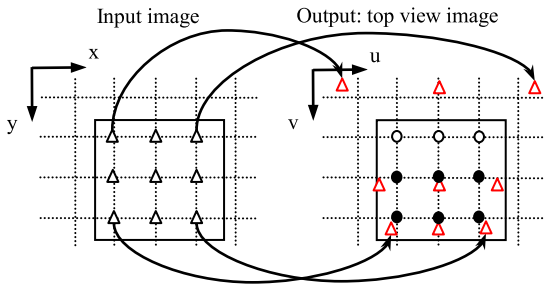
Inverse mapping operates in screen order, projecting each output coordinate to the input image via F_x and F_y. The value of the data sample at that point is interpolated and copied onto the output pixel. The advantage of inverse mapping is that it guarantees that all output pixels are computed. Unlike forward mapping, there are no existing holes in the input image. As shown in Fig. 2, every mapping coordinate of the inverse transformation in the input images is located among the input image pixels. Since there are real input pixels around the inverse mapping coordinates, interpolation can be used to generate a value for the output.

In this paper, inverse mapping is adopted for the convenience of interpolation.

2.2 General LUT-Based Image Warping

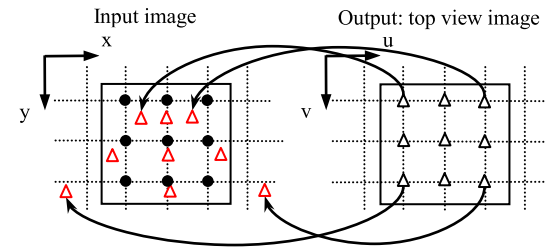
Most warping algorithms use a pair of polynomials to describe the coordinate mapping. For example, the inverse mapping of distortion correction and perspective transformation can be expressed using (3) and (4), respectively [1].

$$(x, y) = \left(\sum_{i=0}^{N} \sum_{j=0}^{N-i} a_{ij} u^i v^j, \sum_{i=0}^{N} \sum_{j=0}^{N-i} b_{ij} u^i v^j \right) \tag{3}$$



△ Pixel coordinates of input image
△ Corresponding coordinates of forward mapping in output image
● The pixels easily interpolated in top view image
○ Holes in top view image (which are difficult to be interpolated)

Fig. 1 Forward mapping of the perspective transformation.



△ Pixel coordinates of output image
△ Corresponding coordinates of inverse mapping in input
● Pixels of input image

Fig. 2 Inverse mapping of the perspective transformation.

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a_{22}a_{33} - a_{23}a_{32} & a_{13}a_{32} - a_{12}a_{33} & a_{12}a_{23} - a_{13}a_{22} \\ a_{23}a_{31} - a_{21}a_{33} & a_{11}a_{33} - a_{13}a_{31} & a_{13}a_{21} - a_{11}a_{23} \\ a_{21}a_{32} - a_{22}a_{31} & a_{12}a_{31} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{4}$$

where corresponding locations on the input and output images have coordinates (x, y) and (u, v), respectively, and a_{ij} and b_{ij} are parameters determined by the mapping itself. In (4), $x = x'/w'$ and $y = y'/w'$, where w' is a scale factor.

Mapping functions are almost as diverse as applications. However, many warping procedures that can be expressed by equations require excessive computational cost. For a video camera in practical applications, the transformation is usually not changed except when the related extrinsic parameters of the camera have been changed. Therefore, once the mapping is known, we use inverse mapping to calculate the target coordinates and store them into memory in LUT format, regardless of the mapping function.

To implement such warping, an LUT linking each output pixel to a location in the input image has been introduced [14]. The basic LUT contains one entry for each pixel location in the output image where each entry is composed of the real coordinates (x, y) of the corresponding location in the input image. With a field of view (FOV) equal to

Table 1 Basic form of LUT for inverse mapping.

	u_0	u_1	...	u_{m-1}
v_0	$x_{(0,0)}, y_{(0,0)}$	$x_{(1,0)}, y_{(1,0)}$	...	$x_{(m-1,0)}, y_{(m-1,0)}$
v_1	$x_{(0,1)}, y_{(0,1)}$	$x_{(1,1)}, y_{(1,1)}$	...	$x_{(m-1,1)}, y_{(m-1,1)}$
⋮	⋮	⋮	...	⋮
v_{n-1}	$x_{(0,n-1)}, y_{(0,n-1)}$	$x_{(1,n-1)}, y_{(1,n-1)}$	...	$x_{(m-1,n-1)}, y_{(m-1,n-1)}$

$m * n$, the basic LUT of the coordinates for inverse mapping is shown in Table 1.

The LUT can be loaded into memory at the initialization stage. Each output pixel value is calculated in screen order. To calculate an output pixel value, the input coordinates (x, y) are retrieved from the LUT based on the current output pixel (u, v). If inverse mapping coordinates (x, y) are outside the input image, the output pixel is set to 0. If (x, y) are both integers, the corresponding input pixel value is directly copied to the output pixel. Otherwise, a filter is needed to interpolate the output pixel value.

2.3 Interpolation

For discrete images, pixels are assumed to be finite elements defined to lie on a discrete integer lattice. In general, the target coordinates of inverse mapping in an input image are not defined and must be interpolated. The obvious way to implement warping is through 2-D resampling. In this process, each pixel's horizontal and vertical coordinates are mapped, and the value of that pixel is then interpolated. One example is bilinear interpolation, which can anti-alias well with a reasonable cost, although the process could be accelerated by a more efficient algorithm.

For bilinear interpolation, the coordinates (u_i, v_i) are divided into two parts: integer and offset. The integer part is used to determine the location of a $2 * 2$ window for interpolation, whereas the offset is used as the weight of the interpolation. The interpolation can be operated according to (5) [15].

$$\begin{aligned} I = {} & (1 - \mathit{offset}_x) * (1 - \mathit{offset}_y) * I_{i,j} \\ & + \mathit{offset}_x * (1 - \mathit{offset}_y) * I_{i+1,j} \\ & + (1 - \mathit{offset}_x) * \mathit{offset}_y * I_{i,j+1} \\ & + \mathit{offset}_x * \mathit{offset}_y * I_{i+1,j+1} \end{aligned} \tag{5}$$

where offset_x and offset_y are the fraction of x and y respectively, and $I_{i,j}$, $I_{i+1,j}$, $I_{i,j+1}$ and $I_{i+1,j+1}$ denote the pixel value at the coordinates of $(\lfloor x \rfloor, \lfloor y \rfloor)$, $(\lceil x \rceil, \lfloor y \rfloor)$, $(\lfloor x \rfloor, \lceil y \rceil)$ and $(\lceil x \rceil, \lceil y \rceil)$.

3. Optimization Strategy for Memory Reduction and System Acceleration

Based on existing works, three possible aspects can be optimized according to the practical characteristics of video camera applications. (a) For general geometric transformation of a video camera image, main warping is usually followed by some post-processing in order to improve the image quality or making it more suitable for the human visual system, such as image shearing or enlarging. That is, there are generally no fewer than two steps of transformation that need to be performed in sequence in real-time. Since the computational costs of one step of real-time transformation are considerable, the first problem is that two or more steps result in a large increase in computation cost. (b) The second and more important problem is that the use of the basic LUT as shown in Table 1 would result in extensive memory usage. Suppose the FOV is $m * n$, the ranges of the coordinates are: $0 \le x < m$ and $0 \le y < n$. The number of bits used to code the integer parts of x and y would be no less than $\lceil \log_2 m \rceil$ and $\lceil \log_2 n \rceil$, respectively. For example, if the image resolution is $640 * 480$, $\lceil \log_2 m \rceil$ is 10 and $\lceil \log_2 n \rceil$ is 9, and the total memory usage is $480 * 640 * (9 + 10) \approx 5.57$ Mbits even if no bits are used to code the fraction parts of the coordinates. This degree of memory demand would require the use of off-chip memory. However, as we know, off-chip memory will increase the control complexity and bandwidth requirements. (c) The efficient performance of interpolation is the third problem. In bilinear interpolation, there are four subtractions, eight divisions, and three additions. Furthermore, we need to simultaneously fetch four pixel values around the pixel which need to be interpolated currently from buffer. Therefore, to achieve a pipelining performance, we need to accelerate the interpolation procedure.

Considering these three problems, we propose the following optimization algorithms.

3.1 One-Step Transformation

Since we adopt an LUT-based warping scheme, several steps can be combined into one-step transformation because a LUT can describe the geometric relationship between the input image and the final output image, which is irrelevant to the middle steps. Without the loss of generality, we can use $\mathrm{S}_1, \mathrm{S}_2, \ldots, \mathrm{S}_n$ to describe n steps of transformation. Thus, S can be considered as the cascade of a series of steps, that is $\mathrm{S} = \mathrm{S}_1\mathrm{S}_2 \cdot \mathrm{S}_n$. The details will be presented in next section together with an example.

3.2 LUT Compression Algorithm

Because an additional LUT is used instead of an on-line calculation, additional memory is required to store the LUT. If the basic LUT is not compressed, it becomes so large that it is unable to be stored in on-chip memory, and off-chip memory, such as SDRAM or flash memory, must be used. This will result in complex memory access and large bandwidth requirements.

Although Oh proposed a data-parallelization scheme that can more efficiently access off-chip memory compared to that of general LUT access [13], another alternative exists. Most video camera images are regular, and there are some characteristics that can be utilized to compress the LUT itself. The LUT can be compressed sufficient small to be store in the on-chip memory. We can then avoid the problems of

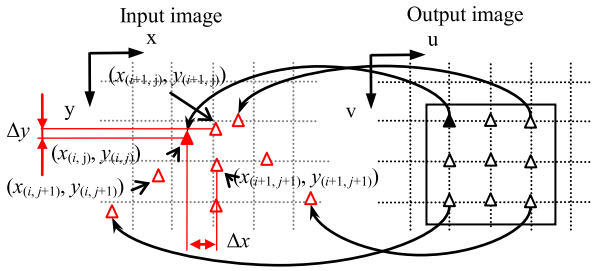


Fig. 3 Illustration of the similarity of neighbor pixels for general inverse mapping.

off-chip memory access and required bandwidth.

3.2.1 Δ-Algorithm

Figure 3 is an illustration of a general inverse mapping in which the current coordinate are $(x_{(i,j)}, y_{(i,j)})$. We can see that: (a) Δy, the difference in y between horizontal neighbor pixels, is usually very small, and (b) Δx, the difference in x between horizontal neighbor pixels, is also small. Based on these considerations, while constructing the LUT, we can only store the coordinate of the first pixel of each row and the difference between the current and previous coordinate from the second pixel of each row. When retrieving the LUT, the registers of x and y coordinates are first initialized by the coordinate values of the first pixel of each row, then we sum the difference up to the initialized values and we can obtain the current coordinate. Because the differences are small, a large amount of memory can be saved compared to the process in which the entire coordinates are stored.

Moreover, as in (6), if the differences between the maximum and the minimum vertical coordinates of every row are all smaller than a given threshold, the differences in the vertical direction can be regarded as insignificant. Therefore, it can be treated as one fixed value in a row.

$$\max_{0\le j\le n-1}\left(\max_{0\le i\le m-1}(F_{y_{ij}}(u,v)) - \min_{0\le i\le m-1}(F_{y_{ij}}(u,v))\right) \le T \quad (6)$$

where m and n are the width and height of output image, respectively; T is a threshold which decided by precision. Though this approximate operation results in some error, the error can be controlled at a proper level because the value of threshold T is adjustable.

3.2.2 Utilization of Symmetric Characteristics

In practical applications, many geometric transformations are symmetric, such as barrel distortion correction, lens distortion correction of fish-eye cameras, the panorama unrolling of central catadioptric cameras, and some perspective transformations.

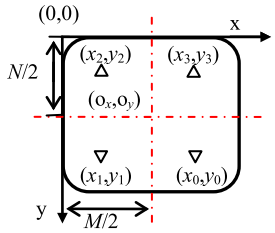


Fig. 4 Illustration of the symmetry principle.

If the coordinates of inverse mapping are symmetric, as in Fig. 4, we can calculate other mapping coordinates according to one part of the known mapping instead of storing all of the coordinates.

When the origin of the coordinates is $(0, 0)$ and the symmetric center is (o_x, o_y), if (7) or (8) hold, we can say that it is symmetric about the horizontal axis or vertical axis, respectively. If both (7) and (8) hold, the mapping is symmetric around the center.

$$y + o_y = f(x + o_x) = f(-x + o_x) \quad (7)$$

$$x + o_x = f^{-1}(y + o_y) = f^{-1}(-y + o_y) \quad (8)$$

where, $o_x = M/2$ and $o_y = N/2$; M and N are the width and height of input image, respectively.

Considering that the values of the width and height of a practical image are always an even number, the coordinates of the input image can be divided symmetrically as long as the image is symmetric. That is, if Eq. (7) holds, according to the symmetric principle, we can calculate (x_1, y_1) if we know (x_0, y_0). For the same reason, if Eq. (8) holds, we can calculate (x_3, y_3). If (7) and (8) both hold, through (x_0, y_0), we can calculate (x_1, y_1), (x_2, y_2), and (x_3, y_3). The detailed method will be illustrated in the next section. Thus, if the inverse transformation is symmetric, we can construct an appropriate coordinate system, and then we can use this characteristic to compress the LUT to 1/2, 1/4, or even 1/8 of its original size.

To avoid generating an error due to the use of the symmetric principle, a precondition should be satisfied. The precondition is that we can identify a symmetric axis or a symmetric center and make the mapping is strict symmetric about this axis or this point, respectively. If this condition does not hold, the symmetric principle cannot be utilized.

3.3 25-Point Interpolation Algorithm

Implementing floating point-based inverse mapping on a FPGA-based system poses the question of the required fixed point format which directly influences the FPGA's resource utilization, external memory usage, and memory transfer bandwidth. The integer component is defined by the maximum allowable deviation within an image. The fractional part is determined by the required subpixel accuracy of subsequent processing blocks, as well as the sensor chip itself [16].

We improve the scheme proposed by Mohan [7] and

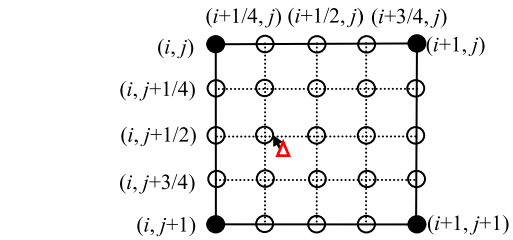


Fig. 5 Illustration of the n-point interpolation algorithm when $n = 25$.

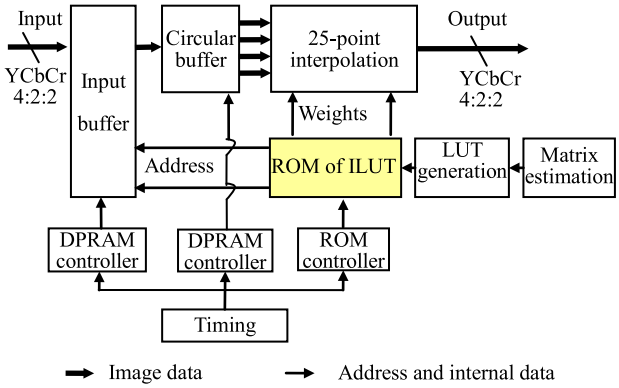


Fig. 6 The proposed ILUT-based image warping system.

propose a novel n-point interpolation algorithm. The number n is used to control the accuracy of the interpolation and can be adjusted according to the precision. The value of n should be $(2^i + 1)^2$, $i = 1, 2, 3 \ldots$. Considering that the requirements for the image quality of video camera images is not very high, a subpixel or 1/4 pixel precision is sufficient for interpolation. In this paper, $i = 2$ and so a 25-point interpolation algorithm is used, as shown in Fig. 5.

Among the $2 * 2$ interpolation window, there are 21 phantom pixels, in addition to the actual pixels. The space between the two actual pixels is 1, and the space between the two artificial pixels is 1/4. The intensity value of the phantom pixel is calculated using bilinear interpolation. Since the space between the two pixels is 1/4, the divisions (i.e., multiplications with fractions) of bilinear interpolation can be replaced by right-shifting. The value of the pixel needing to be interpolated (the red rectangle in Fig. 5) is then equal to the nearest phantom pixel or actual pixel. That is, the red triangle in Fig. 5 is assigned the value of the phantom pixel $(i + 1/4, j + 1/2)$.

Because the only division operation involved can be realized by right-shifting, this algorithm is easy for hardware implementation.

4. Proposed ILUT-Based Image Warping

According to the properties of image warping mentioned in Sect. 2, we use inverse mapping in this paper. Our focus is to compress the LUT and construct a system which centers on the compressed ILUT. In this section, we present the system structure of the ILUT-based image warping. Next, using perspective transformation as an illustration, we show how to generate the ILUT utilizing the characteristics of the images and resample the input image with the 25-point algorithm.

4.1 System Structure of ILUT-Based Image Warping

The proposed LUT-based system structure is shown in Fig. 6. The system mainly consists of five parts: Part I is to estimate the transformation function according to the warping properties; Part II is an initialization process which precalculates the corresponding pixel coordinates of inverse mapping and stores them in the form of a LUT; Part III is the timing module which generates the system timing according to the output timing; Part IV is the memory controller module which controls the reading of the LUT ROM and the writing and reading of the input buffer and the circular buffer; and Part V is the 25-point interpolation module which interpolates hole pixels.

Part I and Part II are implemented by software and are performed only once during system initialization. This design is based on the observation that the relationship between the coordinates of the output image and its corresponding coordinates in the input image are absolutely determined by the camera parameters. Therefore, once the LUT is generated and stored into the ROM at the initialization stage, the system can use it afterward and does not need to change unless the camera parameters change. Of course, the LUT can also be updated if needed. Other modules are implemented by FPGA. The entire image transformation uses a pipelining hardware structure with no more than one frame latency. The detailed design is described in the following section.

4.2 ILUT Generation

This subsection addresses problems of the ILUT generation, including the realization of the one-step transformation, the compression of the basic LUT based on the Δ-algorithm and the symmetric characteristic, and the detailed procedure of encoding and decoding.

4.2.1 Implementation of One-Step Transformation

As mentioned in Sect. 3, if we use $S_1, S_2, \ldots, S_n$ to describe n steps of the transformation, the one-step transformation can be considered to be the cascade of several steps: $S = S_1 S_2 \cdot S_n$.

Figure 7 shows an example of n-step transformations. After perspective transformation, to make the image more suitable for the human visual custom, clipping is necessary. Therefore, as in Fig. 7 (a), 3 steps are required: perspective mapping, shearing and enlarging. Generally, 3 LUTs are required to perform a 3-step transformation; however, all

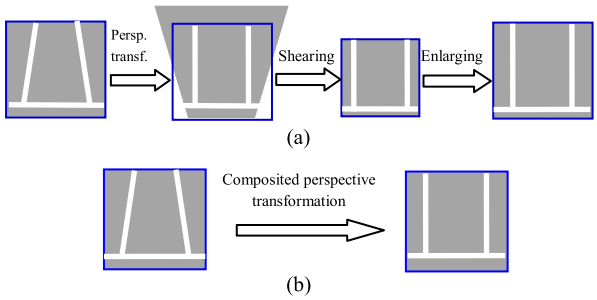


Fig. 7 Illustration of one-stop transformation: (a) Original perspective with shearing and enlarging; (b) Composited one-stop perspective transformation.

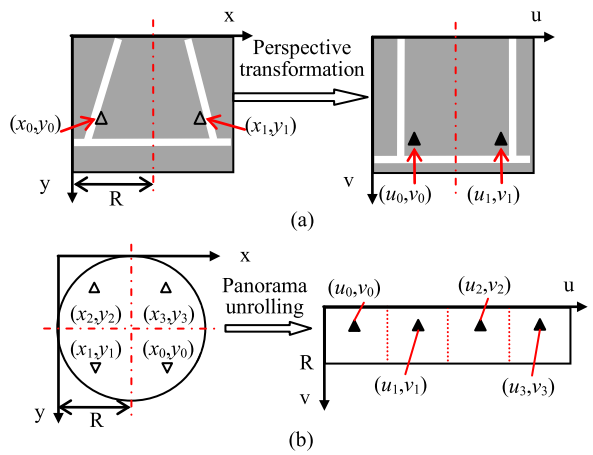


Fig. 8 Illustration of the symmetry characteristics of two geometry transformations: (a) Perspective transformation; (b) Panorama unrolling of omnidirectional images [2].

three transformations are connatural spatial transformations. Thus, we can combine these transformations into a perspective transformation and store it in the form of a combined LUT.

Suppose that the inverse perspective transformation matrix, shearing matrix and enlarging matrix are H_1, H_2, and H_3, respectively, as shown in (9):

$$H_1 = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \quad H_2 = \begin{bmatrix} 1 & s_1 & 0 \\ s_2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$
$$H_3 = \begin{bmatrix} e_1 & 0 & 0 \\ 0 & e_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

The composite matrix H_{comp} of the one-step transformation can then be expressed as

$$H_{comp} = H_1 H_2 H_3 = \begin{bmatrix} e_1(a_{11}+S_2 a_{12}) & e_2(S_1 a_{11}+a_{12}) & a_{13} \\ e_1(a_{21}+S_2 a_{22}) & e_2(S_1 a_{21}+a_{22}) & a_{23} \\ e_1(a_{31}+S_2 a_{32}) & e_2(S_1 a_{31}+a_{32}) & a_{33} \end{bmatrix} \tag{10}$$

As in Fig. 8 (b), the three steps of the transformation of Fig. 8 (a) are combined into a one-step transformation.

Table 2 The ILUT of the inverse mapping of perspective transformation.

Y_j-Table	X_0-Table	$(\Delta x, \Delta y)$-Table			
y_0	$x_{(0,0)}$	$\Delta x_{(1,0)}, \Delta y_{(1,0)}$,	$\Delta x_{(2,0)}, \Delta y_{(2,0)}$	…	$\Delta x_{(m-1,0)}, \Delta y_{(m-1,0)}$
y_1	$x_{(0,1)}$	$\Delta x_{(1,1)}, \Delta y_{(1,1)}$,	$\Delta x_{(2,1)}, \Delta y_{(2,1)}$	…	$\Delta x_{(m-1,1)}, \Delta y_{(m-1,1)}$
…	…	…	…	…	…
y_{n-1}	$x_{(0,n-1)}$	$\Delta x_{(1,n-1)}, \Delta y_{(1,n-1)}$,	$\Delta x_{(2,n-1)}, \Delta y_{(2,n-1)}$	…	$\Delta x_{(m-1,n-1)}, \Delta y_{(m-1,n-1)}$

The combination can be performed by software and only needs to be calculated once. This simple algorithm reduces the memory usage to 1/3 without increasing the number of computations.

4.2.2 Implementation of the Δ-Algorithm

For the integer part of the coordinates, as seen in Table 2, an ILUT of perspective transformation is constructed. In fact, there are 3 sub-LUTs in Table 2. For convenience, the 3 LUTs used in the proposed system are combined into one table. The first LUT is the Y_j-table, which includes the vertical coordinate of the first pixel in each row. The second LUT is the X_0-table, which includes the horizontal coordinate of the first pixel of each row. The third LUT is the $(\Delta x, \Delta y)$-table, which includes the differences between the horizontal coordinates of the current pixel and that of previous pixel, except for the first pixel of each row.

As mentioned in Sect. 3, because the coordinates between neighboring pixels are very similar, a lot of memory can be saved by storing the difference value $(\Delta x, \Delta y)$ instead of the coordinates themselves.

In order to simplify the operation, we use 2-bit fixed-points instead of floating-points to represent the fractional parts of the coordinates. Used in conjunction with the 25-point algorithm, this system achieves fast performance without obvious image quality degradation. The encoding and decoding methods are presented in the next section.

4.2.3 Implementation of the Symmetric Principle

According to the symmetric principle mentioned in Sect. 3, for the symmetric transformations, we need to store only 1/2 and 1/4 of the LUT, respectively.

As in Fig. 8 (a), for the perspective transformation, if (x_0, y_0) is known, then:

$$x_1 = 2R - x_0, \quad y_1 = y_0 \tag{11}$$

For panorama unrolling, as in Fig. 8 (b), if (x_0, y_0) is known, then:

$$\begin{aligned} x_1 &= x_0 - R, \quad y_1 = y_0 \\ x_2 &= x_0 - R, \quad y_2 = 2R - y_0 \\ x_3 &= x_0, \quad y_3 = 2R - y_0 \end{aligned} \tag{12}$$

4.2.4 Encoding and Decoding of the ILUT Entry

The integer parts of the Y_j-table, Y_0-table and $(\Delta x, \Delta y)$-table are coded using natural binary code. The fraction parts of

the Y_i-table, X_0-table and $(\Delta x, \Delta y)$-table are used as 2-bit fixed-points and are encoded as pseudo code, as in Fig. 9, in conjunction the 25-point interpolation algorithm. Therefore, the bit numbers of both x and y are $\max(\lceil \log_2 m \rceil + 2, \lceil \log_2 n \rceil + 2)$ bits, in which $\max(\lceil \log_2 m \rceil, \lceil \log_2 n \rceil)$ bits encode the integer part, and two bits encode the fraction part.

For example, if we suppose that the image size is $640 * 480$, the output of the point is $(0, 0)$, and its inverse mapping coordinates is $(100.29, 50.585)$. Then, according to the rules in Fig. 9, the LUT values of Y_0 and $X_{(0,0)}$ can be encoded as in Fig. 10. In this study, the coordinate's convention in this paper is same with Matlab. That is, the origin of the coordinate system is located at the top-left corner of an image, and the horizontal coordinate x and vertical coordinate y start with zero.

if (0≤offset<1/4)
  Assign interpolation weight to "0", and encode to"00";
if (1/4≤offset<1/2)
  Assign interpolation weight to "1"; and encode to "01";
if (1/2≤offset<3/4)
  Assign interpolation weight to "2"; and encode to "10";
if (3/4≤offset<1)
  Assign interpolation weight to "3"; and encode to "11".
where, offset denotes the fraction part of X and Y.

Fig. 9 Encoding rules of the fractional part of inverse mapping coordinates.

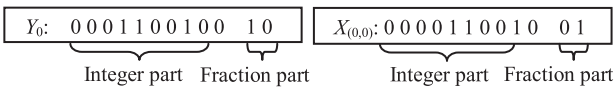


Fig. 10 An example of LUT encoding.

if (codeword = 00)
  decode to "0", and >>2 bits;
if (codeword = 01)
  decode to "1", and >>2 bits;
if (codeword = 10)
  decode to "2", and >>2 bits;
if (codeword = 11)
  decode to "3", and >>2 bits.
Note: The 2-bits right-shifting is performed after the weight multiplying by value of pixel, not right shift directly from the weight.

Fig. 11 Encoding rules of the fractional part of the inverse mapping coordinates.

As such, other x and y values of the LUT can be encoded by the same rule with $x_{(0,0)}$ and y_0. The difference is that the coordinates first need to encode the original value; however, the other pixels only need to encode the Δx and Δy.

The reading of the LUT occurs along with the output timing, that is, the raster-scanning sequence, one pixel one clock, and pixel by pixel. After calculating the values of the LUTs, the decoding procedure occurs as shown in Fig. 11, the inverse procedure of encoding.

4.3 Timing and Memory Control

The proposed system is a pipelining structure in which the output is determined after no more than one frame delay. In detail, the writing of the input buffer is synchronized with the input data timing, while the reading of the LUT ROM and the input buffer is synchronized with the output data timing. For interpolation, one line of latency occurs because a circular buffer is used. The memory usage and memory types are described as following.

There are 3 LUTs used in the proposed system, all of which use single port ROM. The total sizes of the ROM used by the Y_j-table, X_0-table and $(\Delta x, \Delta y)$-table are described in (13). If (6) holds, the fourth item of (13) can be eliminated.

$$n * (\lceil \log_2 n \rceil + 2) + n * (\lceil \log_2 m \rceil + 2) + n * m * (\lceil \log_2 \max(\Delta x) \rceil + 2) + n * m * (\lceil \log_2 \max(\Delta y) \rceil + 2) \quad (13)$$

Because the input data format is YC_bC_r 4:2:2, 3 dual port RAM (DPRAM) chips are used as an input buffer in the proposed system, one of which is for Y, while the other two are for C_b and C_r.

To provide 4 pixel values at same clock for interpolation, 3 line buffers and 12 registers (8 bits) are used for Y, Cb and Cr to construct 3 circular buffers.

4.4 Interpolation

For discrete images, pixels are taken as finite elements defined to lie on a discrete integer lattice. In general, the target coordinates of inverse mapping in an image are not defined, rather, they need to be interpolated. Classical bilinear

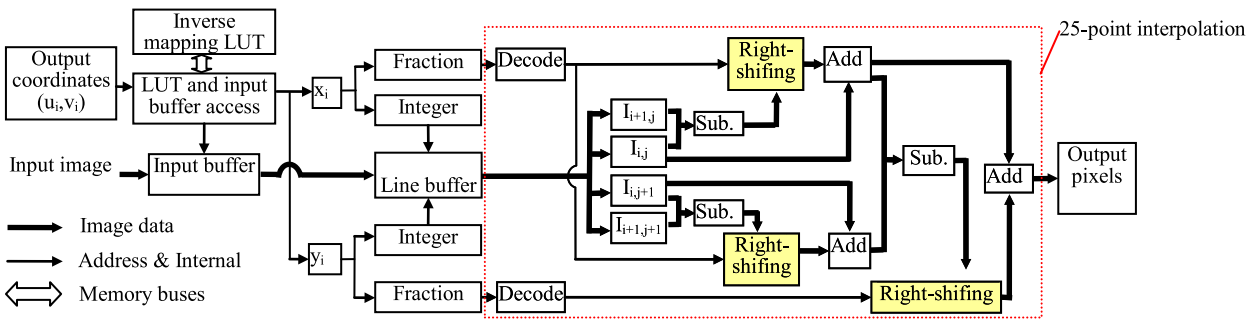


Fig. 12 Proposed hardware structure of LUT access and interpolation.

interpolation can anti-alias well, but it also needs to be accelerated for real-time implementation. Therefore, we adopt the 25-point interpolation presented in Sect. 3.

As shown in Fig. 12, the coordinates (x_i, y_i) of the inverse mapping are divided into two parts: integer and offset. The integer part is used to calculate the address of the input buffer and determine the location of a $2 * 2$ window for interpolation, whereas the offset is used to determine the weight of the interpolation.

Consider the previous example of Fig. 5 in Sect. 3. Because the point is closest to $(i + 1/4, j + 1/2)$, according to encoding and decoding rules, the offset of X and Y are assigned to 1/4 and 1/2 respectively. This is near to their original values of 0.29 and 0.585. Although the nearest algorithm reduces the precision by a small amount, since the 25-point algorithm is only divided by 2 and 4, it is very easy for hardware implementation using a right-shifting operation.

5. Experiments

Two different examples, a top view system and panorama unrolling of an image captured by a central catadioptric camera, are implemented by Verilog HDL and verified by a FPGA-based platform, as shown in Fig. 13.

The hardware platform consists of a CMOS sensor, an FPGA board, and an image capture board. We control the CMOS sensor through the image capture board with an I^2C bus, by adjusting the sensor frequency, input image format, and size.

The experiment devices and its main parameters are illustrated in Table 3. All of the optimization methods introduced in Sect. 3, the one-step algorithm, Δ-algorithm, symmetric principle, and 25-point interpolation, are used in the following two examples.

5.1 Example 1: Top View System

A top view system is usually used for rear cameras mounted on a vehicle. This system virtually adjusts the vertical perspective as if the camera was placed immediately behind the vehicle pointed directly down [17], [18], as illustrated in Fig. 14 (a). These systems are usually used in parking assistance systems to eliminate blind-zones and to assist drivers in driving and parking. It is also used by robots or surveillance to remove perspective distortion for subsequent vision-based applications.

The top view can be realized by an inverse perspective mapping, the function of which is described by (4). Figure 14 shows the experiment performed in a parking lot. Figure 14 (b) is the input image with a serious perspective distortion effect. In Fig. 14 (b), the two lines of the yellow graphic overlay in the vertical direction are not parallel. Fig. 14 (c) is the top view image after clipping and interpolation using the 25-point interpolation. In Fig. 14 (c), the top view system restores the parallel nature of the yellow lines in the graphic overlay. This corrected image can provide a feeling of safety to drivers and help them determine whether or their automobiles are properly positioned.

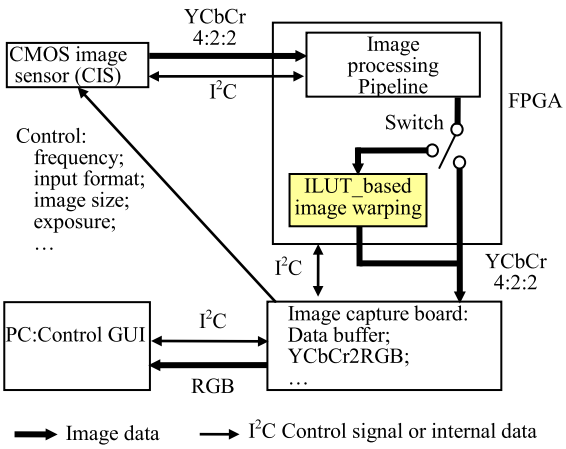


Fig. 13 System block diagram of the test platform.

Table 3 Experimental devices and their parameter settings.

	Top view system	Panorama rolling
Sensor type	CMOS image sensor	Central catadioptric camera
Lens angle	Horizontal: 120°; vertical: 70°	360°
Setting position	Rear of vehicle: h=95cm, Θ=45°	Office room
Input image	640*480	3200*768
Output image	640*480	3200*768
FPGA device	Virtex-5: XC5VLX100	Virtex-5: XC5VLX100
Processing speed	60 frames/sec	30 frames/sec

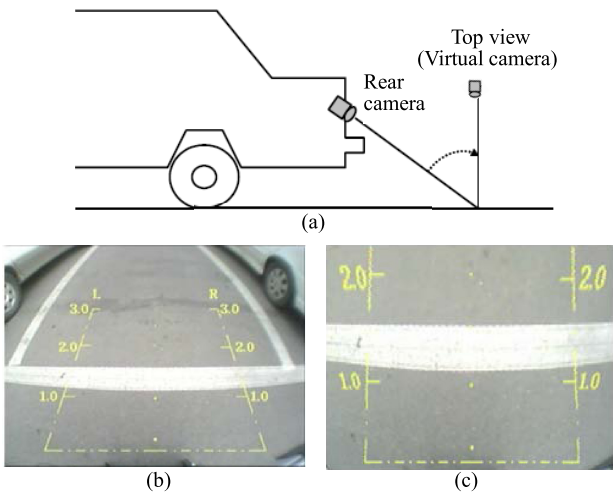


Fig. 14 Experimental results in a parking lot: (a) Top view system; (b) Input perspective image; (c) Top view image.

The system requirement and FPGA device utilization are shown in Table 4, comparing the scheme in Ref. [6] and the basic LUT scheme used in Ref. [12] and Ref. [13].

5.2 Example 2: Panorama Unrolling of an Image Captured by a Central Catadioptric Camera

The central catadioptric camera (as shown in Fig. 15 (a))

Table 4 Performance and cost comparison of conventional works and the basic LUT-based scheme.

	Scheme A [11]	Basic LUT [12][13]	Proposed scheme
Warping method	Online calculation	LUT-based	ILUT-based
Image size	640*480	640*480	640*480
Processor	32bits processor	N/A	N/A
Coordinate generation	N/A	3 LUTs	One-step scheme
Bits of every unit of LUT	N/A	X:12bit; Y: 11bits	Y_j:11bits; $X_{0,0}$:12bits; Δx: 2bits
LUT size	N/A	> 6904 Kbits	604 Kbits
Memory type	N/A	Off-chip	On-chip
LUT control module (slice)	N/A	N/A	125
Interpolation algorithm	Bilinear interpolation	Bilinear interpolation	25-point interpolation

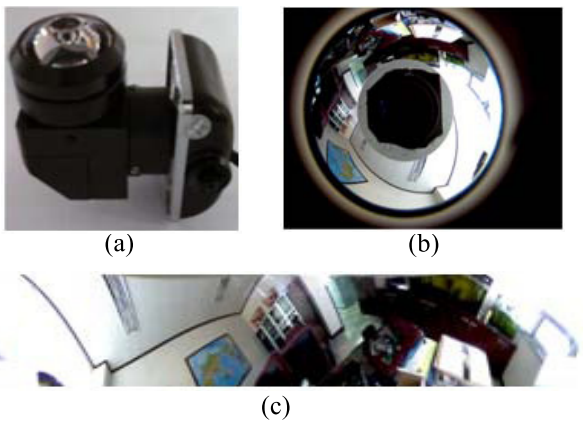


Fig. 15 A central catadioptric imaging system and its panorama unrolling: (a) Central catadioptric camera; (b) An omnidirectional image; (c) A rectangular unrolled panorama image.

utilizes a special mirror to reflect radial lines from a 360 degree field of view onto an imaging sensor, and records an original panorama image as shown in Fig. 15 (b). However, these original images have the problem of concentric circles resulting in deformations. In order to use these images in some vision-based systems, such as robot vision, they need to be unrolled [3].

The example in Fig. 16, used a concentric circle approximate unrolling, which is presented in detail in previous work [19].

The inverse mapping function of panorama unrolling is described in (14).

$$\begin{aligned} x_i &= (R * v_i/H + R) * \cos(2\pi * u_i/W) + O_x \\ y_i &= (R * v_i/H + R) * \sin(2\pi * u_i/W) + O_y \end{aligned} \tag{14}$$

where corresponding locations on the input and output images have the coordinates (x_i, y_j) and (u_i, v_j), respectively; W and H are the width and height of the output image; and R is the parameter of the input image. As Fig. 15 shows, an omnidirectional image, with a size of $1280 * 1024$, is transformed in real-time into a rectangular panorama image, with a size of $3200 * 768$.

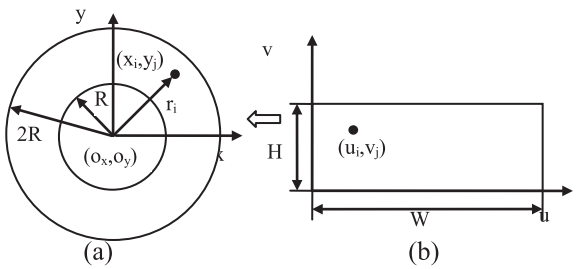


Fig. 16 Illustration of the panorama unrolling of the central catadioptric camera: (a) The coordinate system of the input image; (b) The coordinate system of the unrolled panorama image.

Table 5 Memory usage of ILUT-based panorama unrolling.

	Basic LUT	Scheme in [3]	Proposed
Input image size	1280*1024	1024*1024	1280*1024
Panorama image size	3200x768	3200x768	3200x768
Bits of every coordinate	32 bits	32 bits	1st entry:10bits; Δ-value: 4 bits
The size of LUT	150 Mbits	28 Kbits	9.3 Kbits
Memory type	Off-chip	on-chip	On-chip

All of the memory used in the simulation is on-chip block memory of the FPGA. The memory usages are shown in Table 5, comparing the scheme in Ref. [3] and the basic LUT scheme.

5.3 Experimental Results

In the first example, as shown in Table 3 and Table 4, compared to the online calculation scheme of Ref. [6], our proposed method does not need processor support and is more suitable for mobile applications. Through the use of off-line calculations, the real-time performance of 60 frames/sec can be achieved at the system frequency of 24 MHz. Thanks to the principle of symmetry, the basic LUT is first reduced to 1/2 because the inverse mapping of this example is symmetric about the vertical axis. The LUT is then further compressed through the use of the other three algorithms, resulting in a total ILUT size of only 604 Kbits when the input image size is $640 * 480$. Compared to the basic LUT, the memory usage of the ILUT is reduced to 8.7%. In addition, we can see in Table 4 that the cost of the ILUT control module is not high.

For the second example, the inverse mapping is symmetric about the origin of the coordinates. Therefore, the basic LUT is first reduced to 1/4 according to the principle of symmetry. The number of bits needed to store each entry is then reduced through the use of the Δ-algorithm. We used ten bits and four bits to encode the first values and the differences respectively. Thanks to the 25-point fixed-point interpolation, the size of the LUT is reduced further. As shown in Table 5, the ROM used by the ILUT is reduced to about 4.2%. Compared to the basic LUT-based scheme, the size of the ILUT is very small.

5.4 Error Analysis

There are only two steps that are possible to generate errors: Δ-algorithm and 25-point interpolation; other steps are lossless compression for LUT.

5.4.1 The Error of Δ-Algorithm

Why Δ-algorithm generates error is because that we used approximate operation to vertical coordinates. When (6) holds the vertical coordinates of a row are treated as a fixed value.

However, the error of the first example generated by the approximation of Δy is negligible. Because the threshold T of (6) in our experiment is 1, resulting error of coordinate value are smaller than 1/640 and the final error of output pixel value are smaller than 255/640, which is smaller than the smallest unit of pixel value 1.

In the second example, there are no errors because the vertical coordinates of each row are really same.

To other applications, error can be controlled by adjusting the T. In general, T should be smaller than $255/m$, where m denotes the width of a row. If the application requires high precision rather than low cost, T can be assigned to 1 or even smaller.

5.4.2 The Error of 25-Point Interpolation

The n-point interpolation actually is a special case of bilinear interpolation. This case uses i bits, but not floating point value, to denote bilinear interpolation coefficient. We converted the decimal fraction of coefficient into i bits in binary, where $i = 1, 2, 3 \ldots$, $n = (2^i + 1)^2$. When n approaches to ∞, n-point interpolation is very similar with the bilinear interpolation.

The number of bits and error analysis of the n-point interpolation are list in Table 6. The possible maximum coordinate error because of replacing i bits with floating point is $1/(2^{i+1})$, and the maximum pixel error in horizontal direction using these i bits coordinate as linear interpolation coefficient is $\Delta I/(2^{i+1})$, where $\Delta I = I_{i+1} - I_i$, I_i and I_{i+1} are current and next pixel value, respectively. In vertical direction, it is the same.

In this paper, $i = 2$ and thus a 25-point interpolation algorithm is used. The possible maximum errors of output pixel value are $\Delta I_x/8$ and $\Delta I_y/8$ in horizontal and vertical direction, respectively.

In general, the adjacent pixels usually have similar pixel values, except for edge area. Therefore, ΔI_x and ΔI_y usually are small in flat area, and thus the possible maximum error of 25-point algorithm are also small. In edge area, large error will occur, however, the errors cannot be easily perceived because the effect of the error is same as shifting the edges one pixel in a direction.

In conclusion, comparing to the known nearest and the bilinear interpolation algorithm, the 25-point interpolation is a tradeoff between speed and precision.

Table 6 Number of bits and error analysis of interpolation algorithms.

	Nearest	9-point [7]	25-point	Bilinear (n-point, $n \rightarrow \infty$)
Bits count	0	1	2	i
Coordinate error	$0 \sim 1/2$	$0 \sim 1/4$	$0 \sim 1/8$	$0 \sim 1/(2^{i+1})$
Output pixel error	$0 \sim \Delta I$	$0 \sim \Delta I/4$	$0 \sim \Delta I/8$	$0 \sim \Delta I/(2^{i+1})$

6. Conclusions

In this paper, extending our previous work [20], we proposed an ILUT-based image warping scheme for video camera applications. Utilizing the characteristics of image warping for video applications, the proposed ILUT drastically compresses the basic LUT and allows it to be stored in on-chip memory. Two different examples show that the proposed algorithms efficiently compress the LUT and save memory size, hence resolving the problems of complexity and bandwidth requirements.

The proposed system can be used in video cameras for automobiles, robot vision, surveillance, and general digital cameras. It can also be used as an independent system and as a preprocessing module of an image SOC sensor in cooperation with other modules.

Acknowledgments

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2012-C1090-1200-0010). The IDEC provide research facilities for this study. This work was also supported by the Brain Korea 21 Project in 2012 and the IT R&D program of MKE/KEIT (10035570, Development of self-powered smart sensor node platform for smart and green building).

References

[1] G. Wolberg, Digital Image Warping, pp.41–61, IEEE Computer Society Press, New York, 1990.
[2] J. Varona, J. Gonzàlez, I. Rius, and J.J. Villanueva, "Importance of detection for video surveillance applications," Opt. Eng., vol.3, no.8, pp.087201–1-9, 2008.
[3] L. Chen, M. Zhang, B. Wang, Z. Xiong, and G. Cheng, "Real-time FPGA-based panoramic unrolling of high-resolution catadioptric omnidirectional images," ICMTMA'09., vol.1, pp.502–505, 2009.
[4] R.L. Nagel, K.L. Perry, R.B. Stone, and D.A. McAdams, "Function-based design process for an intelligent ground vehicle vision system," J. Electron. Imaging, vol.19, no.4, pp.043024-1–13, 2010.
[5] He and Y. Li, "Camera calibration with lens distortion and from vanishing points," Opt. Eng., vol.48, no.1, pp.013603-1–11, 2009.
[6] N. Gonçalves and H. Araújo, "Low-cost method for the estimation of the shape of quadric mirrors and calibration of catadioptric cameras," Opt. Eng., 46, vol.16, no.7, pp.073001-1–12, 2007.
[7] Mohan, A.K. Senapathi, R. Shankar, N. Bhat, and V. Kumar, "Hardware acceleration of real time fish eye correction on FPGA," NCIS-2009, 2009.
[8] Barenbrug, F.J. Peters, C.W.A.M. van Overveld, "Algorithms for division free perspective correct rendering," ACM SIGGRAPH/ EUROGRAPHICS workshop Graph. Hardware, pp.7–13, 2000.

[9] J. Fang, K. Moseler, and S. Levi, "A method to reduce number of division operations for perspective texture warping," IEEE Int. Symp. Circuits and Systems, vol.3, pp.618–621, 2000.
[10] B. Chen, F. Dachille, and A. Kaufman, "Forward image mapping," Visualization '99, pp.89–514, 1999.
[11] Huggett, C. Silsby, S. Cami, and J. Beck, "A dual-conversion-gain video sensor with dewarping and overlay on a single chip," IEEE Int. Solid-State Circuits Conf., pp.52–53, 2009.
[12] P. Mattson, D. Kim, and Y. Kim, "Generalized image warping using enhanced lookup tables," Int. J. Imaging Syst. Technol., vol.9, pp.475–483, 1998.
[13] S. Oh and G. Kim, "FPGA-based fast image warping with data-parallelization schemes," IEEE Trans. Consum. Electron., vol.54, no.4, pp.2053–2059, 2008.
[14] G. Woldberg and T. Boult, "Separable image warping with spatial lookup tables," Comput. Graph., vol.23, no.3, pp.369–378, 1989.
[15] K.T. Gribbon and D.G. Bailey, "A novel approach to real-time bilinear interpolation," Int. Conf. Electronic Design, Test and Applications, pp.126–131, 2004.
[16] E. Staudinger, M. Humenberger, and W. Kubinger, "FPGA-based rectification and lens undistortion for a real-time embedded stereo vision sensor," Proc. FH Science Day, pp.18–25, 2008.
[17] Y. Ishii, K. Asari, H. Hongo, and H. Kano, "A practical calibration method for top view image generation," Int. Conf. on Consumer Electronics, pp.1–2, Jan. 2008.
[18] L. Luo, I. Koh, K. Min, J. Wang, and J. Chong, "Low-cost implementation of bird's-eye view system for camera-on-vehicle," Int. Conf. Consumer Electronics, pp.311–312, Jan. 2010.
[19] R. Ahn, L. Luo, K. Min, E. In, and J. Chong, "Low-cost panorama unrolling of catadioptric omnidirectional images," Sixth IEEE Int. Sym. on Elec. Design, Test and Appl., pp.185–188, 2011.
[20] L. Luo, C. Wang, J. Chen, S. An, Y. Jeung, and J. Chong, "Improved LUT-based image warping for video cameras," CSE 2011, pp.453–460, Aug. 2011.

Se-yong Ro received a B.S degree in Electronic Engineering from Hanyang University, Seoul, Korea, in 1983, and the M.S degree in Electronics and Communications Engineering from Waseda University, Tokyo, Japan in 1989. He is currently a senior vice president of LG Uplus, Seoul, Korea and a candidate for the PhD degree in the Department of Electronic and Computer Science, Hanyang University, Seoul, Korea. His research interests include image warping, and mobile communication.

Lin-bo Luo received a B.S degree in Electronics and Information Engineering from China University of Geosciences, Wuhan, China, in 2002, and the M.S degree in Engineering of Traffic Information and Control from Wuhan University of Technology, Wuhan, China in 2005. He is currently a candidate for the PhD degree in the Department of Electronic and Computer Science, Hanyang University, Seoul, Korea.

Jong-wha Chong (M' 85) was born in Nonsan, Korea, on March 10, 1950. He received B.S. and M.S. degrees in Electronics Engineering from Hanyang University, Seoul, Korea, in 1975, and 1979 respectively. He received his Ph.D. degree in Electronics & Communication Engineering from Waseda University, Japan, in 1981. His current research interests are the design of ASIC emulation system, CAD for VLSI, H.264 encoder/decoder design, and communication circuit design, especially UWB modem design.