Adaptive Memory Architecture for Real-Time Image Warping

Andy Motten, Luc Claesen
Expertise Centre for Digital Media
Hasselt University – tUL – IBBT
Wetenschapspark 2, 3590 Diepenbeek, Belgium
{firstname.lastname}@uhasselt.be

Yun Pan
Institute of VLSI Design
Zhejiang University
Hangzhou, China
panyun@vlsi.zju.edu.cn

Abstract—**This paper presents a real time image warping module implemented in hardware. A look-up table (LUT) based reverse mapping is used to relate the source image to the warped image. Frame buffers or line buffers are often used to temporally store the source image. However these methods do not take the underlying pattern of the reverse mapping coordinates into account. The presented architecture uses an adaptable memory allocation which can change the depth and the position of the line buffer between lines. A real-time stereo rectification use case has been implemented to validate the operation of this module. Depending on the scenario, the memory consumption can be reduced by a factor of two and more. A real-time image warping module for video cameras has been implemented in a single FPGA, without the use of off-chip memories.**

Keywords-component; warping; real-time; rectification; memory architecture; system-on-chip; FPGA;

I. INTRODUCTION

Image warping is an important research topic in computer vision and graphics. It is a spatial transformation of an image based on a geometric relationship [1]. The image warping module takes a stream of pixels from a camera, temporally stores them and outputs them in a different order (Fig. 1). The specific order depends on the geometric relation which can either be calculated on-line or pre-calculated and stored in a look-up table (LUT). In a video pipeline, the data stream is a synchronized source of pixels coming from an image where the first element of the stream is the top left pixel in the image and the following elements are corresponding pixels on the same line. For each pixel in the output image a color is selected from the input image using its inverse-warped location.

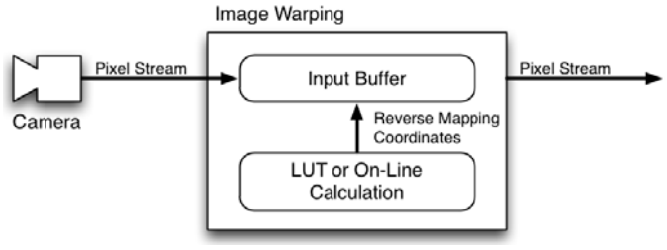

Fig. 1. Image warping

Real-time image warping is necessary in several applications. In medical endoscopy it is used to remove the large radial distortion caused by the small size of the lens [2]. For the generation of panoramic videos, the images of the cameras need to be warped with respect to each other in order to stitch them together [3]. In a stereo camera setup, two different kinds of distortions are present; the first one is the lens distortion, the second one is a misalignment of the two cameras. Since the search space for stereo matching is located on the epipolar line, both distortions should be resolved before the matching can be performed [4]. Other applications include image rotation, scaling, translation or a combination of them [1].

For all these applications, not only real-time is important, but also low latency and cost. We believe that this can only be achieved by implementing warping into hardware and by removing the usage of external memories.

Luo implemented a look-up table (LUT) based image warping module in hardware [5]. By using a compressed LUT it doesn't need off-chip memory to store it. However, it still needs off-chip memory to temporally store the input stream of the camera. Oh implemented a FPGA based fast image warping module which focuses on latency reduction [6]. A single LUT multiple access method is proposed which stores the LUT in on-chip memories. Off-chip memories are used to store the output image. Rodrigues implemented a real-time rectification module for stereo images [7]. It used a Microblaze processor to calculate the reverse mapping coordinates and made use of off-chip memories to store the input stream.

The goal of the presented architecture is to reduce memory usage in order to implement the complete warping module on-chip without the use of off-chip memories. The focus is oriented towards the reduction of the pixel input buffer. Several data structures will be presented which reduce the memory usage by taking the underlying pattern of the reverse mapping coordinates into account. The reverse mapping coordinates are stored in a LUT and bilinear interpolation [8] is used to get sub-pixel accurate results.

The remainder of the paper is organized as follows: section two describes image warping in greater detail. Section three introduces the different memory architectures for the pixel input stream buffer. Section four presents the hardware

This research has been sponsored in part by the BOF (Bijzonder Onderzoeks Fonds uHasselt), Flanders FWO (Fonds voor Wetenschappelijk Onderzoek) and Chinese MOST (Ministry of Science and Technology) project number G.A.063.10

architecture. Section five discusses the implementation results. Section six concludes the presented architecture.

II. Warping Overview

Image warping consists of three main parts. First, the reverse mapping coordinates need to be provided. They can be pre-calculated and stored in a LUT or calculated on-line. The LUT based method has the advantage that no expensive calculations are needed but with the cost of additional memory usage. For this architecture, a memory efficient LUT based implementation is chosen. Second, the input pixel stream needs to be stored in order to select the output pixels from. Third, the output pixels are resampled in order to get sub-pixel accuracy.

A. Reverse Mapping Coordinates

For each pixel of the warped image a pixel of the source image is selected. The index difference between the warped and the source image pixel are called the reverse mapping coordinates.

Storing the mapping coordinates for each pixel uses a large amount of memory. When the mapping coordinates do not change drastically from pixel to pixel, it suffices to only store the mapping coordinates of certain pixels. These pixels are chosen to be located on a regular grid. The desired grid size depends on the amount of distortion in the image.

Bilinear interpolation is used to reconstruct the mapping coordinates for the complete image (Fig. 2). Formula (1) shows how the mapping coordinates for pixel 'p' (map_p) are calculated from the mapping coordinates of pixel 'a' (map_a), 'b' (map_b), 'c' (map_c) and 'd' (map_d), which are located on the rectangular grid.

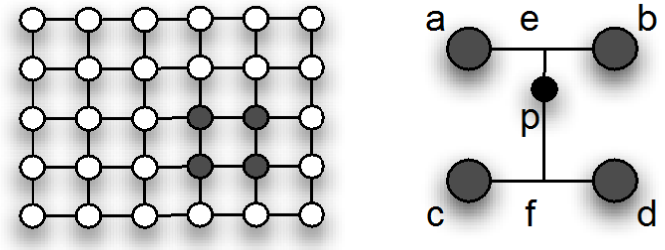

Fig. 2. Grid based mapping (left: complete grid, right: bilinear interpolation between grid points).

$$\begin{cases} map_e = (|ae| \cdot map_b + |eb| \cdot map_a)/|ab| \\ map_f = (|cf| \cdot map_d + |fd| \cdot map_c)/|cd| \\ map_p = (|ep| \cdot map_f + |pf| \cdot map_e)/|ef| \end{cases} \quad (1)$$

B. Storage of the Source Image

The warped image is constructed by selecting the pixels from the source image whose coordinates are provided by the reverse mapping LUT. In order to allow the selection of pixels from previous lines, it is necessary to store a previous number of pixel lines in a memory (Fig.3). Most commonly a frame buffer or line buffers are used to store the pixels. However this is not efficient when the underlying structure of the reverse mapping coordinates is known. In the next section, this will be discussed in more detail.

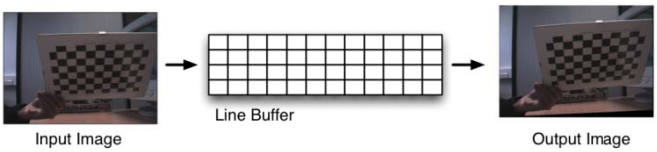

Fig. 3. Warping of input to output pixel stream using line buffer storage.

C. Sub-Pixel Resampling

When the mapping coordinates are integer, only one source pixel is needed for the warped image. However a better result can be obtained when making use of mapping coordinates which contain fractional values (Fig.4) When using mapping coordinates with sub-pixel accuracy, a window of four pixels is used from the source image. Bilinear interpolation is hence used to calculate the resulting warped pixel [8].

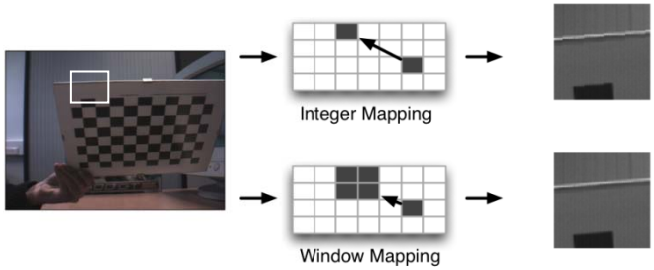

Fig. 4. Sub-pixel resampling (top: without resampling, bottom: with resampling).

D. Examples

Fig. 5 shows some examples of common warping examples. The repercussions on the memory consumption will be discussed in the next section.

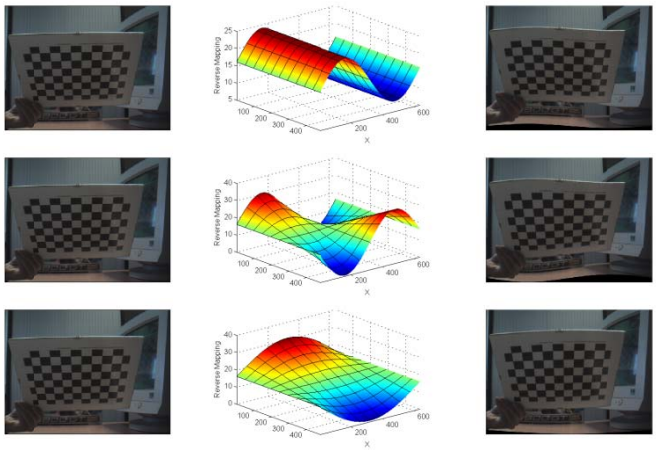

Fig. 5. Warping examples, from top to bottom: wave, expand and compress and barrel distortion (left: source image, middle: vertical warping LUT, Right: warped image).

III. Memory Architecture

In order to perform reverse mapping, the source pixels need to be stored in a temporal buffer. The size of this buffer is primarily determined by the vertical warping coordinates. The larger its range across the image, the more line buffers needs to be buffered. The horizontal warping coordinates will only determine some additional pixel buffers. In the remainder of this paper, only vertical warping coordinates are taken into account.

A. Circular Buffer

When the vertical offset is uniformly distributed on an image line or when the pattern of the reverse mapping coordinates are not known, a line buffer implementation is the most suitable. Fig. 6 shows an example where no pattern can be found in the reverse mapping LUT.

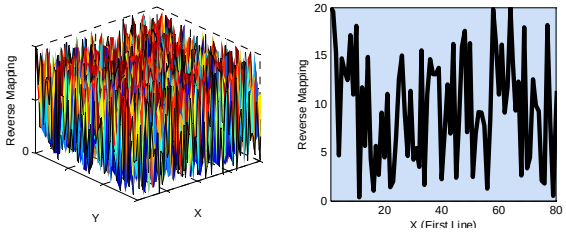

Fig. 6. LUT without known pattern (left: vertical reverse mapping LUT, right: memory access, gray indicates the memory access envelope).

The size of the buffer equals the width of the image multiplied with the number of lines. This in turn equals the highest vertical offset in the reverse mapping coordinates. A circular buffer is used as data structure (see Fig. 7).

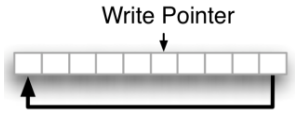

Fig. 7. Circular buffer

B. Split Circular Buffer

When the vertical offset is not uniformly distributed on an image line, parts of the stored pixels are never actually read out. On Fig. 8, the mapping coordinates have a sinusoidal pattern. The resulting memory access is depicted on Fig. 8. The left side of the image needs a large buffer, while the right side of the image only uses a much smaller buffer.

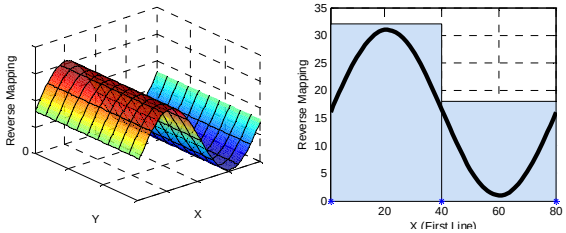

Fig. 8. LUT with wave pattern (left: vertical reverse mapping LUT, right: memory access, gray indicates the memory access envelope).

Instead of using a line buffer with a fixed vertical offset across the image line, it is more memory efficient to split up the line buffer into two (or more) slices; One large line buffer for the left slice of the image and one smaller line buffer for the right slice. Note that these line buffers can be located in the same physical memory. For each memory slice, a write pointer is stored (Fig.9). For every pixel write, it is determined in which memory slice the pixel needs to be written.

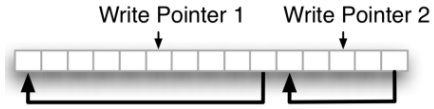

Fig. 9. Split circular buffer

The read pointer is calculated from the current write pointer; the additional calculation step includes the determination of which circular buffer to read from.

C. Adaptive Split Circular Buffer

When the vertical offset is not uniformly distributed on an image line and the horizontal offset changes during consecutive image lines an adaptive split circular buffer is needed. This situation occurs when in the first slice, the image is expanded vertically while in the second slice, the image is compressed vertically (Fig. 10). The memory access will change gradually with consecutive lines. The split line buffer needs to be adapted in between image lines to accommodate this change.

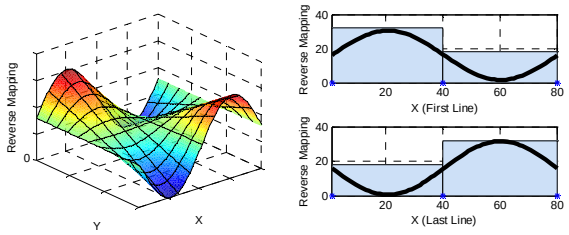

Fig. 10. LUT with changing wave pattern (left: vertical reverse mapping LUT, right: memory access, gray indicates the memory access envelope).

In this scenario, the data structure consists of several circular buffers which are linked together. These linkages will change during consecutive image lines. In the top of the image, the gray memory block (Fig. 11) will be assigned to the left slice of the image, while on the bottom of the image it will be assigned to the right slice of the image.

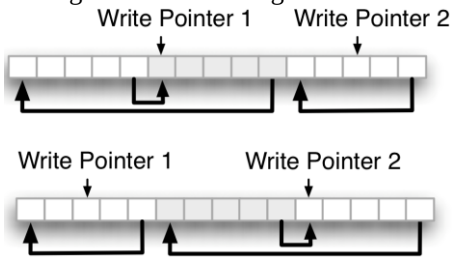

Fig. 11. Adaptive split circular buffer

The information about memory allocation for different slices of the image can be stored in small additional memories.

D. Circular Buffer with Line Read Postponing

In this scenario, the vertical offset decreases during consecutive image lines. First the complete buffer needs to be filled with pixel data before reading can be started. This is a result from the fact that all lines are normally processed one after another. Second, on each line, only parts of the buffer are used. It is more memory efficient to take a smaller memory and to postpone line reads when needed.

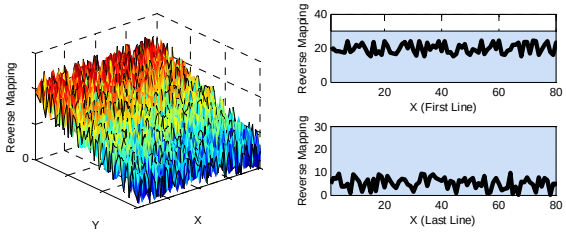

Fig. 12. LUT with decreasing random pattern (left: vertical reverse mapping LUT, right: memory access, gray indicates the memory access envelope).

Every time, when reading is postponed for a complete line, the next line that is being read will have an additional offset of plus one. The assumption is that the input pixel stream has blanking lines and that the next image processing steps are prepared for handling them. The first assumption holds when making use of a simple CMOS camera; after the last line, it will send a couple of pixel lines that are black, the number of these blanking lines can be programmed. The second assumption depends on the architecture of the next processing steps. Fig. 13 shows an image where reading has been stopped for a couple of lines on two separate places. This leads to a reordering of the blanking lines.

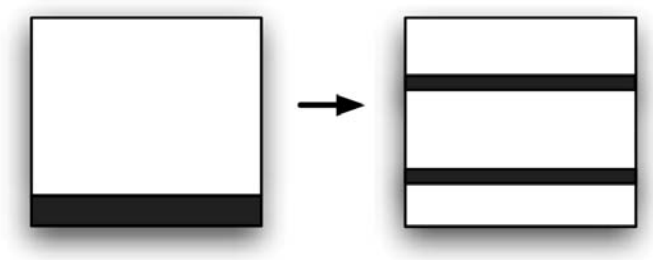

Fig. 13. Reordering of blanking lines.

The reverse mapping LUT has to be changed every time a read line is skipped. Since the line skip is predetermined, the LUT can be changed off-line in order to reduce the size of it. For each line skip, all mapping coordinates from that line on will be decremented by one.

Fig. 14 Shows the result of the modification of the reverse mapping coordinates of Fig. 12 when line skipping is used. In this case a reduction in memory usage of more than 50% is obtained.

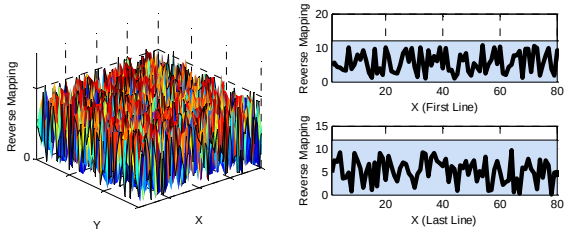

Fig. 14. LUT with decreasing random pattern, modified for line read postponing.

E. Adaptive Split Circular Buffer with Line Read Postponing

This is the most complex case investigated in this paper and resembles the scenario when a camera is corrected from its barrel distortion. As seen on Fig. 15, the vertical offset decreases continually during consecutive lines. The vertical offset is not uniform within a line and it changes during consecutive lines.

When only using a split circular buffer approach, not much reduction in memory usage is obtained. The reason is the flatness of the reverse mapping curve on the first line (Fig. 15).

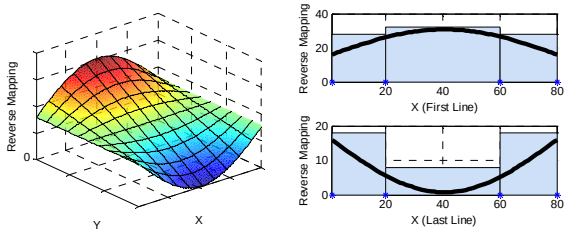

Fig. 15. LUT for barrel distortion (left: vertical reverse mapping LUT, right: memory access, gray indicates the memory access envelope).

When only using the circular buffer with line read postponing, a reduction of 50% is obtained since the value range of each line in the reverse mapping LUT is halve of the maximum value of this LUT.

When adding the adaptive split circular buffer approach with line read postponing, a larger reduction of memory usage is obtained. On the lower part of the image, read line skipping is used to reduce the need for extra line buffers. Fig. 16 shows the result of the modification of the reverse mapping coordinates of Fig. 15 when line skipping is used.

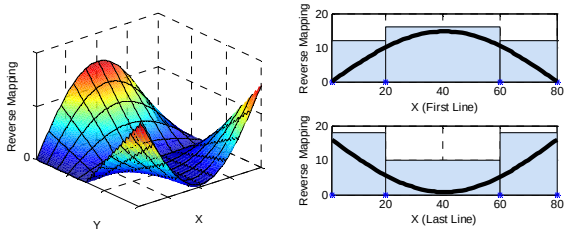

Fig. 16. LUT for barrel distortion, modified for line read postponing.

The combination of the adaptive split circular buffer with line read postponing leads to a reduction of more than 50%.

IV. Hardware Architecture

The goal of the presented architecture is to reduce memory usage in order to implement the warping module on-chip with room to spare for additional applications like stereo vision. The image warping module receives pixels coming from a camera and places them in an input buffer (see Fig. 17). The reverse mapping coordinates are calculated from the LUT using bilinear interpolation. The integer part of these coordinates is used to select the four part window from the input buffer. The fractional part is used to calculate from this window the warped pixel using bilinear interpolation.

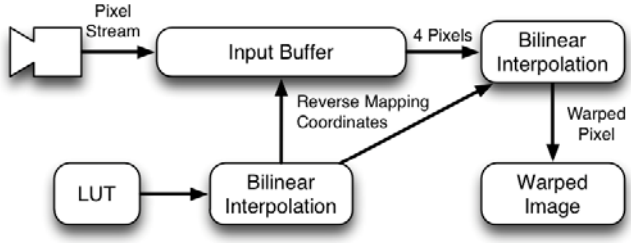

Fig. 17. Image warping architecture.

The main focus of this paper is the reduction of memory usage in the input buffer. Instead of using a single line buffer, a method is chosen which allows for adaptable line buffers for different slices of the images (see section III) and allows the postponing of a line read. The on-chip memory is a dual port RAM which has separate read and write address inputs and a four pixel window output (see Fig. 18).

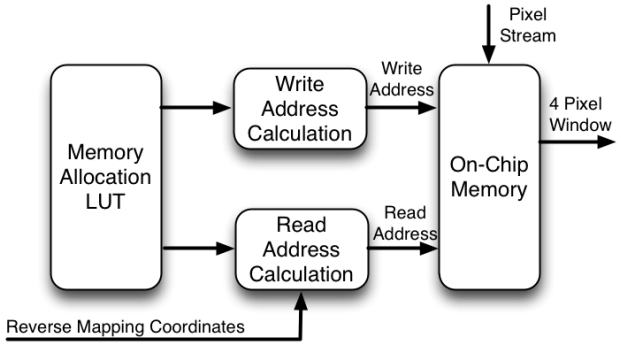

Fig. 18. Input buffer architecture.

The memory allocation LUT provides the write and read address calculation blocks with information about the structure of the memory. It is a list containing the memory blocks and their linkage for each slice of the image (see Fig. 19). It also contains information about when to switch from one memory allocation to another and when to postpone reading of a pixel line.

For each memory slice (e.g. left and right slice in Fig. 10) a write address pointer is stored. In Fig. 20 this is denoted by the array *WR_addr*, where the index *nr* indicates the current image slice. Register *a* keeps track of the current horizontal position with respect to the start of the image slice; when *a* equals the width of the image slice, it jumps to the next slice.

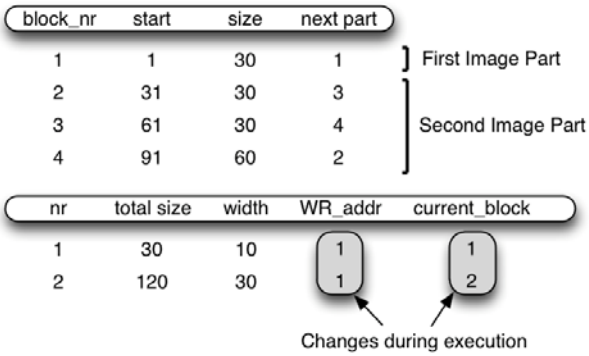

Fig. 19. Memory allocation LUT.

Each memory slice can consist of one or more memory blocks which are linked together. When *WR_addr* equals the end of the current block, the *WR_addr* jumps to the next memory block (or the start of its own block).

```
if (a > width(nr)), then a = 1; nr++; else a++;

if WR_addr(nr)  >
size(current_block(nr))+ start(current_block(nr))
then  current_block(nr) = next_block(current_block(nr));
      WR_addr(nr) = start(current_block(nr));
else  WR_addr(nr) ++;

mem(WR_addr(nr)) = Io(x,y);
```

Fig. 20. Main steps for the write address calculation

The read address is calculated from the memory allocation LUT and the calculated reverse mapping coordinates. The first step consists of determining in which image slice the warping pixel is located (Fig. 21). This depends on the value of the horizontal mapping and the current horizontal position with regards to the start of the image slice (*a*). The *RD_addr* is initially calculated from the assumption that each image slice only contains one memory block, the *relative_write_address* is the write address in this fictional memory block. Next the vertical offset is subtracted. If the result is lower than zero, it is subtracted from the total size of this fictional memory block. The last part calculates the position of the *RD_addr* pointer in the separate memory blocks using a mapping module.

```
if (horizontal_offset > a)
then  nr_read = nr-1;
      RD_addr = relative_write_address(nr_read) -
                horizontal_offset + a;
else  nr_read = nr;
      RD_addr = relative_write_address(nr_read) -
                horizontal_offset;

RD_addr = RD_addr - width(nr_ read)* vertical_offset
if RD_addr < 0, then RD_addr = RD_addr + slice_size(nr_ read);

RD_addr = mapping(RD_addr);
Iw(x,y) = mem(RD_addr);
```

Fig. 21. Main steps for the read address calculation

Incorporation of the line read postponing function is straightforward. On the start of each image line, it is checked if its needs to be postponed or not. If yes, the read address calculation module doesn't do anything for this line. Notice that the number of read lines will not be changed; every line read that is postponed will add an extra line read at the end of the frame. The reverse mapping coordinates do not need to be modified online since they already have been modified off-line.

V. Implementation

Matlab has been used to generate and optimize the several look-up tables. From the full warping coordinates, the best memory architecture is chosen taken into consideration the additional calculation steps of more complex architectures.

The architecture and methods presented in this paper have been implemented on an FPGA system, based on an Altera Cyclone IV with 114,480 logic elements and 432 memory blocks. The sources of the input stream are two cameras with a resolution of 640x480 and a pixel clock of 16 MHz resulting in a frame refresh rate of 52 Hz. The application chosen is the rectification of two stereo streams. In this case, the proposed memory buffer is an adaptive split circular buffer with line read postponing.

The architecture has been constructed to reduce memory usage. Hence there is no need for external memories. The reduction of external memory usage has the additional advantage that the latency between input frame and output frame becomes minimal.

Table 1 shows the resource consumption of the main blocks presented in this paper. The synthesis results are obtained using Quartus II 11.0 for an Altera Cyclone IV. Note that the proposed buffer method reduces the memory usage significantly (16% of available memory blocks) while just using a small number of extra logic elements (1% of available logic elements). Fig. 22 shows the results before and after the warping module. It is clear that matching along the epipolar line (white line) will be improved after rectification of both cameras.

TABLE I
Resource Usage for the Rectification of a Stereo Camera

	Module Name	#	Logic Elements		Memory Blocks	
			Single	**Total**	**Single**	**Total**
Input Buffer	Circular Buffer	2	60	120	75	150
	OR					
	Proposed Buffer	2	840	1,680	40	80
Reverse Mapping	LUT	2	0	0	20	40
	Horizontal Interpolation	2	428	856	21	42
	Vertical Interpolation	2	428	856	21	42
	Resampling	2	960	1,920	21	42
Total with Circular Buffer				**3,732**		**316**
Total with Proposed Buffer				**5,312**		**246**

VI. Conclusions

A real-time image warping module has been presented. The main focus was the storage of the input stream before the pixels are warped. The presented architecture uses an adaptable memory allocation which can change the depth and the position of the line buffer between lines. It is shown that making use of this method reduces the memory usage in several use-cases. The reverse mapping coordinates are stored in a LUT and bilinear interpolation is used to get sub-pixel accuracy.

In the case of stereo vision, the image warping module is used to perform real-time rectification. A memory reduction of approximately 50% is obtained compared to a common line buffer implementation.

The architecture has been implemented in a field programmable gate array (FPGA) without making use of external memories.

Fig. 22. Image rectification results before (top) and after (bottom) warping for a stereo camera.

References

[1] D. King, "Applications for real time image warping," in Conference Record IEEE Southcon-1996, 1996, pp.298-302.

[2] R. Melo, J.P. Barreto and G. Falcao, "A New Solution for Camera Calibration and Real-Time Image Distortion Correction in Medical Endoscopy–Initial Technical Evaluation," IEEE Transactions on Biomedical Engineering, vol. 59 (3), 2012, pp.634-644.

[3] F. Huang, R. Klette, J-C Tien and Y-W Chang, "Rotating sensor-matrix camera calibration," in Proceedings IEEE ICIP-2010, 17th International Conference on Image Processing, 2010, pp.4245-4248.

[4] A. Motten, and L. Claesen, "Low-cost real-time stereo vision hardware with binary confidence metric and disparity refinement," in Proceedings IEEE ICMT-2011, International Conference on Multimedia Technology, 2011, pp.3559-3562.

[5] L. Luo, C. Wang, J. Chen, S. An, Y. Jeung and J. Chong, "Improved LUT-Based Image Warping for Video Cameras," in Proceedings IEEE CSE-2011, 14th International Conference on Computational Science and Engineering, 2011, pp.453-460.

[6] S. Oh and G. Kim, "FPGA-based fast image warping with data-parallelization schemes, " IEEE Transactions on Consumer Electronics, vol. 54 (4), 2008, pp.2053-2059.

[7] J.G.P. Rodrigues and J.C. Ferreira, "FPGA-based rectification of stereo images," in Proceedings IEEE DASIP-2010, Conference on Design and Architectures for Signal and Image Processing, 2010, pp.199-206.

[8] K.T. Gribbon and D.G. Bailey, "A novel approach to real-time bilinear interpolation," in Proceedings IEEE DELTA 2004, Second IEEE International Workshop on Electronic Design, Test and Applications, 2004, pp.126-131.