EURASIP Journal on Image and Video Processing
a SpringerOpen Journal

RESEARCH Open Access

An FPGA-based processing pipeline for high-definition stereo video

Pierre Greisen[1,2*], Simon Heinzle[2], Markus Gross[1,2] and Andreas P Burg[3]

Abstract

This paper presents a real-time processing platform for high-definition stereo video. The system is capable to process stereo video streams at resolutions up to 1, 920 × 1, 080 at 30 frames per second (1080p30). In the hybrid FPGA-GPU-CPU system, a high-density FPGA is used not only to perform the low-level image processing tasks such as color interpolation and cross-image color correction, but also to carry out radial undistortion, image rectification, and disparity estimation. We show how the corresponding algorithms can be implemented very efficiently in programmable hardware, relieving the GPU from the burden of these tasks. Our FPGA implementation results are compared with corresponding GPU implementations and with other implementations reported in the literature.

Keywords: Video processing pipeline, Stereoscopic video, FPGA, Disparity estimation, Image warping

1 Introduction

Multi-view camera systems are becoming ubiquitous in today's camera landscape. This trend has been driven by the advent of affordable high-quality imaging sensors on one hand and by novel multi-view applications on the other hand. Prominent examples of such new applications are 360° vision panoptic cameras [1] that provide immersive video experiences and vision systems for automatic vehicle/robot control or for augmented reality [2].

A particularly interesting and commercially most relevant real-time application is stereoscopic 3D (S3D) movie and broadcast production [3]. Corresponding video capture systems often employ two spatially offset cameras; however, imperfections of such systems require real-time processing of the video streams that reach beyond the processing routinely done today in monoscopic camera systems. In particular, different cameras usually exhibit different sensor responses that can lead to unpleasant viewer experiences. Also, the physical alignment is often not sufficiently accurate and the resulting videos can lead to eyestrain [3]. To correct for non-idealities in the camera system, the raw video streams need careful color correction to achieve similar color responses across all views. In addition, an image warping process must remove distortions and correct potential mis-alignments of the hardware. Most recent camera systems furthermore rely on real-time analysis of the captured images for guiding the camera operator [4,5] and for automatic camera control [4]. A vital element of such systems is the analysis of the physical layout of a captured scene by analyzing screen space *disparities*. Disparities are the resulting displacements of a scene point across both camera views, which in turn directly relate to the actual geometry of the scene.

Unfortunately, such processing of high-definition video streams for real-time applications is a challenging task since computationally demanding algorithms need to be applied to high-resolution images. In addition, image processing tasks require a considerable amount of memory bandwidth. Current CPUs and GPUs are therefore often completely occupied when performing the full low-level processing and analysis pipeline including disparity analysis in real-time. FPGA platforms on the other hand offer great potential for streaming-based tasks. Video pipelines can be implemented in parallel for low latency and high performance. Furthermore, optimal speed-precision-area tradeoffs can be accomplished using fixed-point arithmetic, and custom-tailored caching architectures can be employed to alleviate bandwidth bottlenecks.

Related work

Although some work using FPGA pipelines has been published recently, all existing real-time hardware

* Correspondence: pierre.greisen@disneyresearch.com
[1]ETH Zurich, 8092 Zurich, Switzerland
Full list of author information is available at the end of the article

systems for video processing and disparity analysis focused on lower-resolution video streams [6-12]. No work has been successfully extended for high-definition video due to the super-linear increase in processing complexity. The corresponding increased memory bandwidth bottlenecks furthermore impose additional hardware constraints for higher video resolutions.

Contributions

In this work, a processing platform for high-definition stereo video is presented. The platform employs FPGAs to perform low-level image processing, image warping, and disparity estimation in real-time and in full HD resolution. We describe the corresponding hardware-efficient algorithms, the essential components of the associated hardware architecture, and the required caching mechanisms that enable the processing of high-definition video streams. The FPGA is integrated into a heterogeneous PC platform that also comprises a GPU and a CPU for further processing. We argue that this mapping of the stream processing onto the FPGA is advantageous since the CPU and GPU resources can in turn be spent on higher-level operations. Such operations can include control algorithms for system settings [4] or memory intensive algorithms with a more global scope such as segmentation, which often are not well suited for FPGA processing. To this end, we provide reference numbers to illustrate the GPU resources freed for other tasks by the FPGA implementation. Finally, we provide a comparison of our hardware implementation to other state-of-the-art implementations of the core algorithms and the corresponding limitations.

2 Stereo processing pipeline overview

Figure 1 shows a high-level overview of the processing pipeline. Two or more synchronized cameras stream directly to the FPGA board. The incoming video streams are first synchronized in case of slight timing differences. Then, low-level image pre-processing tasks are performed. The corresponding low-level processing unit will be discussed in more detail in Section 3.

The color-corrected video streams then undergo a geometry transformation in the image warping unit. This unit performs radial undistortion and performs almost arbitrary projective transformations of the image. The transformation uses a backward mapping with bilinear interpolation. The most critical issue in the implementation of this unit are the memory requirements and the considerable memory bandwidth. These issues are addressed by a novel block-caching mechanism that alleviates the bottleneck to the off-chip memory with low on-chip memory overhead. This unit will be discussed in more detail in Section 4.

After geometry correction in the image warping unit, the disparity estimation unit searches for matching features along horizontal lines. More specifically, a local, window-based disparity matching [13] using a ZSAD (zero mean sum of absolute differences) cost function is employed. Similar to the GPU implementation of Zach et al. [14], the video streams are downsampled multiple times into a multi-resolution pyramid structure. Disparity estimation is first performed on the lowest resolution using a reduced disparity range. Then, the coarse disparity estimates are subsequently propagated to higher levels for refinement. Using the pyramid structure, fine-grained details can be traded off for higher noise resilience and reduced computational complexity. A more detailed discussion is provided in Section 5.

The FPGA processing pipeline described above is implemented on a PCI Express FPGA board that is equipped with external memory banks. A DMA controller and local bus controller communicate with the PC subsystem for data transfer and configuration. The PCI Express transfers are fast enough to send the data to the

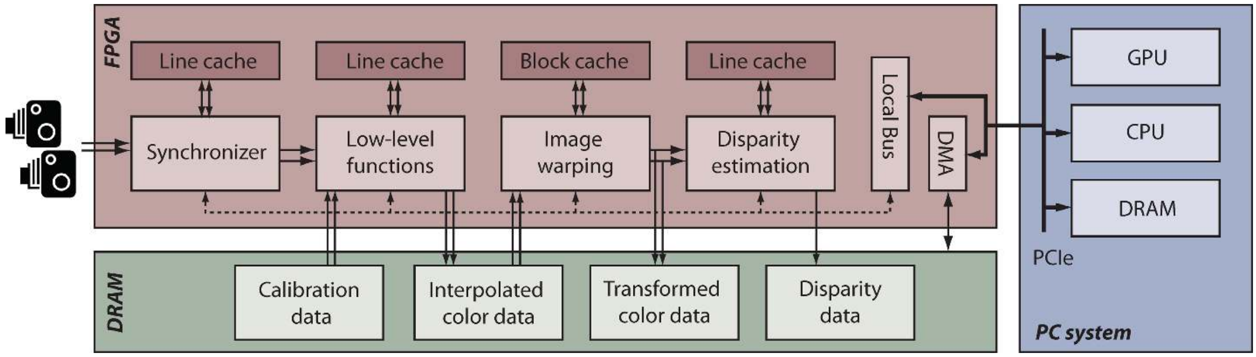

Figure 1 Data flow of the presented stereo camera pipeline. Two video streams are synchronized on the FPGA, before low-level functions (pattern noise correction, color interpolation, and color correction) are performed. Then, the streams can be undistorted and rectified arbitrarily using the geometric transformation unit. Disparity estimation is performed on the well-aligned video streams. The FPGA board features external DDR2 DRAM banks and communicates to the PC subsystem using PCI Express transfers.

GPU and the CPU for further processing, e.g., to implement further post-processing or control algorithms and to store the video sequence on a solid-state disk.

3 Low-level stereo processing

An overview of the low-level stereo processing pipeline is provided in Figure 2. Our system receives two video streams that can be acquired using gray-level or color filter array (CFA) image sensors. Depending on the sensor features, the incoming video sequences need to undergo different pixel-level pre-processing steps before being processed further. Depending on the camera and the system requirements, all steps explained in the following can be bypassed or enabled.

The architecture first removes sensor noise (*fixed pattern noise (FPN)*) and performs an equalization of the individual pixel responses (*pixel response non-uniformity (PRNU)*). Then, the Bayer CFA intensities are converted to full color values using color interpolation (*Bayer demosaicing*). When employing two or more sensors, pixel intensities and colors are often slightly different among the different sensors; also, beam-splitter-based stereo rigs, often employed in S3D production, introduce additional shifts in color and intensity. Therefore, the streams are corrected in two color correction stages: a linear color correction transforms each stream individually to a rendered color space (e.g., sRGB) and a nonlinear color correction matches the two streams. The survey by Ramanath et al. [15] provides more details on the general color image processing pipeline.

The above-described pre-processing techniques can be implemented very efficiently on FPGAs since only streaming pixel operations and operations on small windows are performed, which require small-line buffers only. To sustain the input data rate, all operations are pipelined. Furthermore, all of the following blocks can be enabled or disabled during run-time. All operations are accurately performed in fixed-point arithmetic.

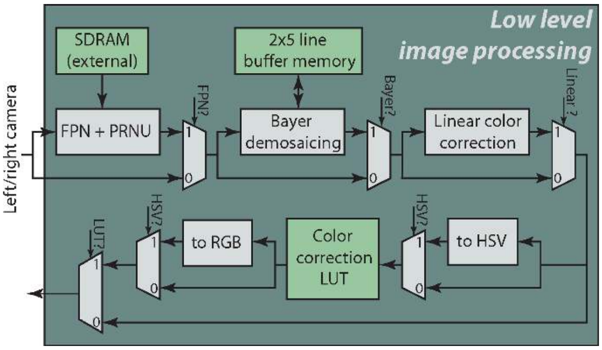

Figure 2 FPGA pipeline of low-level image processing. Only the fixed pattern noise removal and pixel response non-uniformity equalization need access to external memory.

Frame synchronization

The FPGA triggers the cameras at an adjustable frame rate to synchronize the two incoming video streams. However, corresponding pixels from the different cameras do not necessarily arrive at exactly the same time in general, mainly due to slight mismatches of the two camera clocks. To fully synchronize the video streams at pixel level, the incoming video streams are temporarily stored in small line FIFOs to compensate for the time offsets.

FPN and PRNU correction

Fixed pattern noise is inherent to CMOS/CCD sensors and describes the accumulated charges in the pixel sensors that are not due to incoming photons. This noise can be characterized by measuring a *black image* with no light incidence, for a given exposure period. This reference image can then simply be subtracted in the following camera operation. To eliminate temporal noise, multiple black reference images can be averaged for better characterization during the initial calibration phase.

The individual CMOS pixels usually exhibit different response characteristics. This response characteristic can be measured by imaging a homogeneously lit white surface (*white image*). Then, during camera operation, the incoming images can be normalized to the measured intensities from the white level measurement, after FPN correction has been applied. This measurement implicitly includes the white balance.

The *black* and *white* reference images are of the same size as the incoming video frames. Due to the significant amount of data, these images are stored in an external DRAM connected to the FPGA. Because both video streams are synchronized at the pixel level, external memory accesses can be structured to reduce the memory bandwidth requirements. To this end, the reference images for both cameras are stored in interleaved order, such that random accesses are avoided: one value for one pixel of each left and right *black image* and one value for each left and right *white image* can be read together in a single access cycle.

Color filter array demosaicing

Most CMOS sensors apply a color filter similar to the Bayer pattern to capture color images with a single sensor only. In order to reconstruct a full-resolution color image, interpolation filters have to be applied. In this architecture, a linear filter based on the Wiener filter with window size of 5 × 5 pixels is used [16].

The FPGA implementation stores 5 lines for each video in a temporary line cache to perform the interpolation. The arithmetics of the interpolation [16] can be efficiently implemented using adders and bit shifts only, due to the 'hardware friendly' choice of the interpolation coefficients.

Linear color correction

Depending on the lighting conditions and sensor characteristics, a color correction is applied to transform the image to a standardized color space. The 3 × 3 transformation matrix can be obtained from a least-squares estimation between defined values on a color rendition chart and the measured values of the corresponding pixels [15]. The FPGA implementation allows to update the transformation matrix in real-time from the PC system.

Non-linear color correction

To account for non-linear color or intensity shifts between the two sensor outputs, an additional correction step allows for realizing non-linear mappings. The FPGA architecture provides separate lookup tables for all three color channels in RGB or HSV space to be able to apply any kind of component-wise non-linear transformation. If the HSV color space is used, the RGB values are converted to HSV first, color correction is applied, and the values are then converted back to the RGB color space. The lookup tables can, for instance, realize a Gamma correction on the intensity or they can realize color equalization mappings as described in [17,18], for instance.

The necessary FPGA resources for the pre-processing pipeline are listed in Section 6. The pre-processing block consumes approximately 2% of the total logic resources of the target FPGA (see Table 1) and runs at more than 130 MHz, corresponding to a throughput of more than 130 × 10^6 stereo pixels per second.

4 Image warping

Alignment mismatches between two cameras in a stereo camera systems degrade the visual 3D effect [3] and make proper disparity estimation difficult [19]. Perfect alignment is obtained when the two cameras are positioned along a common baseline, and the optical axes of the cameras are parallel to each other and perpendicular to the baseline. In this perfect alignment case, the correspondence of a pixel in one camera image is found on a horizontal (epipolar) line in the other camera image. Matching pixel correspondences therefore reduces to a search along horizontal lines only.

Table 1 FPGA resources on an ALTERA Stratix III EP3SL340.

Unit	Comb. LUTs	Registers	Mem. [kB]	DSP
Pre-proc.	5, 794 (2%)	4, 690 (2%)	26 (1%)	50 (9%)
Warping	29, 283 (11%)	16, 723 (6%)	431 (32%)	187 (32%)
Disp. Estim.	54, 275 (20%)	26, 378 (10%)	140 (7%)	0 (0)
Infrastructure	11, 564 (4%)	14, 577 (5%)	78 (4%)	0 (0)
Total	100, 916 (37%)	62, 368 (23%)	892 (44%)	237 (41%)

The memory column indicates the total amount of block memory used (ALTERA-specific SRAM blocks, M9K/M144K), and the DSP column are ALTERA 18-bit DSP slice elements.

The most prominent alignment mismatches occur from non-linear lens effects (*radial distortion*) and from relative positioning offsets of the cameras. To correct for these effects digitally, the incoming images are first warped to remove any radial and tangential distortion (*undistortion*). Then, the images are projected to a common image plane (*rectification*) to correct for a potential displacement from the common baseline or for a mismatch of the orientation of the optical axes. Figure 3 illustrates the two corresponding correction steps which we are collectively referring to as digital *image warping*.

4.1 Undistortion and rectification

A broad body of work has been published on modeling both distortion and rectification algorithms and on the estimation of the corresponding system parameters. The implemented image warping procedure presented in the following is based on the distortion model of Brown [20] and a homography that corrects for camera misalignment. The corresponding parameters are estimated offline or in regular intervals on the CPU using state-of-the-art calibration procedures.

4.1.1 Radial and tangential distortion

Radial and tangential distortion is caused by imperfections of physical lens systems. In our system, we use the model of Brown [20] that describes the distortion process as follows. First, an undistorted integer pixel coordinate $\mathbf{x_i}$ is normalized with the intrinsic camera parameters: $\mathbf{x} = \text{diag}(\mathbf{f}_c)^{-1}(\mathbf{x}_i - \mathbf{c}_c)$, where $\mathbf{f}_c$ is the 2D focal length vector in pixels and $\mathbf{c}_c$ the principal point

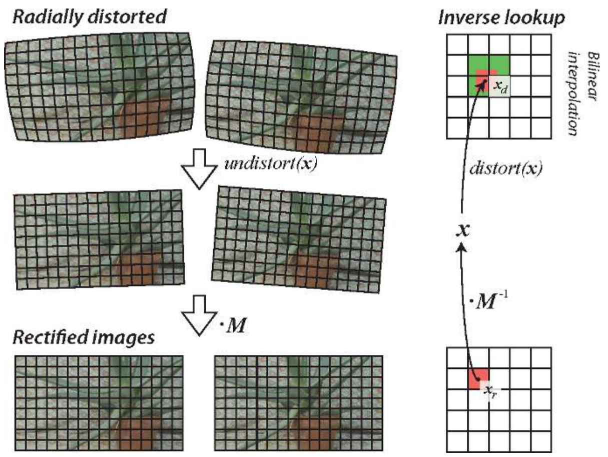

Figure 3 Radial undistortion and rectification. The captured image *(top row)* is transformed to an undistorted and rectified image *(bottom row)*. For each pixel in the rectified image, the respective pixels in the captured image are bilinearly interpolated.

(the geometric center of the image) [19,21]. Next, this normalized and still undistorted image point $\mathbf{x} = (x_1, x_2)^{\mathrm{T}}$ is projected to a distorted image point $\mathbf{x}_d$ according to

$$\mathbf{x}_d = \left(1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6\right)\mathbf{x} + \mathbf{x}_t, \tag{1}$$

where

$$r^2 = {x_1}^2 + {x_2}^2 \tag{2}$$

represents the squared distance to the principal point and where

$$\mathbf{x}_t = \begin{bmatrix} 2\xi_1 x_1 x_2 + \xi_2\left(r^2 + 2{x_1}^2\right) \\ \xi_1\left(r^2 + 2{x_2}^2\right) + 2\xi_2 x_1 x_2 \end{bmatrix} \tag{3}$$

represents the tangential distortion.

The system-specific camera intrinsics $\mathbf{f}_c$, $\mathbf{c}_c$, radial distortion parameters $\kappa = [\kappa_1, \kappa_2, \kappa_3]^{\mathrm{T}}$, and tangential distortion parameters $\boldsymbol{\xi} = [\xi_1, \xi_2]^{\mathrm{T}}$ are determined offline, using camera calibration methods such as the one in [21].

4.1.2 Rectification using homographies

The rectification process performs a 3 × 3 projective transformation on the coordinates. These matrices are often referred to as *homography* matrices and are in general invertible transformations. The rectifying homographies can be estimated with feature correspondences and epipolar geometry properties as described in [22,23].

To perform a projective transformation with eight degrees of freedom on a 2D image point $\mathbf{x} \in \mathbb{R}^2$, the image point needs to be extended to homogenous coordinates $\mathbf{x}_h = [\mathbf{x}, 1]^{\mathrm{T}} \in \mathbb{R}^3$. The point $\mathbf{x}_h$ is mapped to its rectified position $\mathbf{x'}_h$ using the homography matrix $\mathbf{M} \in \mathbb{R}^{3 \times 3}$

$$\mathbf{x'}_h = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = \mathbf{M}\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}. \tag{4}$$

Then, the rectified position can be obtained by projecting the homogeneous coordinates back to $\mathbb{R}^2$ according to

$$\mathbf{x}_r = \begin{bmatrix} x'_1/x'_3 \\ x'_2/x'_3 \end{bmatrix}. \tag{5}$$

In order to be compatible with the Brown distortion model described in the preceding section, the rectification needs to be performed inversely which is done by inverting **M**. Conversely, the inversion of the distortion would be more elaborate since there is no closed-form expression for an undistortion model (inverse Brown model).

4.1.3 Image warping using inverse pixel lookups

The distortion and (inverse) projectivity described by (1) and (5) can be combined into a single coordinate transform. Using the resulting relationship, our implementation performs the image warping using inverse pixel lookups. That is, for each pixel position $\mathbf{x}_r \in \mathbb{R}^2$ in the rectified target image, the corresponding distorted coordinate $\mathbf{x}_d$ in the original image is computed according to (1), which is then used to look up the respective color value (see Figure 3).

The rectification and distortion process usually does not map integer positions to integer positions. A destination pixel position will often be projected in-between multiple source pixels. In this work, we use bilinear interpolation of the neighboring pixels to determine the final pixel-color value. Figure 3 illustrates this process in detail. Note that this scheme can produce aliasing, in cases where the images are shrunk heavily. However, such extreme distortions do not occur in practice when using at least approximately aligned cameras in a stereo rig.

4.2 FPGA implementation

The rectification process performs pixel re-ordering that requires a partially random or unstructured and a-priori unknown access sequence to the large off-chip frame-buffer memory that is needed to store the high-resolution images. The resulting memory-access bottleneck is the main challenge for the implementation of the real-time image warping unit. Our proposed architecture is divided into two units: the warping arithmetic performs backward coordinate transformation and the bilinear interpolation. A dedicated, problem-specific caching and memory interface is employed to provide sufficient memory-accesses bandwidth. Figure 4 shows an architectural overview of our warping core.

4.2.1 Warping arithmetic

The warping arithmetic unit performs the image transformation described in Section 4.1.3. The corresponding hardware is illustrated in Figure 5. Destination pixel coordinates are generated by a 2D counter stepping through the target image. These coordinates are used as input for the inverse radial undistortion block before the coordinates are transformed using the inverse homography block. The necessary divisions (see (5)) are realized with a fully pipelined CORDIC implementation with 16 iterations. The transformed coordinates are used to generate coordinates of the four nearest neighboring pixels and to request the respective color values from the ‘cache + prefetch’ unit. When the corresponding data are available, bilinear interpolation is used to compute the color for the pixel in the target image. Most of the operations are integer operations or require only limited precision. Hence, custom fixed-point arithmetic has been found to be superior to standard floating-point cores in terms of hardware resource usage.

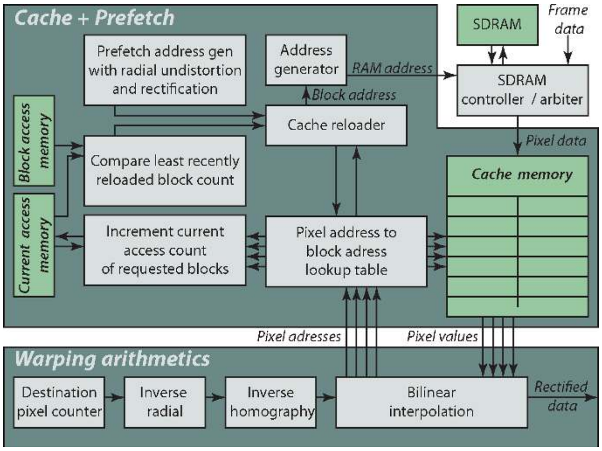

Figure 4 Image warping architecture overview. The cache and prefetch sub-block ensures optimal caching, where every pixel is read only once from external memory. The warping sub-unit computes the necessary coordinate transformations.

4.2.2 Memory bandwidth and access patterns

The warping arithmetic unit requests four pixel values at each clock cycle to perform the bilinear interpolation. Due to the large amount of data of 1080p videos, the incoming high-resolution frames are first stored in an off-chip DRAM before being warped. Without any caching, every pixel will be accessed four times on average. Therefore, the image warping unit will require an external DRAM bandwidth of

$$B = B_{\text{write}} + B_{\text{read}} = 3W\,H f \ + \ 4 \cdot 3W\,H f,$$

bytes per second for a color image (24 bits per pixel) of size $3W\,H$ bytes at frame rate f, when no caching is applied. For 1080p30 stereo video, the required memory bandwidth therefore amounts to approximately 1.8 GB/s. Since the access pattern is strongly dependent on the distortion parameters and homography matrices and therefore not incremental in general, the effective available bandwidth of DRAM components is severely degraded and clearly insufficient to support the desired frame rates.

4.2.3 Caching architecture

Our caching architecture is designed to reduce the DRAM read access bandwidth to approximately $B_{\text{read}}/4$ (by reading each pixel only once) and, more importantly, to increase the access efficiency by reading and writing in longer coherent bursts. To this end, we exploit the fact that the image warping reads pixels that lie approximately along lines, and the lines never intersect. For the cache replacement strategy, we exploit that access patterns remain deterministic for a set of distortion parameters and a given homography. The basic idea is to subdivide the image into sub-blocks, which are preloaded and cached in the order of occurrence during the transformation. As long as the cache is large enough to store a couple of lines, cache misses can be effectively avoided and no pixel is loaded more than once from memory due to the regularity of the transformation. Moreover, by dimensioning the sub-block size properly, only burst-reads in the size of several pixels are performed and the controller efficiency is increased by an order of magnitude compared to random accesses. Figure 6 illustrates the caching principle.

Figure 4 provides an overview of the caching architecture. First, the image is logically subdivided into sub-blocks. The cache and prefetch unit generates the order of first occurrence of each memory block for a given set of image warping parameters online. This order is stored locally and subsequently used to fill the cache blocks optimally. Additionally, the total number of pixel accesses for each block is generated whenever a new set of parameters is loaded and stored in a lookup table. During operation, the current number of pixel accesses is counted for each of the currently used cache blocks. If the number of current pixel accesses matches the predetermined total number of accesses, this cache block can be updated with a new memory block. If the cache memory is configured to hold enough lines, no memory

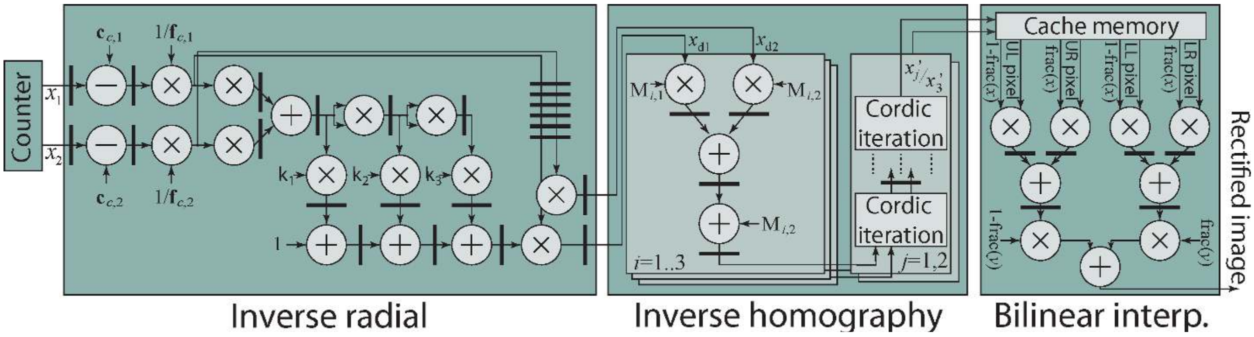

Figure 5 Image warping data path implementation. The design is fully pipelined for maximum throughput. The division in the inverse homography block is realized using a pipelined CORDIC algorithm.

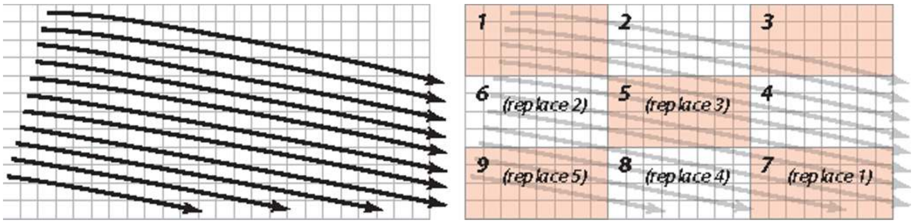

Figure 6 Proposed caching principle. *The left* subfigure illustrates an example access pattern. *The right* subfigure illustrates the cache load and discard sequence.

block needs to be read more than once using this caching strategy. To be able to read four values from cache in parallel, we subdivide the cache into four parallel cache blocks and store values column and row interleaved, since the bilinear interpolation always requires 2-by-2 blocks.

For stereo rectification, two instances of the warping unit are needed resulting in two independent reads and write ports to external DRAM; an arbiter handles these four separate requests. The arbiter is designed to ensure that read and writes to subsequent addresses are sent in bursts, which increases controller efficiency.

4.3 Performance summary

The cache size is configurable at compile-time and is dimensioned to match the type of occurring transformations. For our reference implementation (resource usage provided in Table 1), we use a cache size of approximately 24 1080p lines, which supports even extreme transformations (e.g., rotations by $\pi/4$). It can therefore be considered as worst case value, while in practice smaller cache sizes can be used. The design runs at 130 MHz and the RAM interface provide the required bandwidth of 1080p30 stereo video.

5 Disparity estimation

In a stereo camera setup, an object within the scene is projected to different pixel locations in the two camera views. The displacement (called disparity) between the corresponding points in the two images is inversely proportional to the distance of the object from the cameras. In particular, the depth z of the object in the scene is obtained from the disparity d, the interaxial distance between the two cameras b, and the focal length f according to $z = fb/d$. Disparity estimation is therefore an indispensable tool in stereography to extract three-dimensional scene information from the two images acquired by the two horizontally displaced cameras of a stereo rig.

A wide variety of disparity estimation techniques have been proposed and can be classified into global and local methods (see [13] for a classification scheme for disparity estimation). While global methods usually result in better estimation quality, they require considerable computational complexity and memory bandwidth and are therefore impractical for real-time applications, especially for high-resolution images. The real-time system presented in this paper therefore uses a local method that is able to run at high frame rates and high resolution but still delivers a disparity estimation quality, which is acceptable for camera control [4] and scene analysis.

5.1 Algorithm

The implemented algorithm relies on local, window-based disparity estimation. For each pixel position in one view, the best matching pixel in the other view is found. Matches are computed using a cost function, evaluated on a small window around the pixel positions. Disparities are evaluated for each pixel individually and independently (i.e., local), as opposed to the global methods that optimize the disparities across the full image or a larger neighborhood.

As a result from the rectification step, the corresponding pixel position of a scene object will lie on a horizontal line. The disparity estimation can therefore be limited to a horizontal one-dimensional search with a predefined disparity range. In our implementation, the cost function chosen to determine the best match is the zero mean sum of absolute differences (ZSAD). The algorithm is summarized in Algorithm 1.

```
Data: reference image $I_r$, scan image $I_s$, window size ($w$, $h$), disparity search range [0, $d_{\max}$]
Result: disparity map $d(x, y)$
foreach pixel position ($x$, $y$) of reference image do
    Fetch the (centered) reference window $b_r$ of size ($w$, $h$);
    Fetch the scan window $b_s$ of size ($w + d_{\max}$, $h$);
    foreach horizontal offset $d \in [0, d_{\max}]$ do
        Fetch window $b_{s,d}$ from $b_s$ with size ($w$, $h$) around ($x + d$, $y$);
        Compute ZSAD cost $c(x, y, d)$ between $b_r$ and $b_{s,d}$:
            $c(x, y, d) = \sum_{(x,y)} |b_r(x, y) - \bar{b}_r - b_{s,d}(x, y) + \bar{b}_{s,d}|$
            where $\bar{b}_r$ and $\bar{b}_{s,d}$ denote the mean pixel values.
        end
        $d(x, y) = \mathrm{argmin}_d(c(x, y, d))$
    end
```

Algorithm 1: ZSAD-based local disparity matching.

For an image size of (W, H), i.e., image width W and image height H, and frame rate f images per second, the number of cost functions to be computed per second for a window size (w, h) and disparity range $d_{\max}$ is

$$\mathrm{ops} = W\,H\,f\,whd_{\max}.$$

Unfortunately, increasing the resolution not only increases WH, but also typically requires a larger search range $d_{\max}$. In order to achieve real-time performance for high-resolution video streams, we chose a hierarchical approach to depth estimation. The video streams are downsampled N_s times into a hierarchical pyramid structure, similar to a GPU implementation for disparity estimation described in [14]. The algorithm is illustrated in Figure 7. First, disparity estimation is performed on the lowest resolution image; on the next higher resolution, a search refinement around the upsampled disparity from the previous stage is performed. The refinement range does not need to be larger than +/- 1 pixels, due to the upsampling of two. With the hierarchical architecture, the number of required cost-function computations per second reduces to

$$\mathrm{ops}_{\mathrm{hier}} = W\,H\,f\,wh\left(\left(\sum_{n=0}^{N_s-1}\frac{3}{2^{2n}}\right) + \frac{d_{\max}}{2^{3N_s}}\right). \quad (6)$$

In (6), the sum collects the complexity contributions from the higher resolutions, while the term that is proportional to $d_{\max}$ corresponds to the operations from the first (i.e., lowest resolution) stage. The behavior of the complexity savings, visible from the ratio $\mathrm{ops}_{\mathrm{hier}}/\mathrm{ops}$ for an increasing number of stages, is illustrated in Figure 8 together with the increase in the associated memory requirements.

Algorithmically, using a hierarchical depth estimation has two different effects. On the positive side, depth estimation gets more robust against spatial high-frequency noise and remaining vertical offsets due to imperfect rectification. However, high-frequency structures are lost in lower-resolution images and cannot be recovered in higher-resolution stages. Hence, the best number of stages to use depends on the spatial frequencies to be detected.

5.2 FPGA implementation

Figure 9 provides a high-level view on the hardware architecture of the disparity estimation.

5.2.1 Lowest resolution block

The lowest resolution unit, depicted on the right side of Figure 9, resembles a full disparity estimation unit: for a given *reference window* b_r, the ZSAD costs are computed for all blocks in a *scan window* b_s. This task is carried out by parallel ZSAD units that evaluate the ZSAD cost function for one candidate disparity value in *one clock cycle*. Due to the small effective pixel rate $f_{\mathrm{pixel}} = 2^{-2N_s}W\,H\,f$, achieved by the downsampling, time-sharing can be employed to reduce the number of ZSAD units that need to be instantiated to scan the $2^{-N_s}d_{\max}$ candidates in the subsampled image, which corresponds to a disparity range of $d_{\max}$ in the original image, within the time available for each pixel. Two

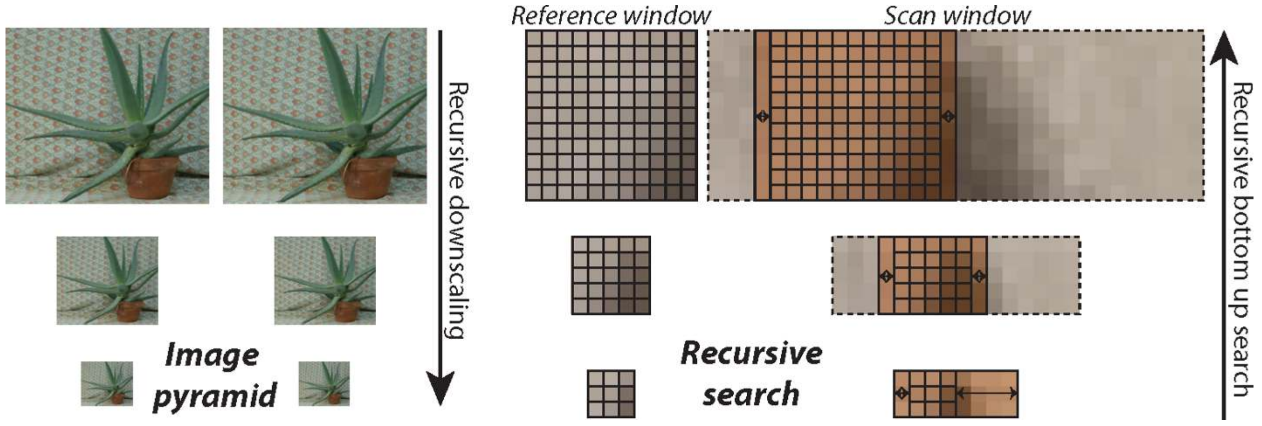

Figure 7 Illustration of hierarchical, local, window-based disparity estimation. *The left* subfigure depicts the image pyramid. The image is iteratively downsampled by a factor of 2 on each stage. *The right* subfigure illustrates the search: the initial search is performed on the lowest resolution (stage 2 in this example). The determined disparity is subsequently refined in a very small search window on all higher resolutions.

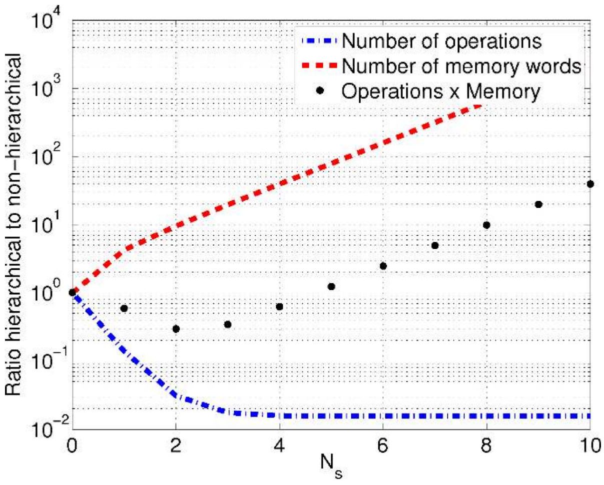

Figure 8 Relative resources of hierarchical depth estimation to standard window-based depth estimation. The lower curve *(blue)* indicates the ratio of necessary window operations, and the upper curve *(red)* indicates the ratio of memory words needed for synchronizing the downsampling.

detailed example implementations with fully parallel ZSAD units and time-shared ZSAD units are shown in Figure 10.

The lowest resolution unit needs to buffer h - 1 lines from the scan- and the reference-images only. One reference window and one block in the scan window need to be accessed in parallel from the buffer by each ZSAD unit. Due to the scan line processing order, the buffer can be efficiently implemented with FPGA SRAM macros in conjunction with a shift register structure.

The ZSAD matching is performed twice, once using the left image as reference and once using the right image as reference window. Although this doubles the necessary hardware resources for ZSAD units and for searching the best match, it allows for a consistency crosscheck performed in the highest resolution stage. The buffer size remains largely identical since on-chip line buffers are implemented for both images in any case.

5.2.2 Higher-resolution stages

The disparities found at the lowest stage are subsequently upsampled and used as offset to perform the matching in the higher-resolution stages. Instead of evaluating the ZSAD matching costs in the full scan area, only three matching windows in a range of +/- 1 pixels from the disparity value found in the previous stage are used. Therefore, only three ZSAD matching units are needed to perform the estimation in one clock cycle.

In addition to the disparity upsampling, the incoming pixel stream is downsampled and forwarded to the next lower stage. A synchronization buffer between incoming pixel values and the corresponding incoming offsets is needed. This buffer temporarily stores the incoming lines until the respective disparities from the lower-resolution stage are available.

In terms of resource usage, the upsampling, downsampling, and synchronization require line buffers and

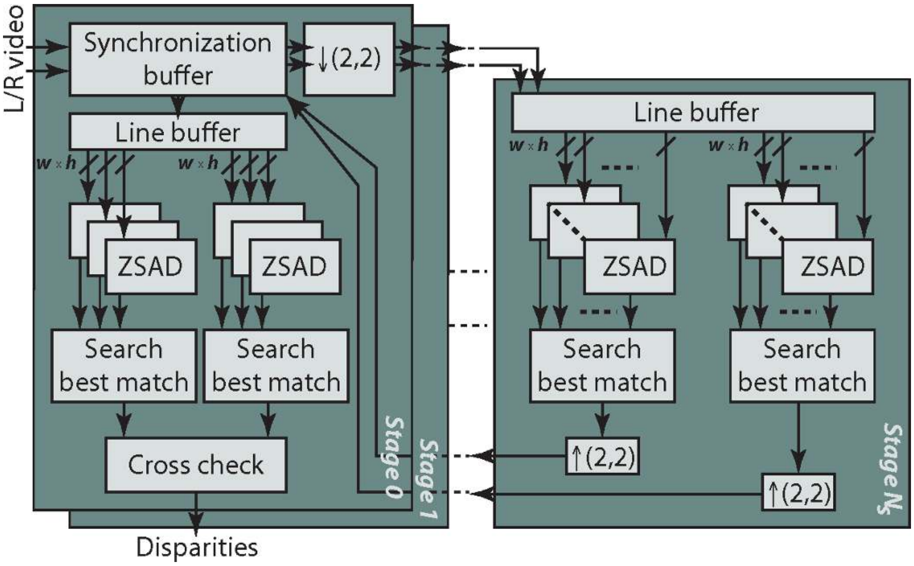

Figure 9 FPGA architecture of the disparity estimation. The higher stages keep the incoming lines in a synchronization buffer, until disparities are propagated from the lowest stages.

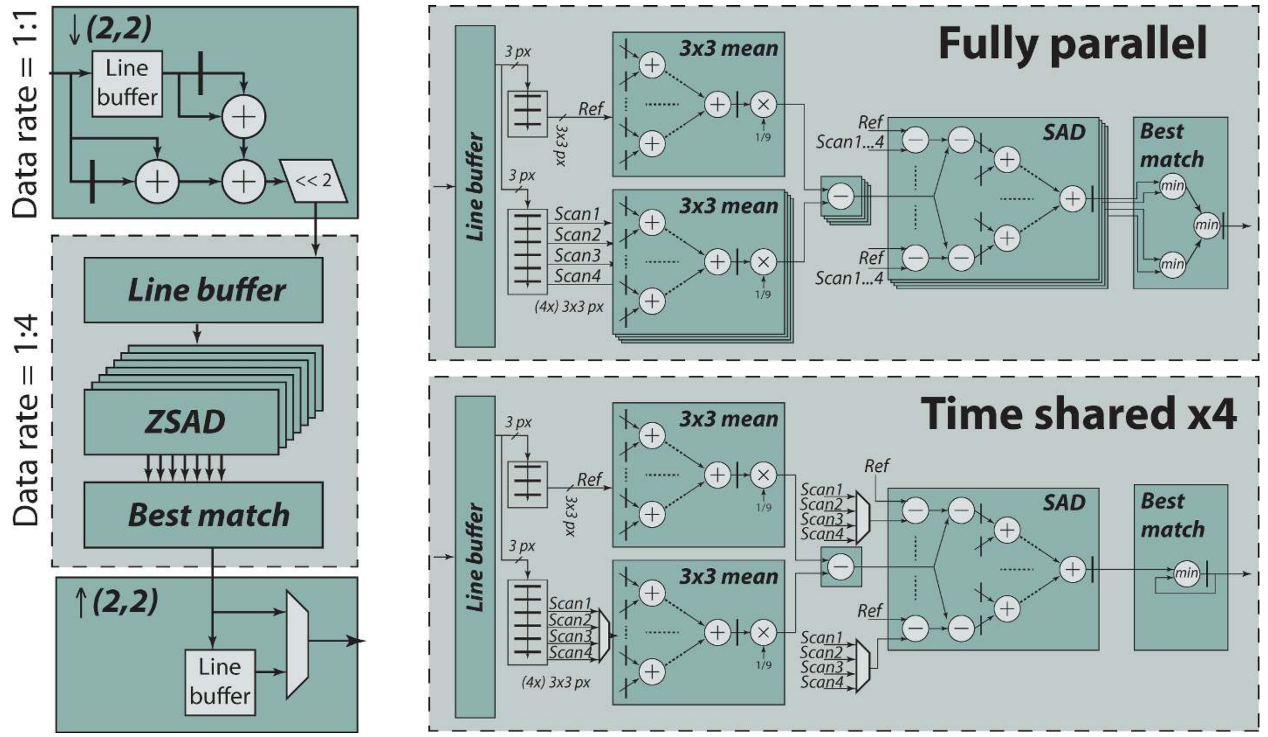

Figure 10 Data path details of one matching stage. For illustration purposes, a simplified example with a matching window size of 3 × 3 pixels and a disparity search range of 4 pixels is shown. Furthermore, only one matching direction is shown (e.g., only left-right match). The illustration on the left side shows a high-level overview of one matching level. Note that the downsampling operation reduces the data rate to $\frac{1}{4}$. A fully parallel implementation *(top-right* illustration) would therefore be un-utilized for many cycles. We employ time-sharing *(lower-right* illustration) using one single matching unit only, which is fully utilized for every cycle without reducing the throughput. Note that the design is parameterized, and the amount of time-sharing is configurable at compile-time.

hence on-chip SRAM memory resources. This is well suited for FPGAs, which are typically equipped with abundant SRAM memory blocks. However, the size of the synchronization buffer grows exponentially with the number of stages, which renders architectures with a high number of stages still too complex even for large FPGAs, unless external memory resources can be used.

5.3 Performance summary

The hierarchical architecture provides considerable complexity savings, and it increases colorimetric and geometric noise resilience. Unfortunately, it also destroys fine-grained details and, for a streaming-based architecture, the exponentially growing amount of synchronization memory limits the number of usable hierarchical stages. The corresponding tradeoff between memory increase and complexity reduction is illustrated in Figure 8 for $d_{\max} = 256$. A good tradeoff is to use $N_s = 2$, which we employed for our reference implementation.

The depth estimation unit can run at up to 130 MHz, which is sufficient for a throughput of 60 frames per second (fps) for full HD resolution with $d_{\max} = 256$. The performance bottleneck of the complete design is currently the path through the adder tree in the ZSAD blocks. The corresponding FPGA resources and performance figures are summarized in Table 1.

6 Results and comparison

The proposed camera architecture has been evaluated using an experimental beam-splitter rig with two synchronized cameras, with a maximum resolution of 2, 048 × 2, 048 pixels at 30 fps. The host platform uses a six-core CPU paired with an NVIDIA GTX480 GPU. The FPGA design is deployed on a PCI Express board fitted with one ALTERA Stratix III with two DDR2 RAMs and a PCIe 8× interface. The design is implemented in VHDL, and a fixed-point golden model (implemented in MATLAB) is used as functional reference. The stereo video stream is captured on two Camera Link interfaces, processed in the FPGA, and finally transferred to the main memory of the PC using DMA. Figure 11 shows our system, and Figure 12 shows different output images from the camera pipeline. For discussions on the *quality of results* achieved by the hierarchical disparity estimation, we refer to [14].

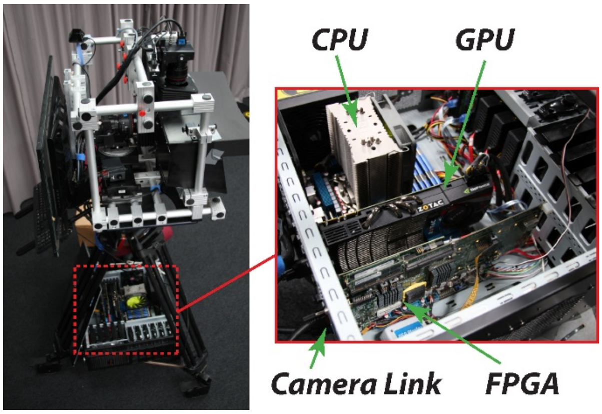

Figure 11 Overview picture of our custom stereo camera system.

6.1 Resource utilization and throughput

The FPGA resource utilization on the ALTERA Stratix III (EP3SL340) FPGA is detailed in Table 1. The entire stereo pipeline fits easily into the device. For the reference implementation, we used a depth estimation with two hierarchical stages and search window sizes of [7 × 7, 7 × 7, 5 × 5], where the last window size corresponds to the lowest resolution stage. Matching is performed in a disparity range of 256 pixels. The rectification unit includes 768 cache blocks, where each block consists of 4 × 16 pixels in RGB format. The total amount of cache memory therefore corresponds approximately to 24 lines of a full-HD image. This generous choice has been made to cover also worst case scenarios, i.e., arbitrary transformations, without throughput degradation and might be reduced by a factor up to five when the necessary image transformations are small.

Our system is running at 130 MHz and is able to process stereo full-HD at 30 fps (1080p30) video in real-time. The PCIe interface to the PC and the Camera Link interface currently limit the system to scale to higher resolutions and/or frame rates. The FPGA design itself currently supports stereo full-HD at 60 fps (1080p60); higher numbers can be reached by using more hardware resources for depth estimation and further DDR2 controller optimization for the rectification. The latencies of the pre-processing and disparity estimation units are only a couple of lines, the latency of the PCI transfer is one frame (double buffering), and the latency of the rectification is also one frame. The rectification latency could be reduced when assuming that the transformations are small and then start the read-out when the frame is still being processed; however, the present implementation does not support this operation.

6.2 FPGA versus GPU

The main parts from the stereo pipeline, depth estimation and image warping, have also been implemented on

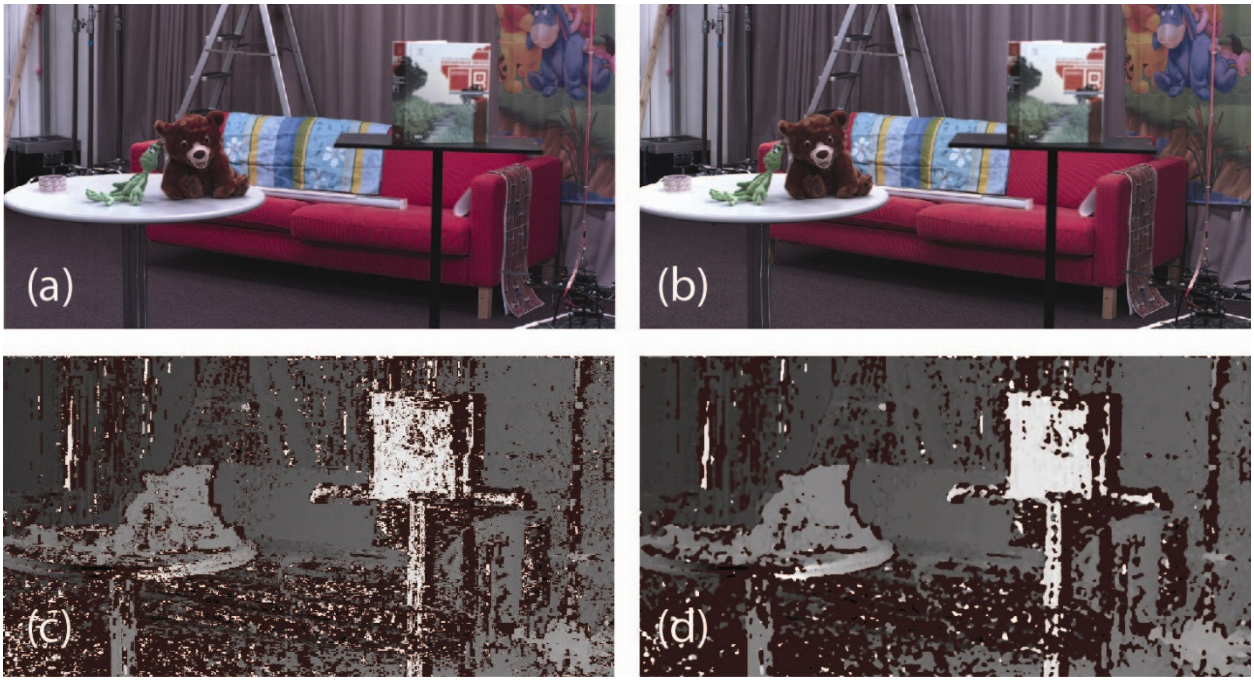

Figure 12 Result images from our stereo pipeline. *Top:* left *(a)* and right *(b)* color processed images. *Bottom:* disparity image, with consistency check (tolerance 5) *(c)* and disparity image with additional median filter (GPU, size 5) *(d)*. Noise in texture-less regions (in particular the left corner) is unavoidable with local disparity estimation algorithms.

the GPU for comparison. Due to the massive amount of caching and memory bandwidth of modern GPUs, rectification does not pose a challenge to the GPU. However, the hierarchical depth estimation requires approximately 190 ms per frame, which is an order of magnitude slower than our FPGA implementation. For this comparison, the same number of parameters has been chosen as for the FPGA implementation. One of the main advantages over a GPU implementation is the parallelism of the FPGA architecture: all different stages are computed in parallel, while the GPU performs all intermediate stages in sequence. Furthermore, memory transfers constitute a considerable overhead for the GPU implementation. Note that these data transfers are not necessary in our stream processing-based FPGA implementation.

6.3 Comparison to related work

The results of this work surpass all related work in terms of throughput and, more importantly, resolution. The comparison in this chapter is limited to the most related work, and earlier related work is covered in the works cited here. Unless stated differently, all FPGA implementations employ a similar FPGA technology.

Morris et al. [6] presented an FPGA implementation for disparity estimation using dynamic programming. The architecture is able to achieve 30 frames per second for 1 million pixel images, but only allows to search a disparity range of 128 pixels. Kalarot et al. [24] compared this implementation to a GPU implementation using a CUDA on GTX 280 and showed superior performance of the FPGA implementation.

Longfield et al. [7] presented a parametrizable FPGA disparity matching core that supports resolutions up to 320 × 240 pixels and computes disparities up to 20 pixels at 30 frames per second.

Georgoulas et al. [8] present an SAD-based depth estimation, reaching up to 768 frames per second for resolution of 640 × 480, with a search range of 80 pixels. The authors claim to be able to reach 251 frames per second at 1280 × 1024, however, at a search range of 80 pixels only.

Gehrig et al. [9] presented a low-power FPGA implementation of semi-global disparity matching that is able to process 640 × 400 resolution at 25 fps with a maximum of 128 pixel disparity.

Recently, Jin et al. [10] presented an FPGA design and implementation of a real-time stereo vision system. Their system rectifies the video stream in real-time and performs disparity estimation using the census transform on 640 × 480 at 230 fps, with a search range of 64 pixels.

Ohmura et al. [11] implemented 752 × 480 at 60 fps using orientation code matching, with a disparity of 127 pixels.

Banz et al. [12] presented a stereo vision system that covers the entire process from noise reduction, rectification, and disparity estimation. They implemented semi-global mapping at 640 × 480 30 fps with 128 pixel disparity range.

Miyajima et al. [25] presented a system that is able to achieve up to 19 fps for 640 × 480 for a maximum disparity size of 200 pixels, however, using older FPGA technology. Their design should thus yield better results on state-of-the-art FPGA devices due to technology scaling.

Unfortunately, all of the mentioned designs scale poorly for videos with higher resolutions, such as 1080p resolutions. Disparity estimation for high-resolution video with realistic search ranges of 256 pixels at real-time frame rates is impossible, due to the non-linear increase in computational complexity and memory accesses. Our work combines an advanced caching scheme for the rectification and hierarchical disparity search in order to allow for real-time performance on high-resolution video.

6.4 Scalability

Our architecture is scalable to higher resolutions and frame rates beyond 1080p60. The most obvious approach is to insert more pipeline registers in the critical path in order to increase the system clock. Furthermore, the disparity estimation block has been designed to be modular, and more calculation units working in parallel can easily be added to improve throughput. The warping block can similarly be extended by increasing the level of parallelization; however, the external memory interface will become the bottleneck eventually.

In addition, we can also scale the design toward multi-camera setups with more than two cameras easily. The resources linearly scale with the number of cameras, since for each stream a new instance of the pipeline can be employed.

7 Conclusion

In this work, an FPGA-based processing platform for high-resolution stereo video is presented. The platform is integrated into a PC platform with CPU, GPU, and FPGA processing. The FPGA architecture performs low-level image processing, color interpolation, and cross-image color correction. Furthermore, real-time rectification and disparity estimation are performed on 1080p video at 30 frames per second. In order to cope with the high data rates and computational complexity, an advanced caching scheme for the rectification and hierarchical disparity estimation is employed.

The pipeline is implemented on FPGA using a modest amount of hardware resources, which frees the CPU and GPU to concentrate on higher-level tasks. We show that image pre-processing can be implemented very efficiently with low hardware overhead. Furthermore, we show that almost arbitrary radial undistortions and homography transformations can be implemented using a cache of the size of a couple of lines only, effectively reducing the amount of off-chip data transfers to a feasible minimum. We show that disparity estimation can be performed in real-time on 1080p video using modest hardware resources by employing a hierarchical evaluation scheme.

In favor of execution speed and hardware resources, a local, window-based disparity estimator is applied. Other, global methods can yield better results but also imply higher execution times. Possible future work could explore hierarchical global methods for hardware implementations. The rectification unit can produce aliasing artifacts in case of extreme minification, e.g., if the pixel distances are reduced by more than a half. Low-pass pre-filtering of the input video could resolve the problem. However, most stereo cameras employ the same lenses and cameras, and therefore, this extreme case can safely be neglected.

Endnotes

[a]The 'de-normalization' is implicitly contained in the homography; [b]Width and height are configurable, and their optimal choice depends on the transformation. Typically, blocks should be much wider than high, since the transformation rotation angle is usually small, after manual coarse rectification; [c]The number of necessary memory words is proportional to 2^{N_s}.

Acknowledgements
The authors want to thank Simon Wright, Dinu Kaufmann, Martin Polak, Dany Sünnen, and Stefan Zwicky for discussions and/or for helping with the implementation.

Author details
[1]ETH Zurich, 8092 Zurich, Switzerland [2]Disney Research Zurich, Zurich, Switzerland [3]EPF Lausanne, Lausanne, Switzerland

Competing interests
The authors declare that they have no competing interests.

Received: 25 March 2011 Accepted: 4 November 2011
Published: 4 November 2011

References

1. Afshari H, Jacques L, Bagnato L, Schmid A, Vandergheynst P, Leblebici Y: The PANOPTIC camera: a plenoptic sensor with real-time omnidirectional capability. *Journal of Signal Processing Systems* 2011.
2. Vancea C, Nedevschi S: LUT-based image rectification module implemented in FPGA. *Intelligent Computer Communication and Processing* IEEE; 2007, 147-154.
3. Mendiburu B: *3D Movie Making–Stereoscopic Digital Cinema from Script to Screen* Elsevier; 2008.
4. Heinzle S, Greisen P, Gallup D, Chen C, Saner D, Smolic A, Burg A, Matusik W, Gross M: Computational stereo camera system with programmable control loop. *ACM Transactions on Graphics* 2011, **30(4)**:94.
5. Zilly F, Mueller M, Eisert P, Kauff P: The stereoscopic analyzer–an image-based assistance tool for stereo shooting and 3D production. *IEEE International Conference on Image Processing* 2010.
6. Morris J, Jawed K, Gimel'farb G, Khan T: Breaking the 'Ton': achieving 1% depth accuracy from stereo in real time. *Image and Vision Computing New Zealand* 2009, 142-147.
7. Longfield S, Chang M: A parameterized stereo vision core for FPGAs. *Field Programmable Custom Computing Machines* 2009, 263-266.
8. Georgoulas C, Andreadis I: A real-time occlusion aware hardware structure for disparity map computation. *Proceedings of Image Analysis and Processing* 2009, 721-730.
9. Gehrig SK, Eberli F, Meyer T: A real-time low-power stereo vision engine using semi-global matching. *Proceedings of Computer Vision Systems* 2009, 134-143.
10. Jin S, Cho J, Pham X, Lee K, Park S, Kim M, Jeon J: FPGA design and implementation of a real-time stereo vision system. *IEEE Transactions on Circuits and Systems for Video Technology* 2010, **20**:15.
11. Ohmura I, Mitamura T, Takauji H, Kaneko S: A real-time stereo vision sensor based on FPGA realization of orientation code matching. *International Symposium on Optomechatronic Technologies (ISOT)* 2010, 1-5.
12. Banz C, Hesselbarth S, Flatt H, Blume H, Pirsch P: Real-time stereo vision system using semi-global matching disparity estimation: architecture and FPGA-implementation. *International Conference on Embedded Computer Systems (SAMOS)* 2010, 93-101.
13. Scharstein D, Szeliski R: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision* 2002, **47**:7-42.
14. Zach C, Karner K, Bischof H: Hierarchical disparity estimation with programmable 3D hardware. *Short communications of WSCG* 2004, 275-282.
15. Ramanath R, Snyder W, Yoo Y, Drew M: Color image processing pipeline. *IEEE Signal Processing Magazine* 2005, **22**:34-43.
16. Malvar H, He LW, Cutler R: High-quality linear interpolation for demosaicing of Bayer-patterned color images. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing* 2004, **3**:485-488.
17. Porikli F: Inter-camera color calibration by correlation model function. *Proceedings of the International Conference on Image Processing* 2003, **2**:133-136.
18. Pitié F, Kokaram A, Dahyot R: Automated colour grading using colour distribution transfer. *Computer Vision and Image Understanding* 2007, **107(1-2)**:123-137.
19. Hartley R, Zisserman A: Multiple View Geometry in Computer Vision. Cambridge University Press, New York; 2003.
20. Brown DC: Decentering distortion of lenses. *Photogrammetric Engineering* 1966, **32**:444-462.
21. Bouguet JY: Camera calibration toolbox for Matlab. 2010 [http://www.vision.caltech.edu/bouguetj/calib_doc/].
22. Zilly F, Mueller M, Eisert P, Kauff P: Joint estimation of epipolar geometry and rectification parameters using point correspondences for stereoscopic TV sequences. *Proceedings of 3DPVT* 2010.
23. Mallon J, Whelan P: Projective rectification from the fundamental matrix. *Image and Vision Computing* 2005, **23(7)**:643-650.
24. Kalarot R, Morris J: Comparison of FPGA and GPU implementations of real-time stereo vision. *IEEE Computer Vision and Pattern Recognition* 2010, 9-15.
25. Miyajima Y, Maruyama T: A real-time stereo vision system with FPGA. *Field-Programmable Logic and Applications* 2003, **2778**:448-457.

doi:10.1186/1687-5281-2011-18
Cite this article as: Greisen *et al.*: An FPGA-based processing pipeline for high-definition stereo video. *EURASIP Journal on Image and Video Processing* 2011 **2011**:18.