J Real-Time Image Proc (2016) 11:83–91
DOI 10.1007/s11554-012-0304-3

ORIGINAL RESEARCH PAPER

Real-time HD image distortion correction in heterogeneous parallel computing systems using efficient memory access patterns

Rui Melo · Gabriel Falcao · João P. Barreto

Received: 19 March 2012 / Accepted: 24 November 2012 / Published online: 12 December 2012

Abstract High-definition video is becoming a standard in clinical endoscopy. State-of-the-art systems for medical endoscopy provide 1080p video streams at 60 Hz. For such high resolutions and frame rates, the real-time execution of image-processing tasks is far from trivial, requiring careful algorithm design and development. In this article, we propose a fully functional software-based solution for correcting the radial distortion (RD) of HD video that runs in real time in a personal computer (PC) equipped with a conventional graphics processing unit (GPU) and a video acquisition card. Our system acquires the video feed directly from the digital output of the endoscopic camera control unit, warps each frame using a heterogeneous parallel computing architecture, and outputs the result back to the display. Although we target the particular problem of correcting geometric distortion in medical endoscopy, the concepts and framework herein described can be extended to other image-processing tasks with hard real-time requirements. We show that a heterogeneous approach, as well as efficient memory access patterns in the GPU, improve the performance of this highly memory-bound algorithm, leading to frame rates above 250 fps.

R. Melo (✉) · J. P. Barreto
Department of Electrical and Computer Engineering, Faculty of Science and Technology, Institute for Systems and Robotics, University of Coimbra, 3000-214 Coimbra, Portugal
e-mail: rmelo@isr.uc.pt

J. P. Barreto
e-mail: jpbar@isr.uc.pt

G. Falcao
Department of Electrical and Computer Engineering, Faculty of Science and Technology, Instituto de Telecomunicações, University of Coimbra, 3030-290 Coimbra, Portugal
e-mail: gff@co.it.pt

Keywords Radial distortion · Heterogeneous architecture · Medical endoscopy · Concurrent processing · GPU optimization

1 Introduction

Endoscopic images are usually acquired using a flexible or rigid lens coupled to a CCD sensor. The lens is introduced into the patient's body through a small port for visualizing the anatomical cavities during surgery and diagnosis. In this type of procedure, the endoscopic video is the only guidance for the medical practitioner and therefore the system has to provide the best imaging quality possible [1]. The latest endoscopic systems use 1080p video (1,920 × 1,080 pixels/frame) at 60 Hz as the standard for visualization. The HD resolution and the high frame rate allow an enhanced visualization of the structures and therefore are likely to significantly improve the surgeon's perception [2].

Due to the small size of the lens, endoscopic images present strong RD (also known as barrel distortion). RD is a nonlinear geometric deformation of the image that moves the points radially toward the center and can severely affect the notion of depth [3]. Several authors have addressed the problem of RD correction in wide angle lenses [4–7], but these either are not suited for real-time HD processing [4, 5] or do not fully model the endoscopic camera [6, 7].

Advances in very large-scale integration (VLSI) systems using the distortion model estimation proposed in [4] showed promising results in the correction of RD using dedicated hardware. Asari presented in [8] an efficient VLSI architecture to correct the RD in wide-angle camera images by mapping the algorithmic steps onto a linear array. Later, in [9], a pipelined architecture was presented that was able to process images at a rate of 30 Mpixels/s. In [10], the authors proposed a VLSI implementation for RD correction that reduced in 61 % the number of cells compared to [9] and achieved a throughput of 40 Mpixels/s. The recent work presented in [11] reduced at least 69 % hardware cost and 75 % memory requirement compared to previous works. In [12], the authors presented a comparison of RD correction implementations on a homogeneous multi-core processor, a heterogeneous cell broadband engine, and an FPGA. They concluded that only an FPGA and a fully optimized version of the code running on the cell processor could provide real-time processing speed (30 fps for input images of 2,592 $\times$ 1,944, which translates into a throughput of 150 Mpixels/s).

While previous software-based implementations fail to process large amounts of data in real time or do not fully model the endoscopic camera, hardware-based solutions lack the versatility to adapt to different devices or lenses (and therefore changes in the projection model in real time) and involve additional costs and effort to implement.

In this work, we propose a system for acquiring and processing the HD video feed from an endoscope in real time using a conventional PC equipped with an acquisition board and a GPU. Our solution is based on the work of [13] that updates the endoscopic camera projection model [6], according to the possible lens rotation at each frame time instant. Our system acts like a plug-and-play module that captures the video feed, processes each frame on a regular PC, and then outputs the result back into the existing visualization system (see Fig. 1). We verify that a homogeneous multi-core CPU is not capable of supporting HD real-time video distortion correction, as observed in [12], and the GPU-based implementation of [13] also fails to deliver the necessary frame rates for the latest endoscopic devices. Our framework for correcting the radial distortion of a HD video stream is based on a heterogeneous implementation that uses both CPU and GPU concurrently. We demonstrate that a hybrid solution, where the computational workload is distributed across the CPU and the GPU in parallel, enables the processing of the video feed (1,920 $\times$ 1,080 pixels/frame) at frame rates up to 250 fps (500 Mpixels/s throughput) when implementing efficient memory access patterns on the GPU side of the heterogeneous parallel system.

2 Radial distortion correction in clinical endoscopy

This article is closely related to the work presented in [13]. While [13] describes the camera projection model, the calibration of the endoscope in the OR, and the estimation of the relative rotation of the lens scope, the current article

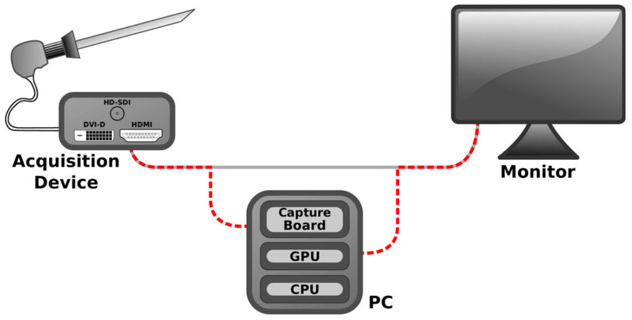


Fig. 1 Proposed system scheme. The video feed is captured directly from the video output of the acquisition device, processed in our heterogeneous system, and then sent back into the existing visualization system

addresses in detail the problem of efficient implementation for real-time execution in HD endoscopic devices.

In [13], the endoscopic camera is calibrated from a single image of a chessboard pattern [14] with the radial distortion being described by the so-called division model [15] that uses a single parameter ξ to quantify the amount of image deformation. Consider $\mathbf{X}$ the 3D coordinate of a point in the world reference frame. The corresponding point $\mathbf{x}'$ in the image plane is determined by the projection Eq. 1:

$$\mathbf{x}' \sim \mathsf{K}\boldsymbol{\Gamma}_{\xi}(\mathsf{P}\mathbf{X}), \tag{1}$$

where $\sim$ denotes an equality up to a scale factor, K is the well-known intrinsics matrix obtained by the camera calibration and P denotes the standard 3 $\times$ 4 projection matrix [3]. $\boldsymbol{\Gamma}_{\xi}$ is the radial distortion non-linear function that maps world undistorted points $\mathbf{x}_{\mathrm{u}} \sim (x_{\mathrm{u}}\ y_{\mathrm{u}}\ z_{\mathrm{u}})^{\mathsf{T}}$ into the corresponding world distorted point:

$$\boldsymbol{\Gamma}_{\xi}(\mathbf{x}_{\mathrm{u}}) \sim \left(2x_{\mathrm{u}}\ \ 2y_{\mathrm{u}}\ \ z_{\mathrm{u}} + \sqrt{z_{\mathrm{u}}^2 - 4\xi(x_{\mathrm{u}}^2 + y_{\mathrm{u}}^2)}\right)^{\mathsf{T}}. \tag{2}$$

Assuming that the 3D point $\mathbf{X}$ is represented in the camera reference frame, $\mathsf{P} \sim (\mathsf{I}_{3\times 3}\ 0_{3\times 1})$ and therefore we can compute the distorted image coordinates of an undistorted image point $\mathbf{x}'_u$ (in pixels) by:

$$F(\mathbf{x}'_{\mathrm{u}}) \sim \mathsf{K}\boldsymbol{\Gamma}_{\xi}(\mathsf{K}_y^{-1}\mathbf{x}'_{\mathrm{u}}). \tag{3}$$

where K_y^{-1} maps the undistorted image point $\mathbf{x}'_{\mathrm{u}}$ into a canonical plane, specifying certain desired characteristics of the undistorted image (e.g., center, resolution) [13].

The camera calibration changes during operation because the doctor rotates the lens scope with respect to the CCD head. As discussed in [13], the problem can be solved by considering an adaptive projection model that takes into account this relative rotation. The authors devised an efficient algorithm for extracting the image boundary and detecting the lens mark (Fig. 2) that relies on the extraction

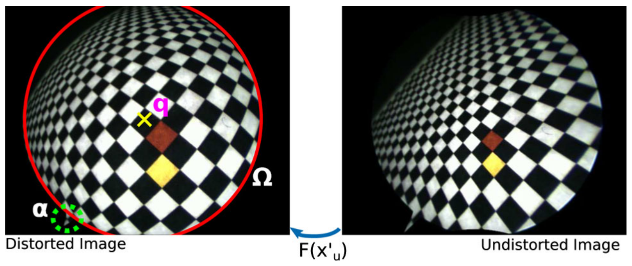


Fig. 2 RD correction of an imaged chessboard pattern. In order to adapt the projection model to the lens rotation, we have to determine the meaningful region boundary Ω, the triangular mark that indicates the relative rotation α between the camera head and the lens, and the lens rotation center coordinates q in the image

of boundary contour points on the GPU, as well as standard methods implemented on the CPU (such as RANSAC [16] and Kalman filtering [17]) to deliver a robust estimation of the rotation parameters. With this new adaptive projection model, the intrinsics matrix is updated by a rotation around the lens rotation center $\mathbf{q}$ and the distortion mapping of Eq. 3 becomes:

$$F(\mathbf{x}'_u) \sim \mathsf{K}_i \Gamma_\xi(\mathsf{R}_{-\alpha_i,\mathbf{q}''_i} \mathsf{K}_y^{-1} \mathbf{x}'_u). \tag{4}$$

where $\mathsf{K}_i \sim \mathsf{R}_{\alpha_i,\mathbf{q}_i}\mathsf{K}$ is the intrinsics matrix updated according to the lens rotation α at time i and $\mathsf{R}_{-\alpha_i,\mathbf{q}''_i}$ is a rotation matrix that rotates the warping result back to the original orientation.

Figures 2 and 3 show the results of the RD correction in different environments using the mapping function in Eq. 3, where the effects of the distortion are easily noticed.

3 Proposed system

GPUs have emerged as powerful processors in the last few years. The recent introduction of the CUDA interface [18] has enabled the scientific community to parallelize computationally intensive algorithms and to achieve faster execution times [19]. The Nvidia GF100 and subsequent architectures (also known as Fermi) introduced significant improvements in memory accesses and a drastic increase in compute capability when compared with the previous G80 and GT200 families. In Sect. 4, we perform experiments using both older architectures (G80 and GT200) and the more recent Fermi family.

The proposed system consists of a regular workstation equipped with a GPU and an HD acquisition board. The HD video feed is captured through the acquisition board and the image is transferred to the GPU that performs part of the processing. At the end, the resulting corrected image is displayed onto the visualization system through the OpenGL buffer. Figure 4 illustrates the processing steps for each frame of the video feed.

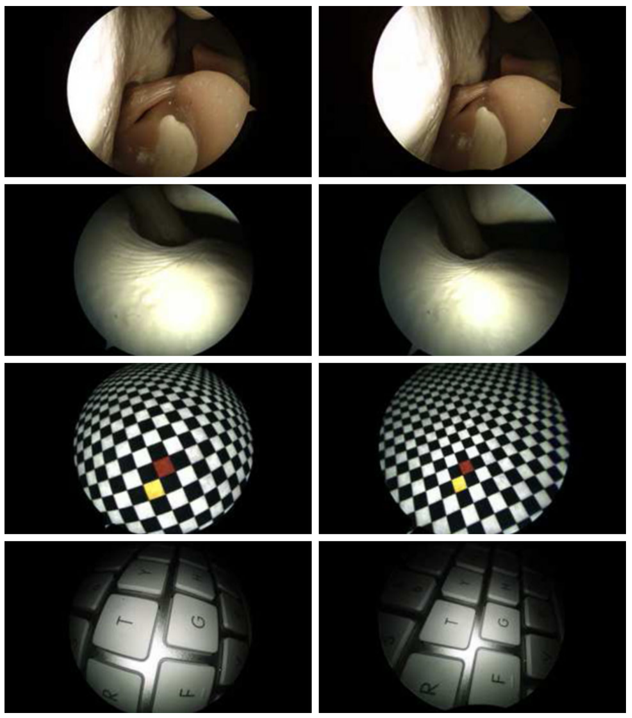


Fig. 3 Result of the RD correction in different environments. The *left column* shows the original image and the *right column* the image after distortion correction

The system runs in a heterogeneous environment with one POSIX thread [20] handling the GPU device calls and the other performing the serialized CPU processing necessary to extract the meaningful region boundary of the image mentioned in Sect. 2.

The system is divided into four main processing blocks:

- Colorspace conversion: After transferring the image into the GPU (in YUV422 format), a colorspace and grayscale conversion is performed. Each RGB channel plus the grayscale value are written to the global memory and later bound to the texture memory space of the GPU.
- Boundary detection: Using the grayscale image and the previous boundary estimation parameters from the CPU thread, the boundary contour points are extracted using the procedure described in [13] and the result is passed to the CPU to compute the boundary for the next iteration.
- RD correction: Using the R, G and B channel textures and the previous boundary estimation parameters, the RD is corrected on the GPU and the result is written to the OpenGL global memory buffer for visualization.
- CPU thread: The CPU thread is responsible for robustly estimating the boundary contour from a set of contour points extracted on the GPU. This procedure involves a RANSAC [16], low pass, and EKF [17] to robustly fit

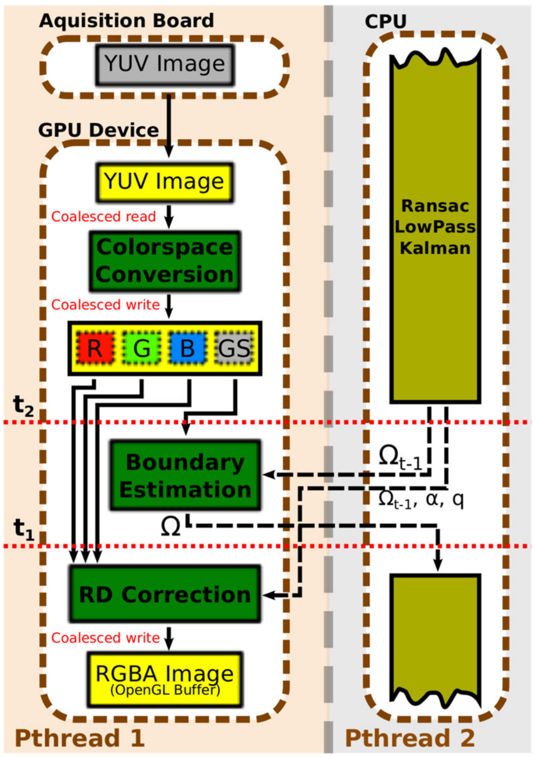


Fig. 4 Processing stages of the RD correction system. The system runs in two POSIX threads. *Pthread 1* is responsible for the acquisition and processing of the acquired frame on the GPU. *Pthread 2* is responsible for the serial parts of the algorithm running on the CPU. The threads are synchronized trough conditional variables placed at the *red horizontal dashed lines*. *Pthread 2* is launched at t_1 and delivers the previous boundary estimation Ω_{t-1} as well as the rotation parameters (α, q) to *Pthread 1* that waits at t_2. In this way, the system processes the current frame based on the previous boundary estimation

an ellipse to the boundary contour and estimate the lens rotation parameters.

Figure 5 shows the processing blocks execution sequence each time a frame arrives from the video stream, where we can observe the concurrent execution on the CPU and GPU. Note that the execution is perfectly balanced between CPU and GPU when the *boundary detection − CPU* time approaches the *RD correction + result image display + image acquisition + colorspace conversion* time.

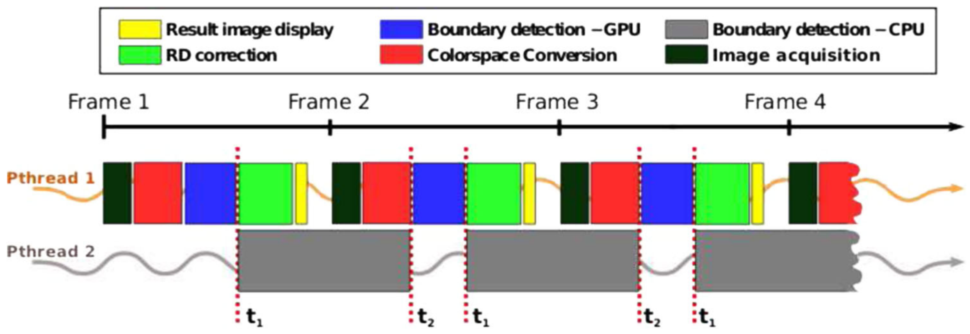
3.1 Image acquisition

The video feed is captured directly from the endoscope's control unit video output using the YUV422 transmission format. Taking human perception into account for chrominance components, the YUV422 format encodes 2 RGB pixels into a single YUV quadruple. This is of great importance when implementing real-time systems since this video format significantly reduces the necessary bandwidth for transmission and, consequently, the latency of the video stream without compromising the image quality. Moreover, as shown in Fig. 6, the memory alignment of the YUV422 image is perfectly suited to fulfil the GPU's optimized memory access patterns presented in this article.

3.2 Heterogeneous processing

As opposed to previous works [4–6, 8–12], where only the problem of RD correction using a static projection model is solved, we address the RD correction under projection model changes due to the possible endoscopic probe rotation. The update of the projection model requires additional computation for determining the boundary contour of the meaningful region of the image and the relative lens rotation [13]. To achieve higher processing performance, we execute both GPU and CPU code concurrently. As shown in Figs. 4 and 5, we split the processing into two POSIX threads: (1) *Pthread 1* is responsible for acquiring the image and performing the CUDA API calls for converting the image colorspace, extracting the boundary contour points, and correcting the RD; (2) *Pthread 2* is responsible for performing the serial part of the boundary contour estimation. This includes a RANSAC, low pass, and EKF operations that are detailed in [13]. The high processing frame rate of the system (more than 250 fps, as depicted in Sect. 4) allows the RD correction of the current frame (at time t) based on the boundary parameters of the

Fig. 5 Image-processing timeline sequence for a generic video stream



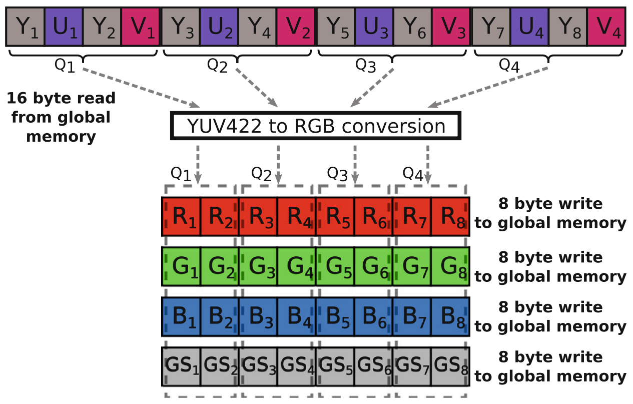

Y1 U1 Y2 V1 Y3 U2 Y4 V2 Y5 U3 Y6 V3 Y7 U4 Y8 V4
Q1 Q2 Q3 Q4
16 byte read from global memory
YUV422 to RGB conversion
Q1 Q2 Q3 Q4
R1 R2 R3 R4 R5 R6 R7 R8
8 byte write to global memory
G1 G2 G3 G4 G5 G6 G7 G8
8 byte write to global memory
B1 B2 B3 B4 B5 B6 B7 B8
8 byte write to global memory
GS1 GS2 GS3 GS4 GS5 GS6 GS7 GS8
8 byte write to global memory

Fig. 6 Memory access pattern per thread for the *colorspace conversion* kernel. Each thread in a half warp accesses a 16-byte word from the global memory (four YUV422 quadruples). For each YUV422 quadruple, the thread computes the two corresponding R, G, B and grayscale values and packs them into 8-byte words that are written to the corresponding global memory location. Since the data are aligned, the 16 threads of a half warp read a total amount of $16 \times 16 = 256$ byte and write $16 \times 8 = 128$ byte for each image channel plus the gray scale using single memory load/store instructions

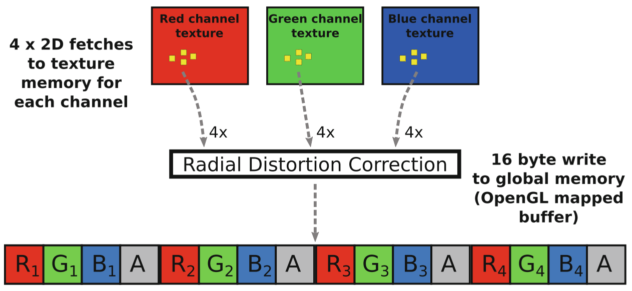

Red channel texture
Green channel texture
Blue channel texture
4 x 2D fetches to texture memory for each channel
4x
4x
4x
Radial Distortion Correction
16 byte write to global memory (OpenGL mapped buffer)
R1 G1 B1 A R2 G2 B2 A R3 G3 B3 A R4 G4 B4 A

Fig. 7 Memory access pattern per thread for the *RD correction* kernel. For each group of 4 RGBA pixels, the radial distortion kernel thread computes the corresponding locations in the distorted space. As the resulting coordinates do not necessarily fall into the regular lattice of the input image, the data are retrieved through 2D texture memory fetches that, through the built-in interpolation hardware, perform the bilinear interpolation of the value. Each thread in a half warp fetches four elements of each channel texture and the result is packed into a 16-byte word (consisting of 4 RGBA pixels) and written to the global memory (that is mapped to an OpenGL buffer). The data to be written into the global memory by the 16 threads of a half warp is perfectly aligned and therefore the operation is coalesced into a single memory write instruction

previous frame (at time $t - 1$) without compromising the accuracy of the correction. By using both the CPU and GPU concurrently, we are able to hide the serialized CPU processing workload, as shown in the results of Sect. 4, and therefore substantially increase the system's performance.

3.3 Efficient GPU memory accesses

The GPU section of the system presented in Fig. 4 carries most of the workload for correcting an HD frame. Since the RD correction problem is mainly memory bound, we devised efficient memory accesses to/from the slow device's global memory to hide data accesses' latency. The optimization of the device's global memory accesses is based on a specific memory alignment procedure, known as *coalescence* that allows reducing the global number of memory accesses. In this way, threads that are processed simultaneously in batches of 16 (known as half warps) by one multiprocessor can perform the corresponding memory accesses during the same clock cycle.

In the *colorspace conversion* kernel of Fig. 4, each thread of a half warp accesses the global memory data as a 16-byte aligned array corresponding to four YUV422 quadruplets (Fig. 6). Each quadruplet is decomposed into two RGB pixels and the data are packed into 8-byte words for writing in global memory (each channel and the grayscale value are stored into different memory locations). In this way, the 16 threads of a half warp read a total amount of $16 \times 16 = 256$ byte data and write $16 \times 8 = 128$ byte for each image channel plus the grayscale image into the global memory. Since data are perfectly aligned, the global memory read/writes are totally coalesced into single memory load/store accesses.

In the *RD correction* kernel, each thread of a half warp fetches four texture values from the texture memory of the GPU and interpolates the result using the built-in bilinear interpolation hardware. The retrieved values are interlaced into 4 RGBA quadruplets and therefore the write operations requested from the 16 threads of a half warp are coalesced into a single 256-byte memory transaction (Fig. 7).

4 Experimental results

4.1 Experimental setup

Since the heaviest workload is distributed on the GPU, we conducted a series of experiments using different GPUs and different HD resolution inputs. We performed experimental tests on four Nvidia GPUs belonging to three distinct architectures: (1) a GTX580 (Fermi architecture) with 16 multiprocessors and a total of 512 CUDA cores running at a clock speed of 1,544 MHz; (2) a high-end C2050 (Fermi architecture) with 14 multiprocessors and a total of 448 CUDA cores running at a clock speed of 1,150 MHz; (3) a 9800GT (G80 architecture) with 14 multiprocessors and 112 CUDA cores running at 1,500 MHz; and (4) a GTX260M (GT200 architecture) with 14 multiprocessors and 112 CUDA cores at 1,375 MHz. For each different hardware, we tested the code using the uncoalesced accesses to the GPU's global memory implementation and

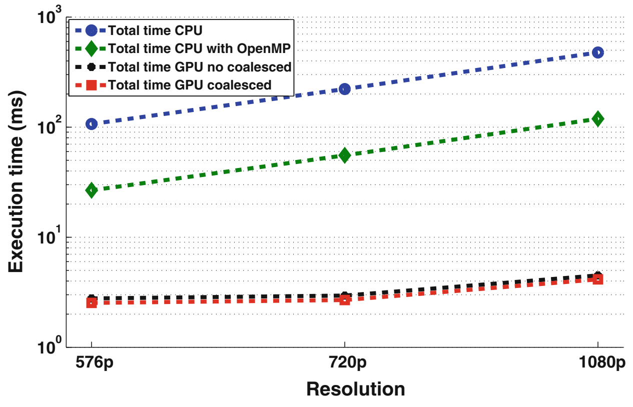


Fig. 8 Comparing CPU and GPU execution times for correcting the radial distortion of HD images. The times represent the mean time needed to correct each frame of the video stream at different resolutions

also the optimized coalesced version on a sequence of 450 frames. The code is written in C++ using CUDA 4.0.

4.2 Time profiling

Figure 8 compares the processing time of four different implementations of our distortion correction algorithm: (1) a naive purely CPU based solution; (2) a hypothetical CPU version using OpenMP[1] directives [21]; (3) our heterogeneous approach using a GTX580 GPU without efficient memory accesses; and (4) our heterogeneous approach using a GTX580 GPU and efficient memory access patterns. The CPU used in the experiment is an Intel®Core™2 Quad CPU running at 2.40 GHz. The comparison given in Fig. 8 shows that the CPU is not able to handle the distortion correction in HD images even when parallelizing the code throughout the multiple CPU cores.

Figure 9 and Table 1 show the mean time needed to process each frame of the input video stream at different resolutions using the four GPUs mentioned above. The times were computed by correcting a sequence of 450 endoscopic video frames and computing the mean time per frame for each resolution used. It can be seen that the system can handle full HD resolution at frame rates above 60 Hz when using the efficient global memory access patterns. The best processing time for a 1,920 × 1,080 video resolution is achieved with the coalesced implementation in the GTX580 GPU. With this setup, the system

[1] OpenMP can be used for shared-memory architectures, such as conventional commercially available off-the-shelf many-core CPUs of the x86 family. As the workstation CPU has four processor cores, OpenMP can be used to parallelize the serial code into all the cores [21]. Doing so would allow achieving a theoretical maximum speedup of 4x relatively to the current CPU code (although the speedup achieved with this kind of parallelization usually does not reach those theoretical maximum values).

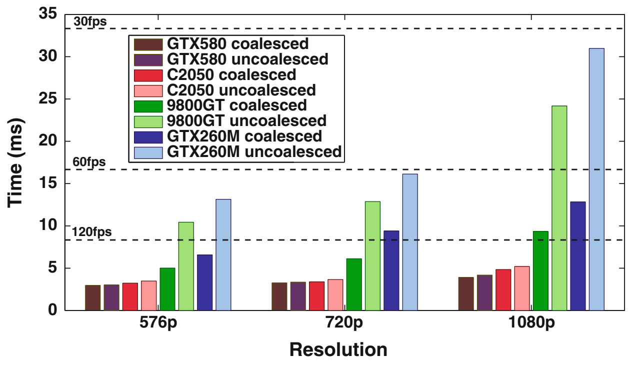


Fig. 9 Mean total time per frame of the system for different GPUs at different resolutions. Both implementations using coalesced and uncoalesced memory accesses are compared. The resulting output frame size of the system is equal to the input resolution. All four devices can process 1080p video resolution at 60 fps when using coalesced accesses to global memory

is capable of correcting the RD of endoscopic images at a frame rate of approximately 250 fps (500 Mpixels/s throughput).

Figure 11 shows the temporal profile of each part of the system individually. It can be seen that, as expected, the use of efficient memory access patterns significantly decreases the processing time of the colorspace conversion and RD correction kernels. Note that the boundary detection on the CPU (textured bar) is overlapped because it runs concurrently with the GPU code (see Figs. 4 and 5).

4.3 Scalability

Concerning the computation on the GPU presented in Fig. 11 and Table 2, we expect a lower gain in performance when applying our efficient memory access patterns on the C2050 and GTX580 GPUs, since Fermi architectures perform intrinsic memory access optimizations when accessing misaligned data from the global memory space. By coalescing data accesses to global memory, we obtain gains of 6.6 and 3.7 % in the kernel execution time for the C2050 and GTX580 GPUs, respectively, and approximately 25 % for the older GPUs. This represents a 7 and 63 % reduction in the total computation time for the Fermi and G80/GT200 architectures, respectively.

The graphic of Fig. 10 shows the performance of our solution as a function of the number of processing cores in the GPU. It can be seen that, by using a GTX580 GPU with 512 CUDA cores, we achieved a processing time 19.5 % inferior to the time of the system equipped with a C2050 GPU (448 CUDA cores). Figure 10 shows that the proposed solution is scalable and that it should suit future requirements of this type of medical imaging systems that expectedly will consist of higher HD image resolutions and frame rates.

Table 1 Mean total time per frame in milliseconds for the different hardwares tested at the resolutions used in Fig. 9

	CPU	GTX260M		9800GT		C2050		GTX580	
		u.	c.	u.	c.	u.	c.	u.	c.
576p	106.75	13.15	6.56	10.44	5.01	3.48	3.23	3.02	2.97
720p	221.72	16.12	9.40	12.87	6.11	3.65	3.39	3.35	3.26
1080p	**476.72** (2 fps)	30.98	**12.84** (77 fps)	24.18	**9.35** (106 fps)	5.20	**4.84** (206 fps)	4.18	**3.90** (256 fps)

The bold values highlight the processing time of our optimized solution at the higher resolution available

Table 2 GPU occupation percentage for host–device transfers (H-D), device–device transfers (D-D), and kernel executions for the *colorspace conversion*, *boundary detection* and *RD correction* (Kernels) in the different GPUs

	H-D (%)	D-D (%)	Kernels (%)	Total GPU (ms)
GTX580 u.	20.8	27.6	26.3	3.37
GTX580 c.	22.5	30.8	**22.6**	3.07
C2050 u.	21.2	29.8	40.3	4.2
C2050 c.	24.7	32.6	**33.7**	3.8
9800GT u.	5.0	9.0	85.3	23.2
9800GT c.	13.8	22.2	**59.6**	8.3
GTX260M u.	4.7	10.0	78.9	29.8
GTX260M c.	12.1	21.4	**51.1**	11.6

The bold values highlight the kernels' occupancy dropdown when using optimized memory acceses

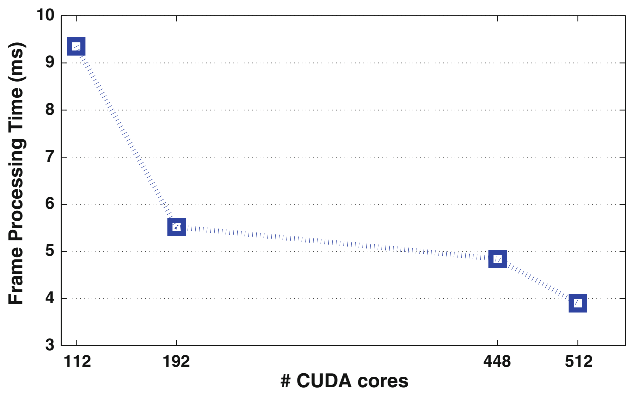


Fig. 10 Execution time per frame of our system processing a 1080p video stream as a function of the number of cores available on the GPU

4.4 Discussion

Table 2 shows the difference in GPU occupancy while using the efficient memory access patterns proposed. We can observe that, since the coalesced accesses to the memory significantly reduce the transfer times, the overall time is decreased and the GPU occupancy is more balanced across data transfers and kernel executions.

As shown in Fig. 5, as long as the boundary detection on the CPU (running on Pthread 2) does not exceed the sum of the *image acquisition*, *colorspace conversion*, *RD correction*, and image display processing times (running on Pthread 1), the CPU computation of the boundary is entirely hidden by the GPU processing. For example, observing Fig. 11d, at 576p resolution for a GTX580 GPU, we can see that the CPU time (grey bar) is higher than the concurrent GPU stages execution time (*image acquisition* + *colorspace conversion* + *RD correction* + *image display*). In this case, the CPU is the bottleneck of the proposed system performance. On the other hand, in most of the remaining setups, the GPU execution is always higher than the CPU execution. The implementation of a heterogeneous system significantly increased the overall performance of the system, truly balancing the workload distribution between CPU and GPU.

5 Conclusion and future work

In this article, we proposed a software-based system for correcting the RD in endoscopic images that is capable of correcting 1080p HD images at 250 fps. The proposed solution is based on a heterogeneous parallel computing architecture that uses both the CPU and the GPU concurrently to process the HD video feed, and not only corrects the RD but also adapts the projection model according to the endoscopic lens rotation. Moreover, we perform memory access optimizations on the GPU that turn out to be fundamental for achieving higher processing frame rates and real-time execution on both new and older GPU architectures. With this work, we proved that a careful and efficient usage of conventional hardware outperforms current software-based solutions and competes with dedicated hardware-based and heterogeneous cell implementations of the RD correction in wide angle lens. Our solution is scalable and will support GPUs with even more processing cores, reducing the video processing times and potentially supporting upcoming video systems, such as 4kUHD (3,840 × 2,160) or 8kUHD (7,680 × 4,320).

The presented HD image-processing pipeline can be extended for purposes other than RD correction, such as stereo reconstruction or visual SLAM for computer-assisted

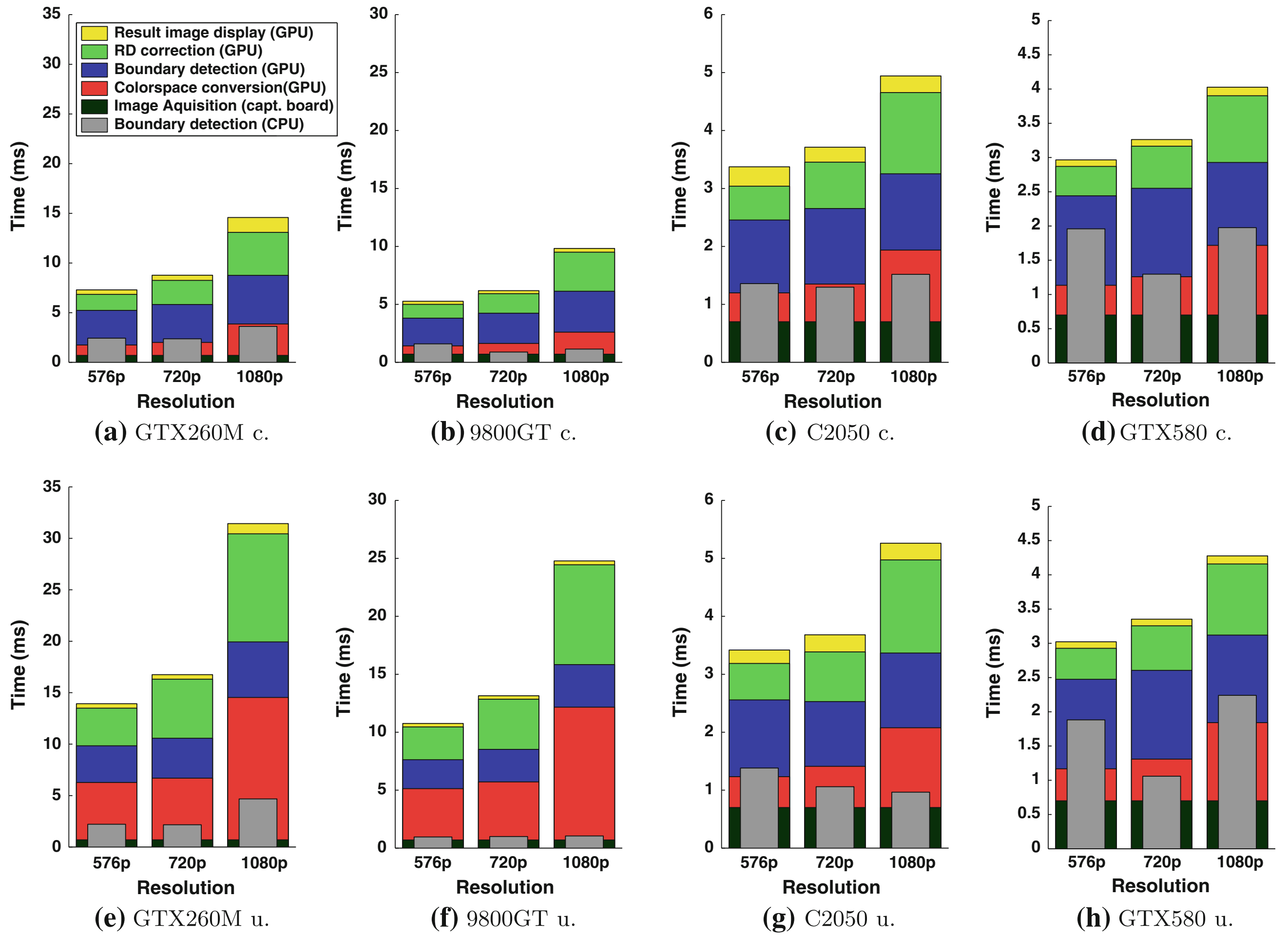


Fig. 11 Time profile of the processing stages for the system using coalesced (c.) and uncoalesced (u.) memory accesses to the GPU's global memory. The *colorspace conversion* time also includes the transfer of the input image from the host to the device (GPU)

surgery. As future work, we will port the code to many-core systems (multiple CPUs/GPUs, for example) to increase the computational capabilities of the system and support more complex image processing in the pipeline.

Acknowledgments This work was supported by the Portuguese Agency for Innovation (ADI) through the QREN co-promotion project 2009/003512 sponsored by the Operational Program "Factores de Competitividade" and the European Fund for Regional Development.

References

1. Soper, N.J.: Mastery of Endoscopic and Laparoscopic Surgery. Lippincott reverend and adventurer, Dhaka (2008)
2. Pierre, S.A., Ferrandino, M.N., Simmons, W.N., Fernandez, C., Zhong, P., Albala, D.M., Preminger, G.M.: High definition laparoscopy: objective assessment of performance characteristics and comparison with standard laparoscopy. J. Endourol. **23**(3), 523–528 (2009)
3. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, Cambridge (2004). ISBN 0521540518
4. Asari, K., Kumar, S., Radhakrishnan, D.: A new approach for nonlinear distortion correction in endoscopic images based on least squares estimation. IEEE Trans. Med. Imaging **18**, 345–354 (1999)
5. Vogt, F., Krüger, S., Niemann, H., Schick, C.H.: A system for real-time endoscopic image enhancement. MICCAI, LNCS 2879, pp. 356–363 (2003)
6. Hartley, R., Kang, S.B.: Parameter-free radial distortion correction with centre of distortion estimation. In: ICCV, vol. 2, pp. 1834–1841 (2005)
7. Jeught, S.V., Buytaert, J., Dirckx, J.: Real-time geometric lens distortion correction using a graphics processing unit. Opt. Eng. **51**, 1–5 (2012)
8. Asari, K.V.: Design of an efficient VLSI architecture for non-linear spatial warping of wide-angle camera images. J. Syst. Archit. **50**(12), 743–755 (2004)
9. Ngo, H.T., Asari, V.K.: A pipelined architecture for real-time correction of barrel distortion in wide-angle camera images. IEEE Trans. Circuits Syst. Video Technol. **15**(3), 436–444 (2005)
10. Chen, P.-Y., Huang, C.-C., Shiau, Y.-H., Chen, Y.-T.: A VLSI implementation of barrel distortion correction for wide-angle camera images. Trans. Circuits Syst. **56**, 51–55 (2009)
11. Chen, S.-L., Huang, H.-Y., Luo, C.-H.: Time multiplexed VLSI architecture for real-time barrel distortion correction in video-endoscopic images. IEEE Trans. Circuits Syst. Video Technol. **21**(11), 1612–1621 (2011)

12. Daloukas, K., Antonopoulos, C.D., Bellas, N., Chai, S.M.: Fisheye lens distortion correction on multicore and hardware accelerator platforms. In: IEEE International Symposium on Parallel Distributed Processing (IPDPS), pp. 1 –10 (2010)
13. Melo, R., Barreto, J.P., Falcao, G.: A new solution for camera calibration and real-time image distortion correction in medical endoscopy: initial technical evaluation. IEEE Trans. Biomed. Eng. **59**(3), 634–644 (2012)
14. Barreto, J., Roquette, J., Sturm, P., Fonseca, F.: Automatic camera calibration applied to medical endoscopy. In: BMVC, London (2009)
15. Fitzgibbon, A.W.: Simultaneous linear estimation of multiple view geometry and lens distortion. In: CVPR, vol. 1, pp.125–132 (2001)
16. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**, 381–395 (1981)
17. Bozic, S.M.: Digital and Kalman Filtering. Edward Arnold, London (1979)
18. NVIDIA.: CUDA Programming Guide 4.0, June 2011. Rev 1
19. Kirk, D., Hwu, W.: Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann, San Francisco (2010)
20. Butenhof, D.R.: Programming with POSIX Threads. Addison-Wesley Longman Publishing Co. Inc., Boston (1997)
21. Chapman, B., Jost, G., Van Der Pass, R.: Using OpenMP: Portable Shared Memory Parallel Programming (Scientific Computation and Engineering Series). The MIT Press, Cambridge (2008)

Author Biographies

Rui Melo received the Integrated Master's degree (BSc + MSc) in Electrical and Computers Engineering from the University of Coimbra, Coimbra, Portugal, in 2009. Since 2009 he is a computer vision researcher at the Institute for Systems and Robotics, Coimbra. He is currently a Ph.D. student at the University of Coimbra, Portugal. His main research interests are computer vision and parallel programming.

Gabriel Falcao graduated at the University of Porto (FEUP), Portugal, where he also concluded a MSc. degree in the area of digital signal processing in 2002. In 2010 he received a Ph.D. in Electrical and Computer Engineering from the Faculty of Science and Technology of the University of Coimbra (FCTUC), Portugal, where he is currently an Assistant Professor. In 2011 he became a Visiting Professor at EPFL, in Switzerland. He is also a Researcher at the Instituto de Telecomunicações and his scientific interests include high-performance and parallel computing, hybrid computation on heterogeneous systems and digital signal processing algorithms, namely those related with biomedical engineering. Gabriel is a Member of the IEEE and IEEE Signal Processing Society.

João P. Barreto (M'99) received the "Licenciatura" and Ph.D. degrees from the University of Coimbra, Coimbra, Portugal, in 1997 and 2004, respectively. From 2003 to 2004, he was a Postdoctoral Researcher with the University of Pennsylvania, Philadelphia. He has been an Assistant Professor with the University of Coimbra, since 2004, where he is also a Senior Researcher with the Institute for Systems and Robotics. His current research interests include different topics in computer vision, with a special emphasis in geometry problems and applications in robotics and medicine. He is the author of more than 30 peer-reviewed publications. He is also regular reviewer for several conferences and journals, having received 4 Outstanding Reviewer Awards in the last few years.