Real-time video breakup detection for multiple HD video streams on a single GPU

Jakub Rosner[a], Hannes Fassold[b], Martin Winter[a], Peter Schallauer

[a]Silesian University of Technology, PhD faculty of Data Mining,
Ulica Academicka 16, 44-100 Gliwice, Poland
[b]JOANNEUM RESEARCH, Institute for Information and Communication Technologies,
Steyrergasse 17, 8010 Graz, Austria

ABSTRACT

An important task in film and video preservation is the quality assessment of the content to be archived or reused out of the archive. This task, if done manually, is a straining and time consuming process, so it is highly recommended to automate this process as far as possible. In this paper, we show how to port a previously proposed algorithm for detection of severe analog and digital video distortions (termed *"video breakup"*), efficiently to NVIDIA GPUs of the Fermi Architecture with CUDA. By parallizing of the algorithm massively in order to take usage of the hundreds of cores on a typical GPU and careful usage of GPU features like atomic functions, texture and shared memory, we achive a speedup of roughly 10–15 when comparing the GPU implementation with an highly optimized, multi-threaded CPU implementation. Thus our GPU algorithm is able to analyze nine Full HD (1920 x 1080) video streams or 40 standard definition (720 x 576) video streams in *real-time* on a *single* inexpensive Nvidia Geforce GTX 480 GPU. Additionally, we present the *AV-Inspector* application for video quality analysis where the video breakup algorithm has been integrated.

Keywords: real-time video breakup, video quality, video distortions, GPU, Fermi architecture, CUDA

1. INTRODUCTION

Manual inspection of video and film material, in order to assess the condition of the material (how severely it is affected by defects and impairments like noise, 'broken' video streams, drop-outs, blocking etc.), is a time consuming and therefore expensive process. Furthermore, the assessment of the severity of various defects in material is subjective and varies between different human operators, even when instructed in the same way.

For these reasons, a fast and automated way to assess the quality of video is highly desirable. In this work, we will address a specific defect, called *video breakup*. Video breakup can occur in analog material which is then digitized or also in digital video streams. As can be seen from Figure 1, its appearance and severity varies significantly. In analog sources, it often appears as geometric and radiometric distortion of one or multiple consecutive rows, typically caused by tape transportation problems. In digital sources, it is often a result of loss of data in the video stream and appears as significant blocking artifacts or as a partial loss of the image content.

In a recent paper [7] a detector for video breakup was proposed, which gives acceptable detection rates of ~ 80 % and false alarm rates of ~ 0.4 false alarm rates per minute. The downside of the algorithm is the fact that, requiring a runtime of 70 milliseconds per Full HD (1920x1080) image on a modern Quad-Core CPU, it is too slow for real-time analysis – especially when analyzing of one or more High Definition video streams as e.g. required in large archive migrating processes.

In this work, we focus on the efficient implementation of this detector on the GPU and show how it is integrated into the AV-Inspector, a prototype of a video quality analysis application. We implement the algorithm for NVIDIA GPUs of the Fermi architecture with CUDA. The Fermi architecture was introduced in 2010 with the GeForce 400 series and introduces a couple of major changes (e.g. the introduction of a full memory hierarchy with L1/L2 cache, larger amount of shared memory, significantly faster atomic operations etc.) compared with previous NVIDIA GPU architectures. The rest of the paper is organized as follows: In section 2 we mention some related work. In section 3 an overview of the recently proposed, original video breakup detector is given. In section 4 the CUDA implementation of the video breakup detector for Fermi architecture is described in detail and in section 5 we present a comparison regarding runtime and

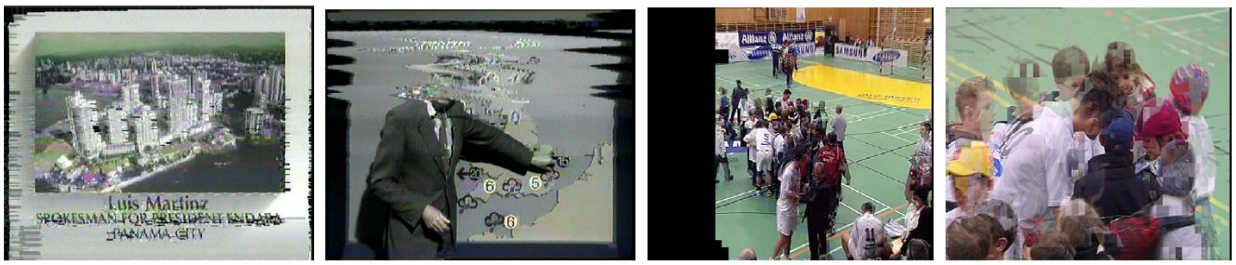

Figure 1: Different appearances of video breakup defects for analog (left two images) and digital (right two images) source material.

quality of the CPU and GPU implementation. Finally, in section 6 the AV-Inspector application is presented which allows for a comfortable analysis of videos and a fast visualization and verification of the detection results.

2. RELATED WORD

Not much work has been published regarding content-based detection of video breakups so far. In [5] an algorithm is presented which aims to detect video packet loss due to transmission errors. The algorithm calculates various measures (intensity variance, horizontal and vertical gradients) for the macroblocks of one image, which are then combined to calculate the severity of the impairment for the image. By nature, it works only for material that has been digitally compressed during transmission.

There are a couple of works which focus on the usage of the most recent NVIDIA Fermi architecture for various scientific areas. Many of them can be found when browsing the session catalogue (presentations[1], posters) of the GPU technology conference 2010. Up to our best knowledge, this is the first work which deals with the usage of Fermi GPUs for video quality assessment related tasks.

3. VIDEO BREAKUP ALGORITHM

In the following, we give an outline of the original algorithm for detection of video breakup defects (see Figure 2 for a schematic workflow) as proposed in [7]. Please note that a comprehensive motivation of the building blocks of the proposed video breakup algorithm and an extensive evaluation of its detection capability (recall, number of false detections per minute, etc.) on a large set of test sequences can be found in [7].

The initial input of the algorithm are two temporally consecutive images I_{n-1} and I_n of the video. As a first step, the images are converted to gray and the difference image $D_n = I_{n-} - I_{n-1}$ is calculated. In order to make the algorithm more robust against content where fast camera or object motion occurs, optionally one of the images can be registered (motion-compensated) before calculating the difference image, e.g. with fast GPU-based optical flow methods [3][6]. In this work, we will employ the basic video breakup algorithm without motion compensation.

Based on the difference image, the original video breakup algorithm proceeds now by calculating two measures, the *row change* measure and the *edge ratio* measure described later. By an appropriately weighted combination of these measures it is possible to calculate a severity measure and decide whether the current frame is affected by a video breakup defect or not.

[1] http://www.nvidia.com/object/gtc2010-presentation-archive.html

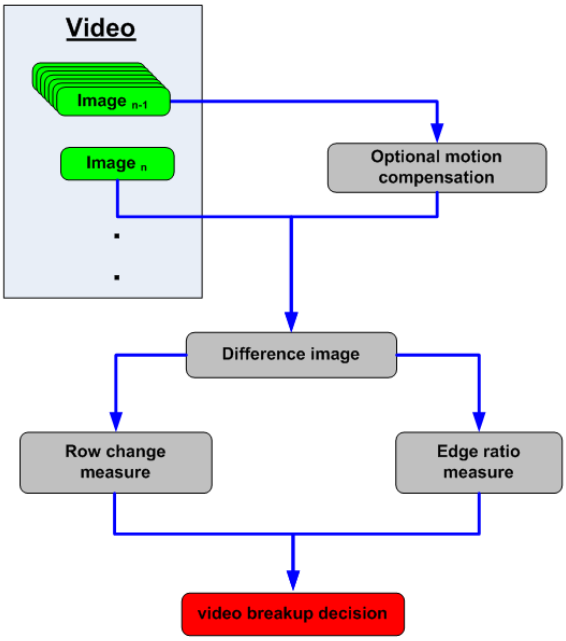

Figure 2: Schematic workflow of the video breakup algorithm for two consecutive frames of. a video.

3.1 Row change measue

First each row of the difference image D_n is compacted into a single column vector v_n, where each element of the vector is calculated as the *average* value of its respective row (other measures for representing the 'typical' intensity of the respective row as e.g. the median would be also possible). Now, the row change measure ρ is calculated as $\rho = dist\ (v_n, v_{n-1})$, where *dist*() is a suitable distance function for vectors, and v_{n-1} is the vector kept up from the previous row change measure calculation for the images I_n and I_{n-1}. We employ the H_6 distance measure from [8] as distance function.

3.2 Edge ratio measure

From the difference image D_n, first the horizontal and vertical edge image is calculated by applying e.g Prewitt or Sobel filters in the respective direction. Afterwards, the pixel-wise edge ratio (ratio of horizontal to vertical edge response) is calculated, resulting in an edge ratio image E. Finally, all edge ratio values in E are summed up to retrieve the edge ratio measure.

4. GPU IMPLEMENTATION

This section describes the GPU implementation and issues that were resolved in order to port the video breakup algorithm efficiently to Fermi architecture GPUs. We assume some basic knowledge about the CUDA programming environment (as can be found e.g. in [2] or [4]).

Before any GPU processing can start the required video frames need to be transferred to the previously allocated GPU memory. Usually the impact of alloction and deallocation of data on the whole processing time is small or even negligible, however in our case it takes at least 30% of the processing time, depending on the frame resolution.

To minimize this time we employ a single context object to store the required buffers that has a lifetime of the whole video analyzing process. This means that the context object will be allocated for the first pair of video frames and its buffers will be reused up until the last two frames are processed after which the context object and its buffers are released.

Note that the computation of the grey image from an color image and the calculation of the difference image D_n map nicely to the GPU, so we will not discuss these steps of the GPU implementation and will focus on the more difficult parts.

4.1 Row change measure

The calculation of the sum (or average) of elements of an array on the GPU is not easy to parallelize. One has to notice that, while the task is trivial on the CPU, GPU algorithm (like the one used in the NVIDIA CUDPP library[2]) typically have to use sophisticated tree-based reduction methods. However, these methods do not utilize well the resources of the GPU when they are applied on small to medium sized arrays as is the case in our application. Here, the typical row size will be between 500 (for SD images) and 2000 (Full HD images) elements. So we developed an alternative approach which calculates for each row an 8-bit histogram, from which then the sum is calculated in a second step.
A further advantage of the histogram-based approach is its flexibility, as it allows us to easily replace the average measure with another measure which is computable from the histogram, like the median or an L_p norm.
Since the difference images have typically the majority of their values close to zero, especially after motion compensation, creating a histogram on GPU suffers from many attempts to increment the same element, which get serialized and badly hurts performance when atomic increment operations are used. For that reason we create *multiple* histograms in shared memory for each row and assign consecutive threads to consecutive histograms. This way we assure that threads of the same warp that increment elements that are close together (and therefore tend to have the same value) access different histograms. To retrieve now the actual histogram for one row, these individual histograms are merged together.
Still such an implementation requires the usage of atomic increment functions, but a significantly lower amount of them is necessary. On the other hand, these atomic operations have got significantly faster[3] on the Fermi architecture by an factor of 5 – 20, so a moderate usage of them is perfectly acceptable.
The second step of the algorithm computes for each row the row sum from its histogram and divides it by the image width to get the average value. Here each thread is responsible for computing the sum of a specific row.
Finally, the vector v_n is transferred from GPU to CPU memory and the distance calculation $\rho = dist\,(v_n, v_{n-1})$ is done on the CPU. The reason for doing it on the CPU is that it is a linear-complexity calculation on a medium sized array and therefore would severely under-utilize the resources of a modern GPU.

4.2 Edge ratio measure

The GPU implementation of the Sobel filter is relatively straightforward. The video frame is divided into rectangular regions that are processed by different thread blocks.
We compute the Sobel filter in shared memory, where consecutive threads in a single warp process elements from consecutive rows in order to achieve highest performance with no bank conflicts as we pad the shared memory matrix[4] approbiately. Then we compute the ratio of horizontal to vertical gradients and write the results back to global memory.
Note that for processing the data in shared memory we need to allocate an additional shared memory buffer of size 32 * 32 * 4 = 4096 bytes to store the intermediate results. Each thread block will therefore require almost 9 KB of shared memory, so it is essential to configure the kernel call to prefer shared memory over L1 in order to be able to process more thread blocks per multiprocessor.
For the final summation of all elements of the edge ration image, we employ a highly efficient tree-based compaction algorithm[5] from the NVIDIA CUDPP library. Since here we sum up a large number of elements and not only a single row, the tree-based algorithm will achieve much better performance than the histogram-based we introduced for the row sum calculation.

5. EXPERIMENTS AND RESULTS

In this section we will describe the results from comparing our CUDA implementation against an optimized reference CPU implementation. The CPU implementation uses OpenMP for parallelizing work across all CPU cores and the Intel Performance Primitive library[6] 6.1 which provides an extensive set of heavily optimized image processing routines.

---

[2] http://gpgpu.org/developer/cudpp

[3] http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[4] The concept and application of shared memory padding – which is important for achieving good performance when using shared memory - is explained in section 3.2.2 of the CUDA Best Practices Guide at http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Best_Practices_Guide.pdf

[5] http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/scan/doc/scan.pdf

[6] http://software.intel.com/en-us/articles/intel-ipp/

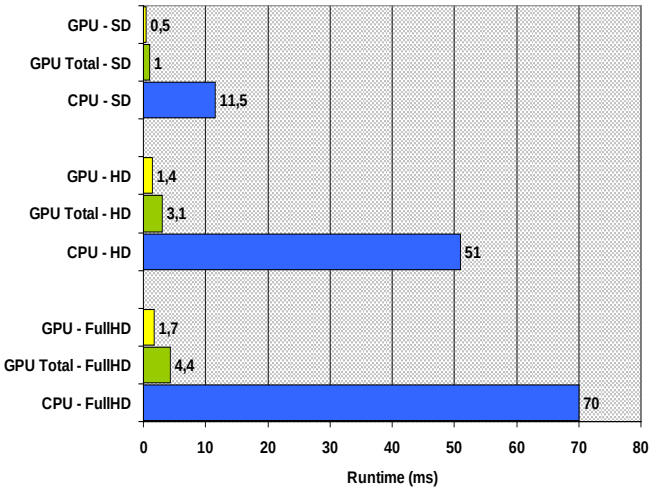

Figure 3: Runtime comparison of CPU and GPU implementation for different resolutions (720 x 576, 1440 x 1080, 1920 x 1080)

The experiments were done on a 3.0 GHz Intel Xeon Quad-Core machine with 4 GB RAM, equipped with a NVIDIA GeForce GTX 480 GPU on a PCIe 2.0 16x slot. The operating system of the machine was Windows XP 64, and CUDA Toolkit version 3.2 was used.

5.1 Runtime test

Runtime tests have been performed for videos with different resolutions, namely SD (720 x 576), HD (1440 x 1080) and Full HD (1920x1080) resolution. The results of these tests are given in Figure 4 (runtime per image/frame, in milliseconds). Here 'GPU' denotes the runtime for the core GPU algorithm, 'GPU Total' includes also the necessary transfer of the image from CPU to GPU and 'CPU' denotes the optimized multi-threaded CPU implementation.
As can be seen from figure 4, a significant speedup factor between 10 (SD) and 15 (Full HD) can be achieved for all examined image resolutions. When the transfer time is excluded and only the core CPU/GPU computation time is compared, the speedup factor is much higher, ranging from 20 (SD) to 40 (Full HD). In fact, the runtime of the GPU implementation is dominated by the transfer time of the input image (which is 8-bit, 3-channel) via the PCI-Express bus, which takes e.g. 2.7 milliseconds for Full HD resolution.
For Full HD video, the total runtime (including transfer) of the video breakup GPU implementation is approximately 4.4 milliseconds, therefore allowing us to analyze *nine* Full HD videos in real-time ($\leq$ 40 milliseconds) on a single Geforce GTX 480 GPU.

5.2 Quality test

The quality results show that our CUDA implementation of video breakup provides the same qualitative results (in terms of detection quality) as the corresponding CPU routine. It should be noted that NVIDIA GPUs of the Fermi architecture have a hardware implementation of arithmetic operations on float and double which conforms to the IEEE-754 standard. This is the same standard as is used by the floating point unit of CPUs, so it is assured that floating-point operations are done on the GPU in a similar way (but likely at a different order) as on the CPU.

6. AV-INSPECTOR APPLICATION

The proposed video breakup detector has been integrated into the AV-Inspector application for the semi-automatic video quality analysis and assessment recently proposed by [7]. It consists of two parts, for analysis and summarization & verification of the video impairments.

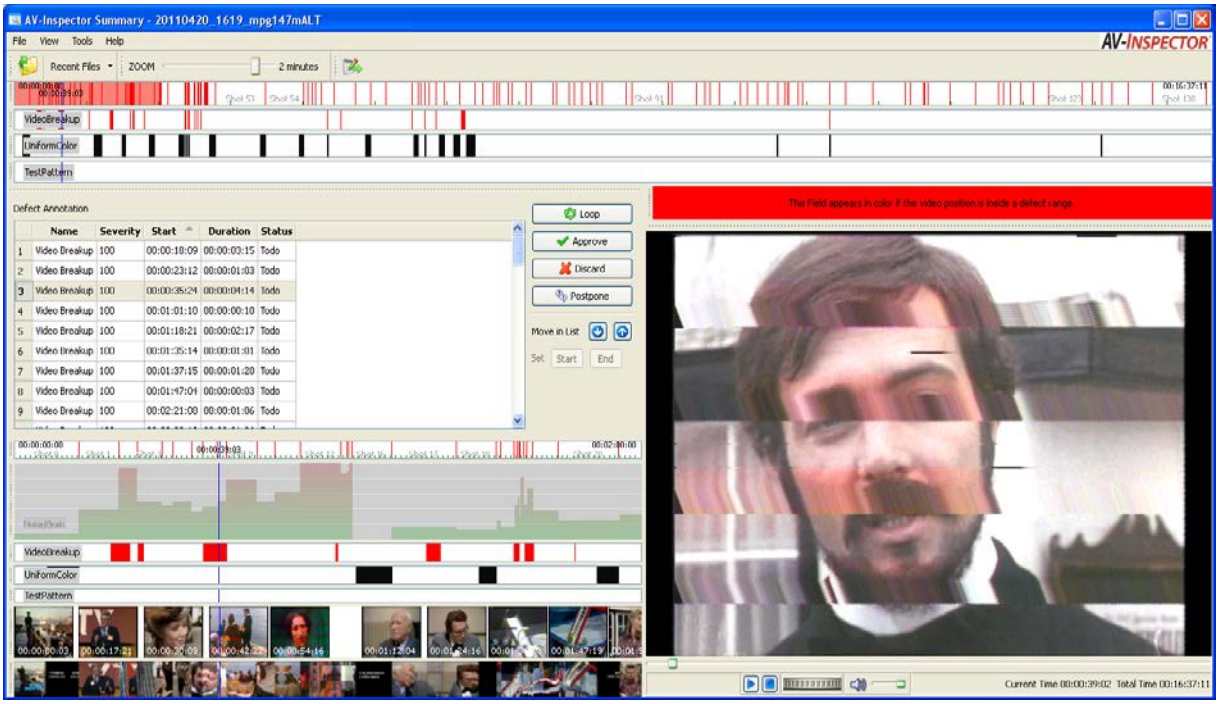

Figure 4: The AV-Inspector summary application, where various detect video defects and video quality measures (video breakup, uniform color forms, noise&grain magnitude etc.) are visualized on different timelines.

The analysis application runs the specified video quality algorithms (for detecting video breakup, test patterns or measuring impairment strength for noise level, flicker level etc.) for a batch of videos locally on a single PC, or distributed over a cluster of workstations. The output of the video quality algorithms is saved in a MPEG-7 conform way as described in [1].

The summarization & verification application (see Figure 4) then allows the user to quickly browse the video and inspect the detection results. For a convenient navigation through the content, the shot structure, automatically selected keyframes and a stripe-image representation are visualized in separate timeline views. Also other video quality measures (like noise level) that return an estimate of the amount of degradation are visualized on a timeline view as graph, whereas the output of detection algorithms like video breakup is marked on a timeline view and also reported as events in a defect annotation list. The human operator can easily inspect the defect annotation list and efficiently approve a detected event or discard events that have been detected by mistake.

7. CONCLUSION

In this paper, the efficient implementation of a previously proposed algorithm for the detection of severe image distortions termed *video breakup* for Fermi Architecture GPUs has been described. A comparison between the optimized video breakup CPU implementation and the GPU implementation shows a speedup ranging from 10 to 15, therefore allowing us to analyze nearly ten Full HD (1920 x 1080) video streams on a single GPU in realtime. Furthermore an application was presented, which integrates the presented video breakup detector, along other video quality related algorithms, and allows the operator to quickly get an assessment of the quality of video.

8. ACKNOWLEDGMENTS

The authors would like to thank Werner Haas, Georg Thallinger, Albert Hoffmann, Hermann Fürntratt and Marcin Rosner as well as several other colleagues at JOANNEUM RESEARCH and Martin Altmanninger and Paul Leitner from Media Services GmbH, who contributed valuable input to the work. Furthermore we give thanks to the Austrian Broadcasting Corporation (ORF), the British Broadcasting Corporation (BBC), RAI Italian Television and the Netherlands Institute for Sound and Vision (BnG) for providing requirement specifications as well as useful videos samples. This work has been funded partially under the 7th Framework Program of the European Union within the IST

project PrestoPRIME (IST FP7 231161). Furthermore, the work of Jakub Rosner was partially supported by the European Social Fund.

9. REFERENCES

[1] W. Bailer, P. Schallauer, The Detailed Audiovisual Profile: Enabling Interoperability between MPEG-7 Based Systems, Proceedings of 12th International Multi-Media Modeling Conference, Beijing, 2006

[2] H. Fassold, J. Rosner, P. Schallauer, W. Bailer, Realtime KLT Feature Point Tracking for High Definition Video, GravisMa workshop, Plzen, 2009

[3] P. Gwosdek, H., S. Grewenig, A. Bruhn, J. Weickert, A Highly Efficient GPU Implementation for Variational Optic Flow Based on the Euler-Lagrange Framework, Proceedings of the ECCV Workshop for Computer Vision with GPUs, 2010.

[4] S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, Optimization principles and application performance evaluation of a multithreaded GPU using CUDA, 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, 2008

[5] D. Shabtay, N. Raviv, Y. Moshe, Video packet loss concealment detection based on image content, 16th European Signal Processing Conference, 2008

[6] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, H. Bischof, Anisotropic Huber-L1 Optical Flow, Proceedings of the British Machine Vision Conference, London, UK, 2009

[7] M. Winter, P. Schallauer, A. Hoffmann, H. Fassold, Efficient video breakup detection and verification, 3rd international workshop on automated information extraction in media production (AIMPro), 2010

[8] D. Weken, M. Nachtegal, E. Kerre, Using similarity measures for histogram comparison, Proc. 10th International Fuzzy Systems Association World Congress, 2003