GPU Video Retargeting with Parallelized SeamCrop

Johannes Kiess, Daniel Gritzner, Benjamin Guthier
Stephan Kopf, Wolfgang Effelsberg
Department of Computer Science IV
University of Mannheim, Mannheim, Germany
{kiess|guthier|kopf|effelsberg}@informatik.uni-mannheim.de
gritzner@pi4.informatik.uni-mannheim.de

ABSTRACT

In this paper, we present a fast parallel algorithm for the retargeting of videos. It combines seam carving and cropping and is aimed for real-time adaptation of video streams. The basic idea is to first find an optimal cropping path over the whole sequence with the target size. Then, the borders are slightly extended to be reduced again by seam carving on a frame-by-frame basis. This allows the algorithm to get more important content into the cropping window as it is also able to remove pixels from within the window. In contrast to the previous SeamCrop algorithm, the presented technique is optimized for parallel processes and a CUDA GPU implementation. In comparison, the computation time of our GPU algorithm is 10.5 times faster (on a 960×540 video with a retarget factor of 25%) than the already efficient CPU implementation.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems; I.4.9 [**Image Processing and Computer Vision**]: Applications

General Terms

Algorithms

Keywords

GPU, SeamCrop, video retargeting, video resizing, seam carving, cropping

1. INTRODUCTION

A lot of videos are produced every day - feature films for cinemas, documental movies or smartphone videos. These videos are watched on a variety of different devices, ranging from widescreen TVs with Full HD resolution to smartphone displays with SVGA resolution. But not every video is suited for every display right away, they have to be adapted in size and resolution. This process is called retargeting.

There are different methods for video retargeting, for example warping [11] scales the frames non-uniformly or seam carving [14] removes paths of pixels from the frames. The computational complexity of most of these methods can get very high because a global optimization problem has to be solved over a whole video in order to generate the optimal retargeted version. The GPU can be used to significantly enhance the performance of such algorithms. For instance, the calculation of the importance functions can often be parallelized as the values are computed for each pixel independently from the others [19, 3]. The goal of these enhancements is to achieve real-time retargeting. This is especially important when the user wants to watch a live event like a football game or another type of livestream over the internet.

In this paper, we present an accelerated version of the SeamCrop approach for videos [6] using a CUDA implementation on the GPU. The basic idea of the algorithm is to find a cropping window of the target size and then extend the borders to get more important content into it through seam carving. As there are a lot of parts like the importance value calculation that can be parallelized, the algorithm is well suited to be implemented on the GPU.

Our main contributions are as follows:

- We present an efficient CUDA implementation of the SeamCrop algorithm for the GPU and explain the changes and adjustements in detail.
- A comprehensive performance test shows the efficiency of the new algorithm.
- The new GPU algorithm is 10.5 times faster (on a $960\times$ 540 video with a retarget factor of 25%) and is able to to retarget videos with a resolution up to 720×405 in real-time (assuming 25 frames per second).

The rest of this paper is structured as follows: other video retargeting algorithms using the GPU are presented in Section 2. In the following Section 3, the differences between the parallel CUDA version and the previous version of the algorithm are discussed and enhancements are described in detail. Section 4 shows the results of our performance test. Finally, Section 5 concludes the paper.

MMSys '14, March 19 - 21 2014, Singapore, Singapore
Copyright 2014 ACM 978-1-4503-2705-3/14/03...$15.00.
http://dx.doi.org/10.1145/2557642.2557648

2. RELATED WORK

A lot of research has already been done in the field of image and video retargeting. First, we will give an overview of media retargeting algorithms. In the next step, we will focus on the speeding up of the algorithms by using the GPU and discuss papers that also use the GPU in their implementation.

2.1 Media Retargeting Algorithms

Retargeting techniques can be roughly categorized into three different kinds of operators – cropping, seam carving and warping [9]. Basically all of these techniques follow a basic workflow. In the first of two steps, the importance of each pixel is determined. For instance, this can be done with gradient magnitude, saliency map, face detection, or a combination of several methods. In the second step, an operator or a combination of operators is used to adjust the size to the target size based on the information from step one. The main principle is always to preserve as much important content as possible. This general explanation defines the basic workflow. The execution can vary in the individual algorithms, for instance in a video that is retargeted frame-by-frame, the steps are repeated for each frame respectively.

Cropping tries to identify the most important region in an image or video and discards the rest around it. A so called cropping window is used with the goal to minimize its size while containing as much energy as possible. For videos, temporal consistency also plays an important role as a shaking cropping window is unpleasant to watch.
Such a cropping window can be found by an optimization of multi-size trajectories throughout a video scene [12]. Each trajectory represents a cropping window with a different size that may also cover different content. In terms of size, each trajectory has a fixed window size that does not change during a shot. Several of these trajectories are found by the algorithm and ranked based on the energy captured in them. The one with the highest energy is chosen for the shot. If there are various interesting regions in a scene that can not be contained in one trajectory, the algorithm is able to combine multiple of them.

Seam Carving removes paths of connected pixels from images and was first introduced in 2007 [1]. In contrast to cropping, this technique can discard content inside of an image, not just at the borders. A *seam* is defined as a connected path of pixels from top to bottom or from left to right with a low importance for the image content. The algorithm works iteratively and removes one seam in each step. In each iteration, the seam with the lowest energy is found and removed, effectively reducing the size of one dimension by 1.
A *vertical seam* is defined as a path of pixels from top to bottom with the following two constraints: The first constraints states that one and only one seam pixel is selected in each row. In the second constraint, the horizontal distance between two adjacent seam pixels is not allowed to be larger than 1. This leads to the seam pixels of vertical seams to be vertically or diagonally connected (8-connected). The cost of a seam is defined as the sum of the energy values of all pixels that are included in it. In order to find the optimal seam, dynamic programming is used to identify the seam with the minimal cost of all possible seams.
A *horizontal seam* is a connected path of pixels from left to right and is found in a similar manner by switching rows and columns of an image. When a seam is removed, all pixels are shifted vertically or horizontally to fill the resulting gap.
Seam carving was later expanded to also work for videos [14] by searching for 2D manifolds in the 3D video cube. Each manifold depicts a seam over the course of the whole video. As the originally used dynamic programming was not suitable for the optimization of a video cube, the authors represent the video as a graph and use graph cuts. Additionally, a new energy criterion called forward energy is proposed that takes into account the energy that is introduced by the new edges that occur by removing a seam. Nevertheless, if a video depicts objects with straight lines like buildings, seam carving typically introduces severe artifacts because straight lines may become curved or disconnected. In previous work, we improved seam carving by applying line detection and modifying the energy map in the local neighborhood of the intersection point of a seam and a straight line [8].
As the optimization over the 3D video cube takes a lot of processing time, another seam carving method for videos [10] was introduced that is more efficient by using a so-called background picture. This picture is created by aligning all the frames of the shot and using a median filter to merge the frame pixels. Seam carving with forward energy is then applied to find seams on the background picture. The found seams are finally mapped back to the individual frames by applying a camera motion compensation model between each frame and the background picture.
Each operator has its limitations, for instance, cropping can not be used for enlargement. *Multi-operator* retargeting tries to overcome these individual limitations by combining seam carving, scaling, and cropping [16]. The order of the operators is determined via dynamic programming in a multi-dimensional space where each dimension represents the use of an operator in either width or height. For videos, the order of the operators is found in the key frames and then interpolated for each intermediate frame.
Normally, a key aspect of seam carving is the constraint that the seams have to be connected. In a novel approach, seams are allowed to be temporally and spatially discontinuous [4]. The algorithm works on a frame-by-frame basis and uses an appearance-based temporal coherence measure to ensure consistency and let seams be temporally disconnected. In a similar manner, the seams can also be spatially disconnected.

Warping is the term that is used when an image or video is scaled non-uniformly in order to retarget it. A mesh is used to map the cells from the source to the target size. Based on the algorithm, the cells of this mesh can have different froms like quads or triangles and different cell sizes, going down to one pixel per cell. As with previous techniques, warping tries to maintain important content unchanged while distorting homogenuous regions that the viewer will most likely not notice.
An interesting approach considers the optical flow as an additional information [17]. The retargeting process is divided into a spatial and a temporal component that can be computed one after the other. First, the algorithm uses warping on each frame seperately. Based on the change in the frame, the motion pathlines of the pixels can be determined through the optical flow and be optimized in comparision of the ones from the original frames. This additional information is then

used in a second run through of the warping on the frames. A more object-based approach uses segmentation to preserve objects especially during the retargeting [13]. The first step of the algorithm is to use a spatio-temporal segmentation in order to find volumetric objects in the 3D video cube. These segmented objects are not objects like a person or a car, but connected regions of a similar color. Important volumetric objects are then kept as unchanged as possible while the other regions are adapted similarily to linear scaling in the retargeting step.

2.2 GPU Importance Detection

Importance functions are a central part of each retargeting algorithm, as the resulting importance map guides the following steps and has a crucial impact in the achieved quality of the result. These functions often have low dependencies between their seperate steps or the information needed depends on a small pixel area which makes them a good candiate for parallelization.

For instance, when computing an importance map for a 640×480 image, Xu et al. [19] are 8.5 times faster using a GPU implementation (with four graphic cards) compared to the CPU implementation. They use the saliency map from Itti et al. [5] as a basis and implement it using the CUDA technology from NVIDIA. Goferman et al. [3] also use CUDA on the GPU to enhance the performance of their importance function, but use a different approach than [19]. In contrast, they try to find the regions that define the content of the image rather than the possible fixation points of a viewer. Their GPU version is five times faster than the corresponding CPU implementation.

2.3 GPU Video Retargeting

Previous work has shown that GPU implementations are also capable of speeding up the retargeting of images and videos. A prominent example is the warping algorithm for streaming videos presented in [11]. It works on the pixel level and has the ability to retarget videos in real-time. This is possible due to high parallelization and an implementation on the GPU. Also, the authors combine automatically detected features with manual annotations to enhance the accuracy of their importance map. They achieve 33.5 frames per second (fps) on a 480×270 sequence with a retarget factor of 50%. Another interesting approach combines warping with cropping [18]. Critical regions are defined with content that should be preserved while everything outside the borders is allowed to be cropped. The content inside the borders is then retargeted through warping. An interesting proposition is to give up the notion that all important content has to be preserved at all times during the course of a video. The GPU is used to calculate the deformation of the mesh during the warping. When using a retarget factor of 50%, the algorithm can retarget a 688×288 sequence with 6 fps.

Seam carving [14] is the basis of a highly parallelizable video retargeting technique [2] for the GPU. In the retargeting process, the video is computed frame by frame, which makes the technique also suitable for images. A frame is enhanced with a newly introduced just-noticeable-distortion (JND) model and then an importance map is computed by applying an edge detection filter. JND is composed of several image features like gradient, entropy, visual saliency, segmentation, etc. Also, a multi-seam search scheme is presented which can find multiple seams without the need to recompute the importance map after each seam by backtracking and splitting found seam paths. The seams are searched sequentially, frame by frame. For temporal coherence, the seams need to be connected which leads to them forming a continuous surface in the 3D video cube. Performance-wise, the algorithm achieves 11 fps on a 512×384 sequence (retarget factor 50%).

3. PARALLEL SEAMCROP

In this section, we start by exlpaining SeamCrop for videos as it was proposed in [6, 7]. Then, we describe how this algorithm can be parallelized and present details of our CUDA implementaion for the GPU.

3.1 SeamCrop for Videos

The main idea of SeamCrop is to use seam carving 'carefully' to not introduce visible artifacts. In the following explanations, a $m \times n$ video cube should be retargeted to $m' \times n$, which is a reduction of width. A change of height is done similarily while a reduction of both dimensions is processed sequentially one after the other.

In contrast to image retargeting, the algorithm starts by searching for an optimal cropping window path with the target size over the course of the whole video. To measure the importance of each pixel, motion saliency and the gradient magnitude are used. First, the importance values in each column $i = 1, ..., m$ are summed up to a column cost value respectively. Based on these values, the energy costs of all possible cropping windows in each frame can easily be optained by summing up the values of the included columns. This leads to a two-dimensional array where each value depicts the position i of a cropping window in a frame t. Dynamic programming is then used to find the path with the maximum energy in a similar way to the search of a seam in seam carving [14]. As the optimal path can be shaky due to small movements to the left and right by one pixel, a Gaussian filter is used to smooth the result.

When the path of the cropping window has been found, the seam carving step of the algorithm begins. In contrast to the cropping step, the search for seams is done frame by frame. Therefore, the borders of the window are equally extended on both sides. In the experiments in [6], the window was enlarged by 20%. If one side of the window is close to a border, it is extended until the border and the rest is added on the other side. Seam carving is then used inside the extended borders to get more important content into the window or to prevent objects from being cut off (see Figure 1).

In order to ensure temporal coherence, energy costs similar to [4] are used.These costs take the position of the seams from the previous frame $t-1$ into account. Assuming that s_k^{t-1} seams were found, the same amount is searched in the current frame. The energy costs are called temporal coherence costs and have to be calculated for each seam s_k^t seperately. They are zero at the positions of the corresponding seam from the previous frame and increase linearly in the horizontal directions from each seam point until a threshold β is reached. This threshold regulates how much the position of the seam from the previous frame is forced in the current frame. All positions after the threshold are set to the same value.

Due to a possible moving cropping window it may happen that positions of a seam from a frame before lie now outside of the extended borders. If this happens for a position,

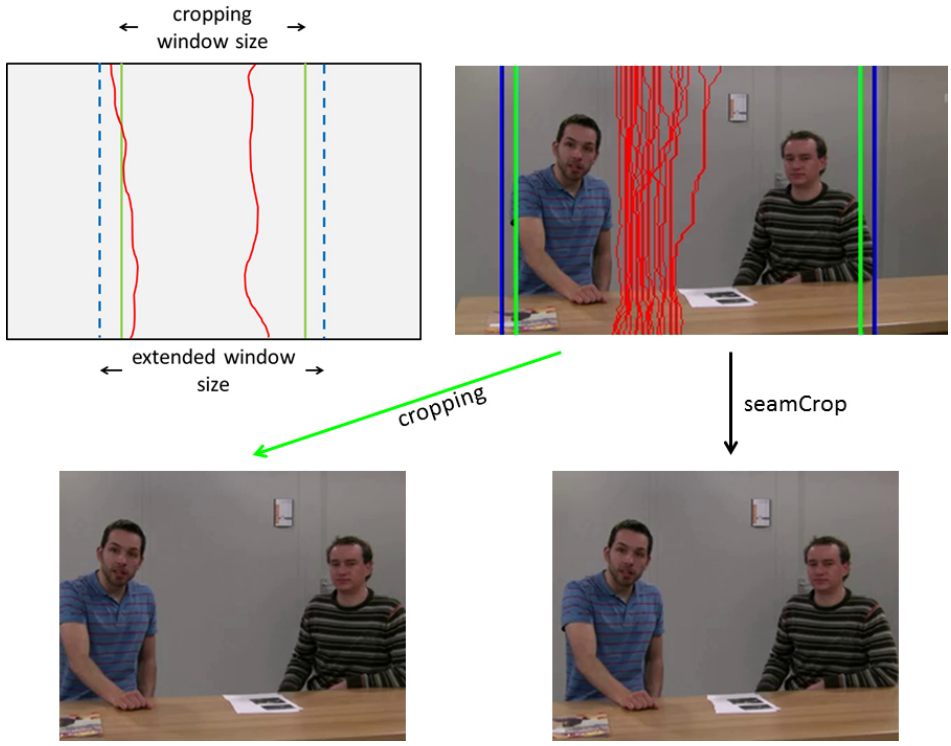


Figure 1: Top row: The estimated cropping window (green lines) is extended (blue dotted lines) and then reduced to the target size again by removing a small number of seams (red). Bottom row: results of the frames being reduced by the cropping operator alone (left) and by SeamCrop (right). *From Kiess et al. [6].*

every cost value in that row is set to the threshold value. When more than 20% of the positions are outside, no temporal costs are added for the seam so that a new seam can be found.

3.2 SeamCrop for GPU

An efficient version of the SeamCrop algorithm is already available[6], using a 2D optimization for the search of the optimal cropping window and doing seam carving on a frame-by-frame basis. To further enhance the processing speed, some changes are made to parallelize more processes and optimize the performance of a GPU implementation. These changes and some insights of our CUDA implementation are presented in detail in the following. If not stated otherwise, assume all steps to be executed on the GPU in order to save copy operations between CPU and GPU.

3.2.1 System Overview

The basic workflow of the algorithm remains the same: two passes are done on the video, the first to find a cropping window over the whole sequence and the second to search for seams frame by frame (see Figure 2).
In both passes, frames are treated in multiple CPU-threads. These threads are responsible for loading the frames to the CPU memory, copying them from CPU to GPU and removing them from the GPU when they are not needed anymore. Also, they organize the workflow on the GPU as they start kernels for the frames and watch the synchronization between the frames, for example that a frame has to wait in the seam carving step until enough seams are found in the previous frame because of the temporal coherence costs. By using multiple CPU-threads and assigning a different CUDA stream to each thread multiple kernels can be executed on the GPU in parallel. This ensures that the GPU utilization is high throughout the runtime.
On the GPU, frames are usually divided into regions that are computed independent and parallel, so called blocks. Each block consists of several threads that have a shared memory. As CUDA has no mechanism for synchronizing blocks, we implemented one ourselves. This is important because most of the steps of the algorithm need that the steps before are finished, for instance, when finding the optimal seam paths the result of each row depends on the previous row. The block synchronization function uses CUDA's thread synchronization to ensure that all threads of a given block enter and leave the block synchronization function at the same time. The first thread of each block is chosen as a representative for the block. This representative uses atomic functions to increase a variable and perform a busy wait on said variable. Once all blocks have reached this point the representatives will end their busy wait and normal operation resumes. As an alternative, kernel launches could have been used for synchronization; but doing so decreased the performance of the program.

3.2.2 Importance Function

As the pixels within a frame are independent from each other during the importance calculation, this step of the algorithm is highly parallelizable. There are several possibilities to determine the importance values like saliency map or histogram of gradients that can be used in this approach. We chose to stay with the simple gradient function from [6], as

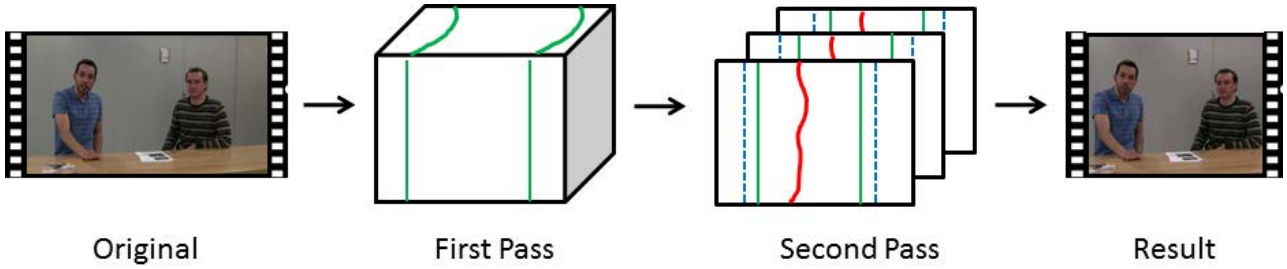


Figure 2: Overview of the basic workflow of the algorithm. In pass one, the optimal cropping window is searched via dynamic programming (green lines) in a global optimization over all frames. Seams are searched in the second pass (red lines) inside the extended borders of the cropping window (blue dotted lines) frame-by-frame.

it provides sufficient information for visually good results.
Also, we keep the simple motion saliency function. As this function is calculated dependent on the previous and the succeeding frame respectively, the thread of the frame has to wait until these other two frames are also loaded in the memory. The resulting motion saliency map is smoothed with a Gaussian function. We implemented a horizontal smoothing kernel and transpose it for the vertical direction. This way, only one implementation is necessary and there are no conflicts with the memory banks of the GPU that would occur in an additional vertical kernel.
After the importance map is calculated, the frame is dropped and the thread can be assigned to another frame. A major difference implementation-wise is the drop of the importance information in the first pass after the necessary data for the calculation of the cropping window has been gained. This is due of the possibility of parallelizing this step efficently so that it is fast to recalculate the importance function. This leads to the advantage of being able to process videos with higher resolutions, as the algorithm needs less memory space than before when it kept all frames and importance maps in the memory.

3.2.3 *Cropping Window*

In the calculation of the cropping window path, there are some dependencies that have to be considered in the parallelization. For instance, the columns have to be summed up before the costs of each cropping window position can be calculated. The columns themselves are independent and all the sums can be calculated at the same time.
In the next step, the cropping path is searched via dynamic programming in a 2D array similar to seam carving [14]. This leads to a dependency between the rows, as each position in a row depends on the values of its potential predecessors in the line above (see Figure 3). Therefore, rows are done one after another while the positions in a row can be calculated inependently. When the costs of the path have been summed up, the optimal cropping path can be found by backtracking from the cheapest value in the last row, which is not very complex. Everything except the calculated cropping path is then dropped from the memory before the second pass starts. The smoothing of the path is done on the CPU as it is not a complex operation. It is the only operation in our implementation that is not computed on the GPU.

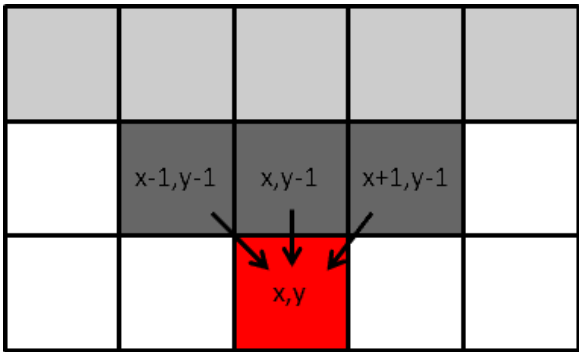


Figure 3: Example for the pixels depending on each other during the search for a seam. The cost of the current pixel (red) is the addition of its importance value with the cost of the cheapest predecessor in the line above (dark grey). Only the three adjacent pixels are possible. These themselves depend on the values of the line above (light grey).

3.2.4 *Seam Carving*

Seam Carving is done in the second pass of the algorithm. As mentioned before, the seams are now computed frame-by-frame instead of a global optimization like in the cropping step. Each seam depends on the seams that have been previously calculated in the current frame t as well as the corresponding seam from the frame $t-1$. This means that only one seam is searched in each frame at a time, although this is done in parallel. Because the algorithm uses information of the seams from the previous frame $t-1$ via temporal coherence costs, each thread waits until the thread of frame $t-1$ has found enough seams. For instance, if seam i should be calculated, the thread waits until the other one has found at least seam $i+1$ before it starts with it. As a lookup table, an array keeps track of the number of seams that have been found in all frames. If a thread has to wait because not enough seams are found in frame $t-1$, the time is given to other threads until it can continue.
Like mentioned before, each row is computed one after another in the dynamic programming step where the cheapest seam is searched (see Figure 3). For the calculation, each column is assigned its own thread so that each pixel in a row can be computed independently. Because the rows depend on each other, each one has to wait until the previous one

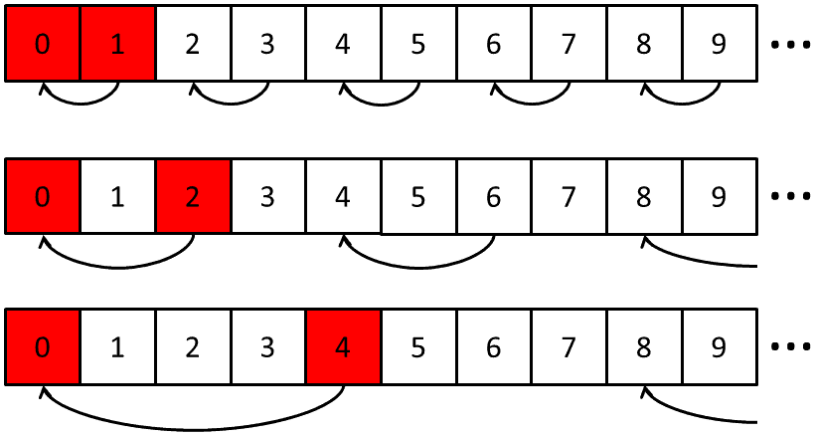


Figure 4: Search for the minimum value in the last row of a frame. Top row of the figure: each second thread checks if its neighbor has a lower value and copies it and its position in this case (i.e., 0 compares to 1 (red), 2 to 3,..). Middle row: the same is done with each fourth thread and the thread two IDs before (i.e., 0 compares to 2 (red), 4 to 6, ...). This is repeated until the first thread has the optimal value of the block and its position.

is finished. After this computation, the cheapest value in the last row has to be found because it marks the end of the optimal seam. This is done differently than in the original approach, where the minimum is found by just going through from left to right.

Instead all threads of a block are used to find the minimum. First, each thread copies its importance value and its position into the shared memory. Then, every other thread checks if its neighbor has a better value and copies it and the position in that case (see Figure 4). After that, every fourth thread checks if the thread two IDs after it has a better value. This goes on until the optimal value and its position have been copied all the way to the first thread of each block. Lastly, each block writes its best value and position into the global memory and the first thread of the first block determines the global minimum and more importantly the position of the global minimum.

In order to save computation time, the importance map is only calculated once and then modified by each found seam in a way that the following seams will not pick the same pixels. This is done by setting the importance values of the used pixels really high.

4. DISCUSSION

We conducted a detailed performance test in order to prove the efficiency of the GPU implementation. In the performance test, we compared GPU against CPU, measured how long pass one and two of the algorithm take and tested different parameters. As we did not fundamentally change any of the core principles of the SeamCrop algorithm [6], we pass on a quality of results comparision to other state-of-the-art video retargeting algorithms.

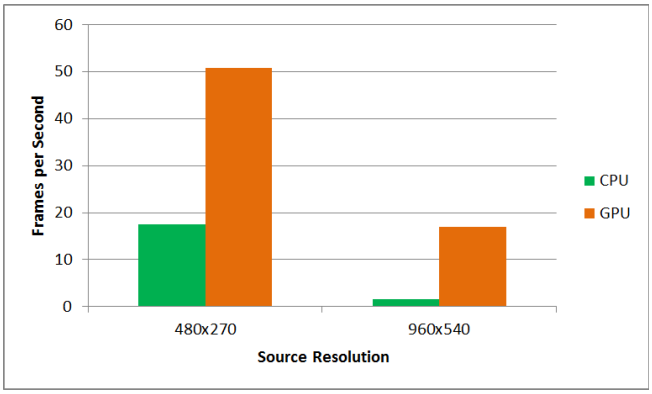
All the tests were performed on a PC with the following specifications: Intel i7-3770 processor with four cores at 3.4 GHz, 16 GB DDR3 RAM and a NVIDIA GeForce GTX 650 TI with 1024 MB memory and 768 CUDA cores. In the implementation, eight CPU threads are used as the PC has four cores and is able to support two threads per core. The factor for the extended borders is set to 20% and only the width is reduced in all tests while other parameters vary. Also, except for GPU test number four, all sequences have 183 frames for a better comparision of the results.

We first did a performance comparison between the already efficient CPU implementation and our new GPU version of the algorithm (see Figure 5). The tests where done on a 480×270 and a 960×540 sequence with the retarget factors 25% and 50%. We take those factors from a comparative study on image retargeting by Rubinstein et al. [15], where they are regarded as a considerable resizing. In the 480×270 sequence, the GPU version is about 3 times as fast as the CPU one. The factor gets higher for the 960×540 sequence as the parallel processes come more into effect. There, the new algorithm is 10.5 times faster for a retarget factor of 25% and 8 times faster for 50%.

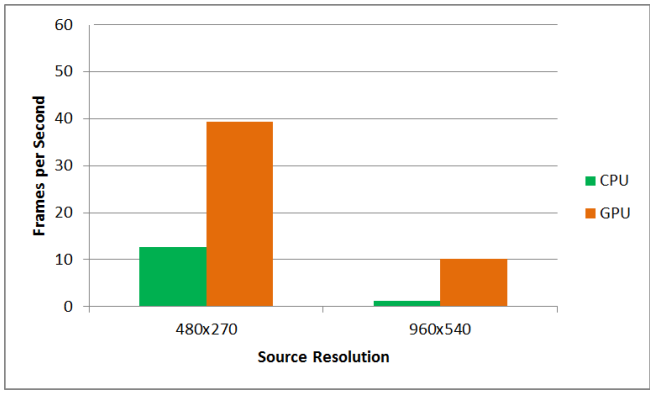
In addition to the comparison to the CPU version, we also did detailed performance tests on the GPU implementation. There are four tests with varying parameters that are presented in the following.

In our first GPU test case, we measured how long the algorithm takes to reduce the size of a video by 25% and by 50% (see Figure 6). For the reduction, six sequences are used in four different resolutions (1920×1080, 1440×810, 960×540 and 480×270). For the 480×270 sequence, the gap when reducing the retargeting factor from 25% to 50% is not big. This is caused by the computation overhead of steps that have to be done regardless of the target size. The gap gets even smaller with increasing resolutions as the capacity of the GPU is fully used and the frames per second drop in general. When analyzing the percentage load of each pass, there are clear differences (see Figure 7). The first pass is less complex and has more potential to be parallelized than the second pass. Especially the seam carving in pass two with its need to synchronize row after row takes time. Nonetheless, the GPU version is a lot faster than the CPU version.

GPU test case number two uses a 960×540 sequence and varies the retarget factor width from 10% to 50% (see Figure 8). The decrease in fps over the increasing retarget factor has to do with more possible cropping window positions per frame as well as a larger number of seams that has to be found. It is not a linear progression because the use of the



(a) Retarget Factor 25%



(b) Retarget Factor 50%

Figure 5: Performance comparison between the CPU and the GPU version of the algorithm.

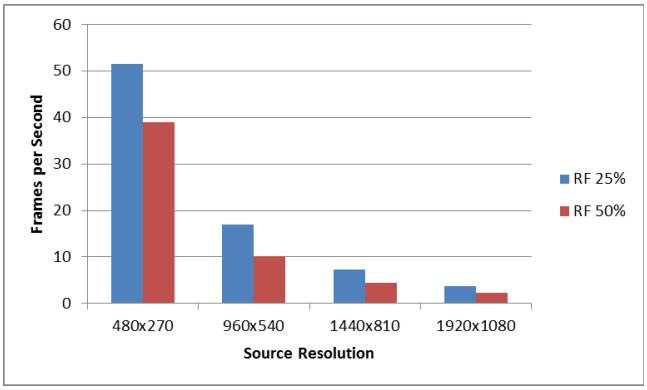


Figure 6: GPU test number one: Frames per second on four different resolutions and two retargeting factors (RF). For the figure, the results from the six used sequences are averaged.

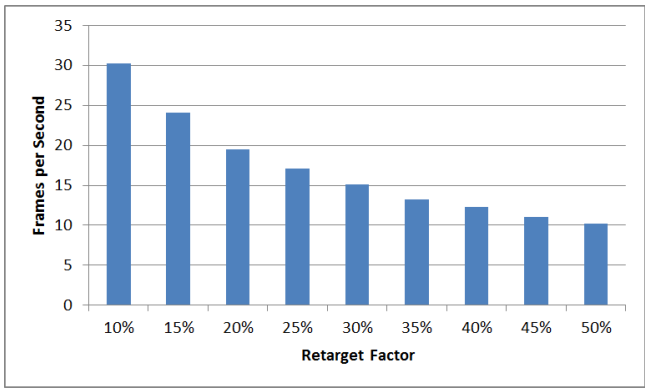


Figure 8: GPU test number two: Frames per second of a 960×540 sequence with a varying retarget factor.

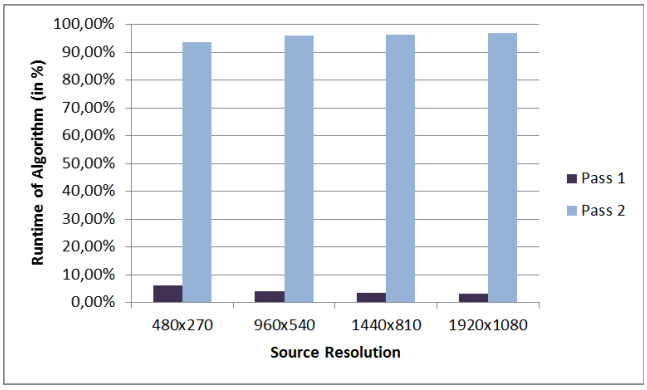


Figure 7: Percentage that the two passes take in relation to the pure processing time (without loading and saving the frames) on the GPU. For this figure, the results for both retarget factors are averaged as the percentages are nearly identical.

GPU becomes more efficient in the later stages as there are more computations that can be done in parallel.

In test case number three, we retarget a sequence by 25% but vary the resolution of the source video (240×135, 480×270, 720×405, 960×540, 1200×675, 1440×810, 1680×945, and 1920×1080). Like in the previous test, the progression is also non-linear (see Figure 9). This has to do with two factors: First, not all steps in the figure have the same amount of pixel increase. For instance, 480×270 has four times the pixels than 240×135, but 720×405 has only 2.25 times the pixels than 480×270. Second, the GPU is not working to full capacity in the beginning with the low resolutions and is later more efficient in parallelizing at the high ones. Assuming 25 frames per second and always keeping a shot in the buffer, our algorithm is able to achieve real-time retargeting up until a resolution of 720×405 pixels.

Lastly, in our fourth test case, a 960×540 sequence is retargeted by 25% with a varying number of frames (183, 244, 305, 366, 427 and 488). As expected, the fps stays at the same amount (in this case seventeen) no matter how many frames the sequence has (see Figure 10).

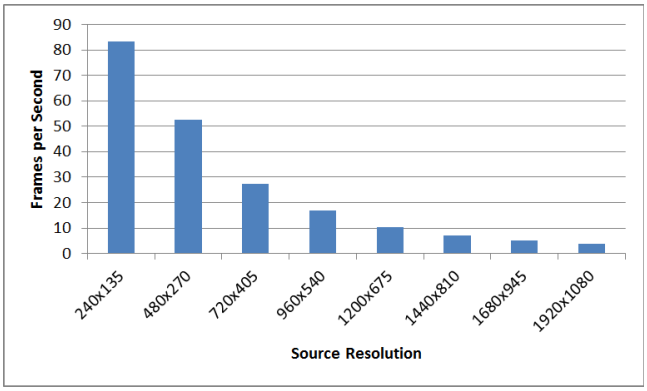


Figure 9: GPU test number three: Frames per second with increasing frame resolutions and a retarget factor of 25%.

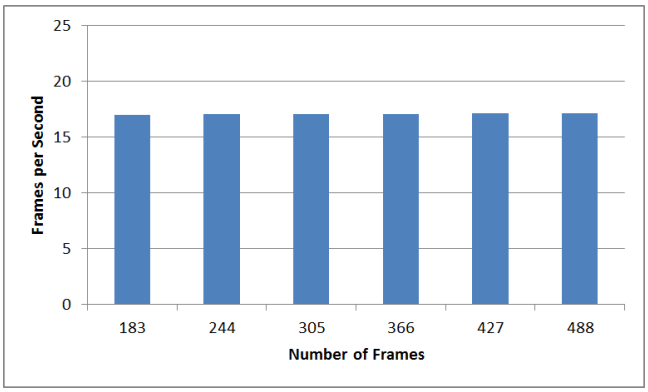


Figure 10: GPU test number four: The number of frames of the video sequence (960×540, retarget factor 25%) does not affect the frames per second that are achieved.

5. CONCLUSION

We presented a GPU implementation of the SeamCrop algorithm for videos with a significantly enhanced performance compared to the original. The differences and the adjustements between the two versions are thoroughly discussed and measurements of the efficiency are shown in a detailed performance test. In comparison to the already efficient CPU implementation of the original paper, the computation time of our algorithm is 10.5 times faster (on a 960×540 video with a retarget factor of 25%). Also, our algorithm is able to retarget videos with a resolution up to 720×405 pixels in real-time (assuming 25 frames per second).
In future work, we would like to extend our GPU algorithm to stereoscopic videos. However, this is not an easy and straightforward transition as new important factors like the disparity between the two views have to be considered.

6. REFERENCES

[1] S. Avidan and A. Shamir. Seam carving for content-aware image resizing. *ACM Transactions on Graphics, SIGGRAPH 2007*, 26(3), 2007.

[2] C.-K. Chiang, S.-F. Wang, Y.-L. Chen, and S.-H. Lai. Fast jnd-based video carving with gpu acceleration for real-time video retargeting. *IEEE Trans. Cir. and Sys. for Video Technol.*, 19(11):1588–1597, 2009.

[3] S. Goferman, L. Zelnik-Manor, and A. Tal. Context-aware saliency detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(10):1915–1926, 2012.

[4] M. Grundmann, V. Kwatra, M. Han, and I. Essa. Efficient hierarchical graph-based video segmentation. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2141–2148, 2010.

[5] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 20(11), pages 1254–1259. IEEE Computer Society Press, November 1998.

[6] J. Kiess, B. Guthier, S. Kopf, and W. Effelsberg. SeamCrop: Changing the size and aspect ratio of videos. In *Proceedings of the 4th Workshop on Mobile Video*, MoVid '12, pages 13–18, New York, NY, USA, 2012. ACM.

[7] J. Kiess, B. Guthier, S. Kopf, and W. Effelsberg. SeamCrop for image retargeting. In *Proceedings of IS&T/SPIE Electronic Imaging (EI) on Multimedia on Mobile Devices*, volume 8304. SPIE, 2012.

[8] J. Kiess, S. Kopf, B. Guthier, and W. Effelsberg. Seam carving with improved edge preservation. In *Proceedings of IS&T/SPIE Electronic Imaging (EI) on Multimedia on Mobile Devices*, volume 7542, pages 75420G:01 – 75420G:11, January 2010.

[9] S. Kopf, T. Haenselmann, J. Kiess, B. Guthier, and W. Effelsberg. Algorithms for video retargeting. *Multimedia Tools and Applications (MTAP), Special Issue: Hot Research Topics in Multimedia, Springer Netherlands*, 51:819–861, January 2011.

[10] S. Kopf, J. Kiess, H. Lemelson, and W. Effelsberg. FSCAV: Fast seam carving for size adaptation of videos. In *Proceedings of the 17th ACM International Conference on Multimedia (MM)*, pages 321–330, New York, NY, USA, 2009. ACM.

[11] P. Krähenbühl, M. Lang, A. Hornung, and M. Gross. A system for retargeting of streaming video. In *ACM SIGGRAPH Asia 2009 papers*, pages 1–10, New York, NY, USA, 2009. ACM.

[12] Y. Li, Y. Tian, J. Yang, L.-Y. Duan, and W. Gao. Video retargeting with multi-scale trajectory optimization. In *MIR '10: Proceedings of the international conference on Multimedia information retrieval*, pages 45–54, New York, NY, USA, 2010. ACM.

[13] S. Lin, C. Lin, I. Yeh, S. Chang, C. Yeh, and T. Lee. Content-aware video retargeting using object-preserving warping. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2013.

[14] M. Rubinstein, S. Avidan, and A. Shamir. Improved seam carving for video retargeting. *ACM Transactions on Graphics, SIGGRAPH 2008*, 27(3), 2008.

[15] M. Rubinstein, D. Gutierrez, O. Sorkine, and A. Shamir. A comparative study of image retargeting. In *ACM SIGGRAPH Asia 2010 papers*, pages

160:1–160:10, New York, NY, USA, 2010. ACM.

[16] M. Rubinstein, A. Shamir, and S. Avidan. Multi-operator media retargeting. *ACM Transactions on Graphics, SIGGRAPH 2009*, 28(3):1–11, 2009.

[17] Y.-S. Wang, J.-H. Hsiao, O. Sorkine, and T.-Y. Lee. Scalable and coherent video resizing with per-frame optimization. *ACM Transactions on Graphics, SIGGRAPH 2011*, 30(4), 2011.

[18] Y.-S. Wang, H.-C. Lin, O. Sorkine, and T.-Y. Lee. Motion-based video retargeting with optimized crop-and-warp. *ACM Transactions on Graphics, SIGGRAPH 2010*, 29(4):article no. 90, 2010.

[19] T. Xu, T. Pototschnig, K. Kuhnlenz, and M. Buss. A high-speed multi-gpu implementation of bottom-up attention using cuda. In *IEEE International Conference on Robotics and Automation, 2009. ICRA '09.*, pages 41–47, 2009.