High Performance Graphics (2012)
C. Dachsbacher, J. Munkberg, and J. Pantaleoni (Editors)

Algorithm and VLSI Architecture for Real-Time 1080p60 Video Retargeting

Pierre Greisen[1,2], Manuel Lang[1,2], Simon Heinzle[2], Aljosa Smolic[2]

[1]ETH Zurich, Switzerland
[2]Disney Research Zurich, Switzerland

Abstract

Aspect ratio retargeting for streaming video has actively been researched in the past years. While the mobile market with its huge diversity of screen formats is one of the most promising application areas, no existing algorithm is efficient enough to be embedded in such devices. In this work, we devise an efficient video retargeting algorithm by following an algorithm-architecture co-design approach and we present the first FPGA implementation that is able to retarget full HD 1080p video at up to 60 frames per second. We furthermore show that our algorithm can be implemented on embedded processors at interactive framerates. Our hardware architecture only requires a modest amount of hardware resources, and is portable to a dedicated ASIC for the use in consumer electronic devices such as displays or mobile phones.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Content-aware Video Processing—Non-linear video deformation

1. Introduction

Today's display landscape is composed of an abundance of different aspect ratios, whereas video content is typically produced for a specific target aspect ratio. While TV and cinema formats are fairly standardized, the widespread adoption of hand-held smart-phones and tablet PCs leads to huge aspect ratio differences compared to the original content. In order to cope with the different output formats, traditional methods such as linear scaling or letter-boxing (adding black bars) are often employed to retarget video content. However, recent user studies [STR*05, KSVC07, RGSS10] have shown that linear downscaling leads to perceptually unpleasant viewing experiences due to the overall image distortion. Letterboxing on the other hand does not distort the image content, but reduces the usable display area which can be problematic especially for the small displays used in hand-held devices.

In order to alleviate this problem, many feature-preserving non-linear rescaling methods have been proposed [WGCO07, RSA08, YSWL08, KLHG09]. Instead of globally distorting the image, these rescaling methods locally hide the distortions in less important regions, whereas the aspect ratio of visually important features is preserved. A recent user study [RGSS10] compared various of these state-of-the-art retargeting approaches to linear scaling and image cropping, and concluded that linear scaling was always perceived worst. Even more interestingly, some of the advanced non-linear algorithms [KLHG09, RSA09] were often perceived superior to cropping, even though non-linear retargeting algorithms almost always introduce distortion artifacts.

So far, no computationally efficient solution for video retargeting in end-user devices has been presented. We therefore present a retargeting algorithm based on a modified version of [KLHG09] to achieve quality results at much lower computational complexity. Based on the new algorithm, we describe an efficient hardware architecture for mobile devices and validate this architecture using a prototype FPGA implementation. We show that our implementation is able to retarget HD input videos in real-time, at modest hardware cost and at low clock speeds. We furthermore present a comparison to a mobile general-purpose embedded processing unit.

Contributions. To summarize, the contributions of this paper are as follows. First, we present a hardware-efficient, real-time video retargeting algorithm. Many algorithmic design decisions where influenced by co-designing for a corresponding hardware architecture. We compare the results of our algorithm to previous work, and show that the quality is

DOI: 10.2312/EGGH/HPG12/057-066

similar to existing approaches, however, at much lower computational complexity. Second, we present the first dedicated hardware implementation of video retargeting. The FPGA implementation is highly efficient in terms of resource usage and achieves 1080p60 retargeting performance.

2. Related Work

Image retargeting has been extensively researched in recent years, and Rubinstein et al. [RGSS10] present an excellent comparative study on a broad body of previous work. In this section, we will focus on video retargeting.

Video Cube Based Algorithms. Video retargeting algorithms can be separated into two classes. The first class of algorithms operates on a video cube, i.e, a large number of frames simultaneously, which needs to be processed offline on high-end computing platforms. While algorithms such as [WFS*09, WLSL10, YSWL11] achieve high-quality retargeting results, they usually require huge computational resources as well as a huge amount of memory and memory bandwidth. The algorithms cannot be applied to streaming video, and are therefore not suited for real-time retargeting on resource-limited devices.

Streaming Video Based Algorithms. The second class of algorithms are streaming video based approaches that do not operate on 3D video cubes directly, but rather on individual frames with inter-frame consistency constraints. One drawback of such approaches are temporal artifacts due to the limited amount of available future video frames. Our work is based on the streaming video retargeting system of Kraehenbuehl et al. [KLHG09]. In their system, the input video is analyzed to detect spatio-temporal saliency and other visually important cues. The analysis results are then used to formulate a global energy minimization to find the optimal retargeting transformation grid. The global energy minimization furthermore includes a temporal consistence constraint, that smoothes the deformation grid using the previous deformation. Unfortunately, the energy minimization problem is very intense in terms of computations and memory bandwidth, which is not suited for embedded devices. The authors report performance of up to 10 frames per seconds for 720p video, however using a high-end CPU/GPU computing system which has several orders of magnitude more computational resources and memory bandwidth than small FPGAs/ASICs and embedded processors.

The approach from [SWDL09] computes the visual importance map offline and calculates the resampling grid on the target device. Along similar lines, [ZHM08] precomputes a shrinkability map, which can directly be applied on the target device without much overhead. While both approaches relieve the target device of the retargeting computation, both will only work in the presence of precomputed meta-data. The recent work by Kim et al. [KJJK11] uses columns that are adaptively grouped in stripes, which

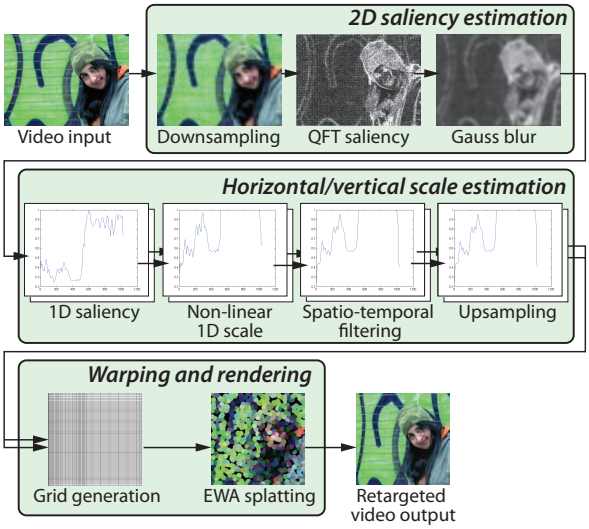

Figure 1: *Overview of the video retargeting algorithm. Original image from [RGSS10].*

are then scaled independently in the horizontal direction. However, their approach is computationally intensive, and not suited for hardware implementations. In summary, none of the current streaming video approaches can achieve real-time retargeting of HD video.

Hardware Architectures. Bae et al. [BCP*11] present an FPGA implementation to determine visual saliency for gaze prediction. However, their architecture only supports 4M pixels/second and cannot be extended to high-definition video easily. Greisen et al. [GSH*12] present a general-purpose video rendering architecture based on elliptical-weighted-average splatting. We adapt a simplified version of the rendering architecture as one component in this work. To the best of our knowledge, no hardware architecture for video retargeting has been reported yet.

3. Video Retargeting Algorithm

Our algorithm consists of three steps. As a first step, a visual saliency map is computed. In a second step, two resampling patterns both in horizontal and vertical direction are computed based on the saliency information. As a result, areas of low importance are scaled non-uniformly whereas the aspect ratio of more important areas is kept uniform. In a last step, the video stream is warped according to the resampling pattern, and finally streamed out. To achieve temporal stability, we filter the resampling patterns over time. Figure 1 summarizes the overall algorithm.

Our algorithm combines and modifies techniques from previous works. At its core, we separate the 2D warp computation of [KLHG09, PWS12] into two separate 1D warp

problems. Similar to [PWS12], this separation leads to a rectilinear retargeting mesh, as opposed to arbitrary transformations [KLHG09].

3.1. 2D Saliency Estimation

Similar to previous work, we compute a low-level saliency map to determine the visually important parts of the video. As a first step, the input images are down-sampled before computing the actual saliency. While the saliency could be computed at higher input resolutions, this usually does not add substantial information (see [JDT10]).

Our estimation algorithm is based on [GMZ08] and shows a good trade-off between computational complexity and estimation quality. The original algorithm relies on the observation that visual saliency can be obtained from the phase spectrum of the video sequence: the phase carries information on where discernible objects are located in an image [GW08] The phase spectrum of a video sequence (RGB with motion, RGBm) is extracted using the quaternion Fourier transform (QFT). First, the RGBm information is transformed to a quaternion representation (complex number with four components), and then transformed to frequency space using a QFT. In the frequency domain, the phase information is extracted by normalizing each quaternion with its magnitude. An inverse QFT transforms the phase information back into the image domain. Note that the QFT can efficiently be calculated by two separate 2D fast Fourier transforms (FFTs).

We compute the 2D FFT by first performing a 1D FFT on the rows, followed by a 1D FFT on the columns. Unfortunately, this operation corresponds to a matrix transposition, which can be costly in terms of memory bandwidth – especially if the 2D array does not fit entirely in the cache [CS00]. Therefore, we modify the original algorithm to compute a block-wise saliency. More specifically, we partition the input image into N sub-images. Each of the sub-images spans the full width of the downsampled input image, but at only $\frac{1}{N}$ of the vertical resolution. Then, the saliency computation can be performed on all sub-images individually, avoiding the need to transpose the full image data. The factor N can be chosen depending on the on-chip cache size of the target architecture.

The naive block-wise saliency decomposition leads to artifacts at the block boundaries, as shown in Figure 2b. We mitigate this problem by overlapping the individual blocks by 25% on each side, which greatly reduces the artifacts (Figure 2c). In contrast to the global QFT, all blocks are normalized individually and therefore the overall saliency scale is not preserved (e.g. top part of Figure 2c). We normalize each block by its mean frequency magnitude to reduce these scaling errors (Figure 2d). Similar to previous work, we apply a final spatial Gauss blur to remove remove high frequency noise from the saliency result.

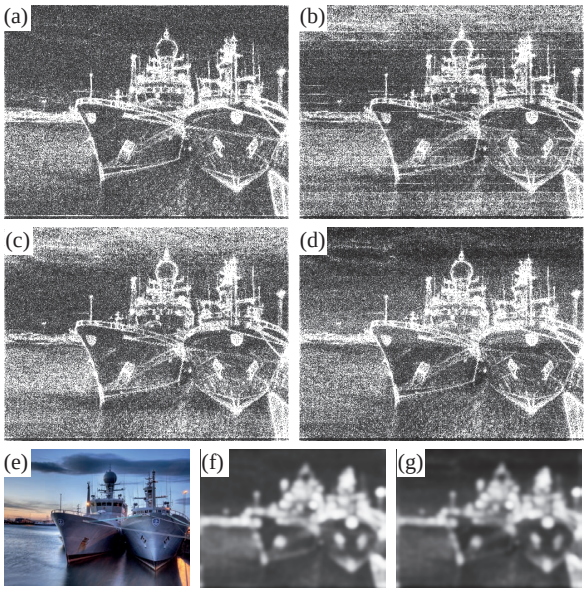

Figure 2: *Saliency estimation algorithm: (a) the original QFT results as proposed in [GMZ08], (b) our block saliency approach, (c) our block saliency approach with overlapping blocks, and (d) our block saliency approach with overlapping blocks and per block scaling. The last row contains the input image (from [RGSS10]), and the filtered output saliency proposed by [GMZ08] (middle), and our approach (right), corresponding to (d).*

3.2. Rectilinear Transformation Computation

In a second step, our algorithm computes a non-linear deformation based on the saliency map described in the previous section.

3.2.1. Saliency Projection

The 2D saliency is projected separately onto a horizontal and a vertical 1D saliency profile, in order to determine the importance of each row and column. As design goal for the 1D projection operator, we strive to achieve high saliency values even if there is only one small but very salient object in a given row or column. As further design choice, adding more salient objects should not increase the saliency compared to just one salient object. We therefore propose a block-wise mean computation of the saliency. To compute the horizontal saliency, each row of the input 2D saliency $\mathcal{S}(x,y)$ is partitioned into contiguous blocks $\mathcal{B}_\mathrm{H} = \{\mathcal{B}_\mathrm{H}^{(i)}\}$. For each of these blocks, the mean saliency value is computed, and the maximum of all mean values is used as 1D projection

$$\mathcal{S}_\mathrm{H}(x) = \max_{\mathcal{B}_\mathrm{H}^{(i)}} \left(\sum_{y \in \mathcal{B}_\mathrm{H}^{(i)}} \mathcal{S}(x,y) / \|\mathcal{B}_\mathrm{H}^{(i)}\| \right), \qquad (1)$$

where $\mathcal{S}_H(x)$ represents the horizontal saliency profile, and the block size $\|\mathcal{B}_H^{(i)}\|$ denotes the number of pixels contained in $\mathcal{B}_H^{(i)}$. The vertical saliency profile $\mathcal{S}_V(y)$ is computed accordingly.

We add a perceptually motivated center bias to the 1D saliency profiles. As shown by the user studies based on real eye tracking [JEDT09], human observers mostly focus on the center region of images and video. The center bias therefore modifies the input saliency to reduce the saliency near the image borders, while retaining the saliency near the center. Since the center bias is perceptually motivated and there is no ideal form, we choose a simple piece-wise linear function in our implementation. The bias reduces the saliency profiles towards the endings of the profiles and keeps the middle 60% constant.

3.2.2. Non-linear Scale Computation

The next step of our algorithm computes the mapping from the 1D saliency profiles to a rectilinear resampling pattern. In order to retain the aspect ratio in important regions, we seek to compute a *scaling vector* that equals 1 in such areas, both in horizontal and vertical direction. The visually less important regions should have a scaling value < 1 in the case of minifications, and a value > 1 in the case of magnifications. Furthermore, the sum of all elements of the *scaling vector* should match the requested target size. In the following, we consider retargeting an input image of dimensions $W \times H$ to a target image of dimensions $W' \times H'$. Due to the downscaling for the saliency computation, both widths and heights need to be downscaled accordingly for this computation.

Minification (W'<W, H'<H). We seek to find a scaling vector that maps important regions to 1 and scales unimportant areas by values < 1. Similar to [ZHM08], the following mapping fulfills these requirements

$$f_k^{\min}(\alpha) = \min(\alpha \mathcal{S}_H(k), 1), \quad \text{s.t.} \sum_{k=1}^{W} f_k^{\min}(\alpha) = W', \quad (2)$$

where $f_k^{\min}(\alpha)$ describes the horizontal scaling vector, and the vertical scaling vector is computed accordingly. Thus, our algorithm needs to determine a value α that fulfills the above nonlinear optimization problem. Since $f_k^{\min}(\alpha)$ is a monotonically increasing function, we can use a simple iterative approach that will always converge to the exact solution. A binary search quickly delivers the solution for a given precision requirement. In our setup, 10 to 15 iterations are necessary to obtain a precision threshold of 10^{-4} pixels.

Magnification (W'>W, H'>H). For magnifying transformations, we need to modify the scaling vector to allow rescaling values larger than 1 for less important image regions. Visually important regions should be kept equal to one. One way of achieving this is by mirroring the scaling vector function (2) around the horizontal axis $f_k^{\min}(\alpha) = 1$, which leads to $f_k^{\text{mag}}(\alpha) = (2 - f_k^{\min}(\alpha))$. Substituting this expression into (2) yields:

$$f_k^{\text{mag}}(\alpha) = 2 - \min(\alpha \mathcal{S}_H(k), 1), \quad \text{s.t.} \sum_{k=1}^{W} f_k^{\text{mag}}(\alpha) = W', \quad (3)$$

where the vertical scaling vector is computed accordingly. Note, that (3) only supports individual pixel magnifications up to a scale of 2, but can easily be adapted for cases with larger aspect ratio differences.

Combination of Horizontal and Vertical Resampling. Our algorithm independently computes the horizontal and vertical scale vectors to match the output resolution W' and H', respectively. To couple the horizontal and vertical scale computation, we propose to solve a different problem with target image size $W'' \times H''$ such that $W''/H'' = W'/H'$. This allows for reformulating the retargeting problem with an additional degree of freedom (DoF) to control the distribution of the non-linear distortions to the horizontal and vertical directions, while keeping the target aspect ratio. The DoF could potentially be obtained with a coupled optimization; however, we found that this optimization often leads to unstable results. We therefore leave the additional DoF as design parameter. Note that this parameter is kept constant for all our experiments.

3.2.3. Spatio-temporal Filtering

The scaling vectors (2), (3) obtained in the previous section are directly related to the input saliency map. Therefore, high-frequency and low-frequency fluctuations in the saliency will directly lead to spatio-temporal fluctuations in the output video. In contrast to previous work, which often uses regularization terms for a non-linear optimization, we directly enforce regularization by applying post filters after the non-linear scale computation.

First, high-frequency content is removed using an acausal finite impulse response (aFIR) filter. We collect the horizontal and vertical scaling vectors over time (e.g., 10 frames), and assemble each of them in a separate matrix. We then apply a 2D smoothing filter on the spatio-temporal scaling matrices. The filter is acausal in the time domain in order to consider future changes in the scaling vectors. Different aFIR filters such as bilateral filters and Gaussian filters have been evaluated, and we found that all tested filters provide similar output results.

In order to be able to filter low-frequency content efficiently in hardware, we use an infinite impulse response (IIR) filter in a second step. The advantage of the IIR filter is that the cut-off frequency is not limited by the filter length, but by the filter coefficients. We use a one-tap IIR filter

$$s_{\text{out}}[k] = a s_{\text{out}}[k-1] + (1-a) s_{\text{in}}[k], \quad 0 \le a \le 1, \quad (4)$$

where k corresponds to the time index. The filter coefficient a is largely dependent on the type of scene, and should be

adapted to the video content. In the current form, we set a manually ($a \approx 0.9$ for the video sequences, the 'cowboy' scene has a higher value), but the coefficient might be adapted automatically using image analysis tools to detect e.g. scene cuts, global camera motions or static backgrounds. For instance, scene cuts could be used to clear the IIR filter ($a = 0$); low global motion should result in a large a, large global motion in a smaller a.

Linear upsampling. Before generating the resampling grid, the horizontal and vertical scaling vectors need to be linearly upsampled to achieve the actual target resolution.

3.3. EWA Rendering

The filtered and upsampled scaling vectors from the previous section are first converted to a resampling grid using a cumulative sum. In order to retarget the input image, we choose an elliptical weighted average (EWA) [GSH*12] resampling algorithm which implicitly handles anti-aliasing very efficiently. In contrast to the arbitrary deformations allowed in [GSH*12], our algorithm generates rectilinear deformations which allows for various simplifications. In the following, we will briefly summarize the original EWA algorithm, before listing our simplifications.

The EWA framework performs image resampling using forward mapping. In a first step, a continuous input signal is reconstructed using Gaussian kernels as interpolation filters. More specifically, each input pixel w_k is represented by a circular 2D Gaussian reconstruction kernel with covariance matrix $V_i = \sigma_i^2 I_2$, where σ_i^2 denotes the variance of the kernel and I_2 denotes the 2-by-2 identity matrix. The reconstruction kernels are then transformed to target space, and low-pass filtered using a circular 2D Gaussian anti-aliasing kernel with covariance matrix $V_a = \sigma_a^2 I_2$. Finally, all kernels are evaluated and accumulated at the output pixel locations. The resampling equation of EWA splatting is defined as [ZPBG02]:

$$f_{\mathrm{EWA}}(\mathbf{x}) = \sum_{k \in D_s} \frac{w_k |J_k|}{2\pi |\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-m(\mathbf{u}_k))^T \Sigma^{-1} (\mathbf{x}-m(\mathbf{u}_k))}, \quad (5)$$

where $\mathbf{u}_k$ is the spatial position of the pixel with index k in the source image D_s, and w_k denotes the RGB value of the pixel. The image transformation function $m(\mathbf{u})$ is defined by the resampling grid, and the Jacobian J_k denotes the locally affine approximation of $m(\mathbf{u})$. The covariance matrix of the target kernel is $\Sigma = J_k^T V_i J_k + V_a$.

In our application, J_k can be efficiently determined by using a Sobel filter evaluated on the output grid positions. Also, every input kernel will only be transformed by an axis-aligned non-uniform scaling matrix followed by a translation. Therefore, the off-diagonal elements of J_k are always zero. This considerably simplifies the overall method, since $\Sigma = \sigma_i^2 J_k^2 + \sigma_a^2$ is diagonal. For the selection of σ_i and σ_a we use the adaptive anti-aliasing approach of [GSH*12], which replaces Σ with $\tilde{\Sigma}$

$$\tilde{\Sigma}(n,n) = \max(\sigma_a^2, \sigma_i^2 J_k^2(n,n)), \quad n = 1,2 \quad (6)$$

with $\sigma_a = \sigma_i \approx 0.39$.

Thus, the final (simplified) adaptive EWA rendering expression reformulates to

$$f_{\mathrm{EWA}}(\mathbf{x}) = \sum_{k \in D_s} c_k e^{-\frac{1}{2}(d_{k,1}^2/\Sigma_{1,1} + d_{k,2}^2/\Sigma_{2,2})}, \quad (7)$$

with $d_{k,i} := x(i) - m(u_k(i))$, and

$$c_k = w_k \frac{|J_k|}{2\pi\sqrt{|\tilde{\Sigma}|}},$$
$$c_k = w_k \frac{\min(1, J_k(1,1)) \min(1, J_k(2,2))}{2\pi\sigma_i^2}. \quad (8)$$

4. Hardware Architecture

In the following, we describe an efficient hardware architecture for the algorithm proposed in the previous section. The architecture is designed for real-time performance of 1080p60 video (1920x1080 resolution at 60 frames per second), and is very efficient in terms of hardware resources. Figure 3 shows a system-level overview of our retargeting architecture. The architecture is partitioned into device-dependent infrastructure blocks and device-independent core functionality. All blocks are operating in a pipelined fashion, i.e. all parts are able to operate in parallel on possibly different images. The architecture could easily be realized with either an FPGA or ASIC implementation.

The retargeting core is divided into saliency estimation, scale estimation, and rendering. The saliency estimation block receives two input video streams from the external frame buffer, where the first video stream contains the current frame and the second stream contains the previous frame. Then, the spatio-temporal QFT is computed using the two input frames, and the resulting saliency values are blurred for smooth results. As a next step, the scale estimation unit computes the non-linear scale based on the maximum block-mean saliency values, and sends the temporally filtered and upsampled retargeting patterns to the rendering block. As a last step, the rendering block performs EWA sampling on the full resolution input stream – note, that this input stream is delayed by multiple frames, due to the latencies of the previous computations as well as the acausal filter. The retargeted video is then streamed to a standard video output (DVI/HDMI).

4.1. Saliency Estimation

Figure 4 provides a simplified block diagram that shows the main components of the datapath. The *quaternion* transformation is realized using a simple arithmetic datapath, before sending the data to the *FFT* cores. In its original formulation, the QFT algorithm requires one quaternion FT and

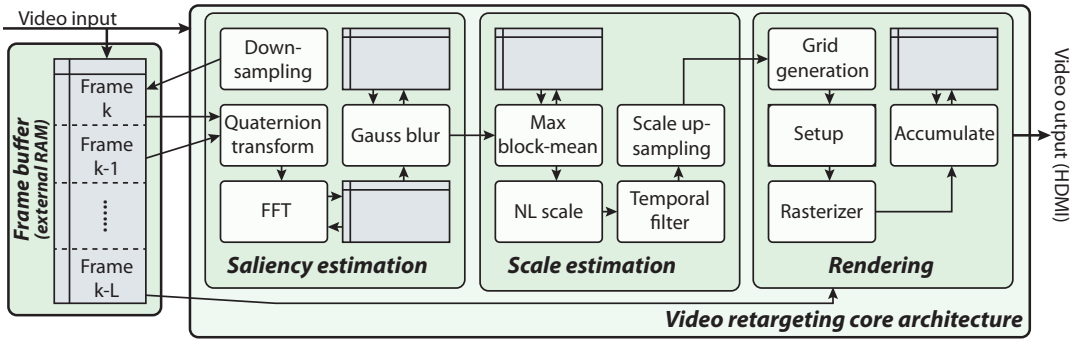

Figure 3: *Hardware architecture overview. The saliency estimation unit performs a QFT-based frequency analysis, and blurs the resulting saliency to achieve smooth results. The scale estimation block determines the optimal rectilinear transformation, which is then rendered in the subsequent rendering unit. Note that the blocks corresponding to the infrastructure have been omitted for better clarity.*

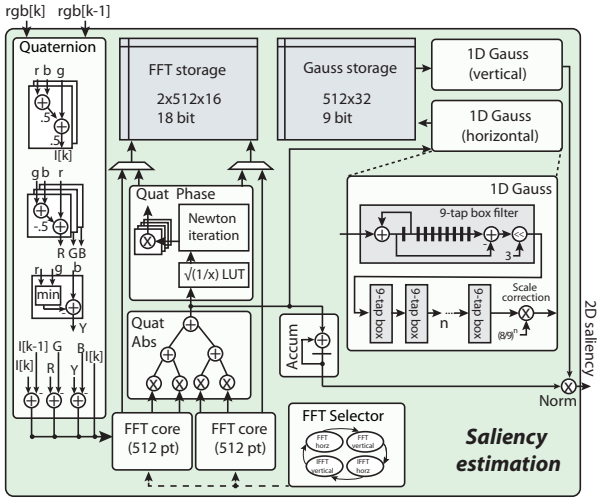

Figure 4: *Block diagram of saliency estimation unit. See Figure 9 for an explanation of the symbols.*

one inverse quaternion FT, which can be realized using four 1D FFTs and four 1D inverse FFTs (see Section 3.1). Since the the saliency is computed on a low-resolution image, the necessary data rate of the FFTs is low compared to the rest of the system. This makes it possible to use only two 1D FFT cores, which evaluate the eight required FFTs sequentially in a time-shared manner. An on-chip RAM buffer is used as temporary *FFT storage* and for transposing and bit-interleaving the 1D Fourier outputs. The *quaternion norm and phase extraction* are realized using a simple arithmetic datapath.

The resulting saliency map is spatially low pass filtered, using a 2D *Gaussian* kernel with a variance 5.8, similar to the initially proposed QFT algorithm. A direct implementation of the 2D Gaussian would require a minimal filter size of 34 pixels, which in turn would require a considerable amount of resources: 34^2 multiplications and $\log_2(34^2)$ additions for each saliency value. We apply two algorithmic transformations that greatly reduce the resource footprint: first, we exploit the separable filter property of the 2D Gaussian, and perform two 1D Gaussian filters instead. Second, we approximate the 1D Gaussian by a series of 9-pixel wide box filters, which converge to a Gaussian (central limit theorem). The box filters are efficiently realized with a 9-tap accumulation delay line followed by a subtraction. In practice, four box filters result in a reasonable approximation of a 1D Gaussian with variance 5.8.

The horizontally filtered image is stored in an intermediate *Gauss storage* buffer before filtering in the vertical dimension. We choose a buffer of 32 downsampled saliency lines, in order to support filtering over two consecutive blocks. The final filtered values are then *normalized for each block* by the sum of previously computed absolute quaternion values.

4.2. Resampling Grid Generation

A simplified block diagram of the resampling grid calculation is given in Figure 5. In a first step, the incoming horizontal and vertical saliency values are accumulated in the *max block-mean unit.* Once a block boundary is reached, the accumulated value is compared to the current maximum, and the accumulators are cleared. Two output buffers store the current saliency vectors. An additional intermediate buffer is required to perform the vertical saliency accumulation, since the input saliency is streamed horizontally. The final saliency vectors are ready as soon as the last saliency value of a frame is received by the mean max block.

As soon as the saliency vectors are ready, the *non-linear scaling block* starts its binary search for the non-linear scale

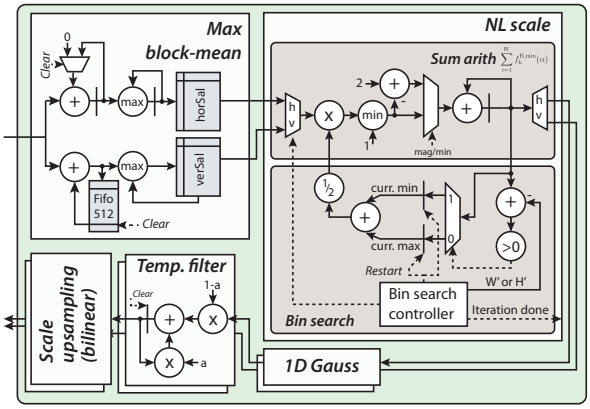

Figure 5: *Block diagram of the resampling grid generation unit. See Figure 9 for an explanation of the symbols.*

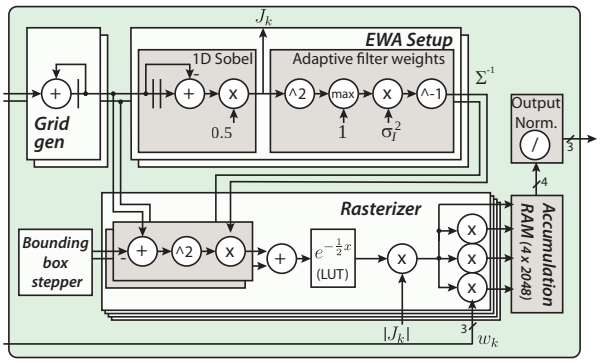

Figure 6: *Block diagram of the adaptive EWA rendering unit. See Figure 9 for an explanation of the symbols.*

factor, both for the horizontal and vertical vectors. The block works in an iterative manner and stops as soon as the predefined error threshold

$$\sum_{i}^{W} f_k(\alpha) - W' \leq 10^{-4},$$

is reached.The computation time of this block is negligibly compared to the the other blocks, since convergence is reached after 10 to 15 iterations and each iteration requires at most 512 cycles, i.e., the dimensions of the downsampled saliency image.

After the non-linear scaling vectors are determined, the vectors are stored in a temporary buffer. The horizontal and vertical scaling vectors are then *filtered* with a 1D Gaussian each, where the implementation of the Gaussian filter is equivalent to the ones used for the saliency computation. Finally, the filtered scaling vectors are linearly *upsampled* to the target output resolution.

4.3. EWA Rendering

Figure 6 shows a block diagram of the rendering unit. The datapath is very similar to [GSH*12], although most of the matrix operations simplify to scalar operations due to absence of off-diagonal terms. Furthermore, the necessary size for the accumulation buffer is much smaller due to the regular access pattern, and only a couple of lines need to be buffered in the on-chip SRAM blocks.

As a first step, the *transformation grid* is computed by a cumulative sum with a simple accumulator. Then, in the *EWA setup stage*, a 1D Sobel filter is used to compute the Jacobian, and the inverted diagonal terms of the covariance matrix are computed subsequently. A bounding box stepper iterates over all pixels within each Gaussians' bounding box, similar to [GSH*12]. The resulting sampling locations are sent to four *rasterizers* operating in parallel. The exponential function is realized as look-up table, using linear interpolation between the stored sampling points. Finally, the color channels are multiplied with the normalized filter value, and are then sent to the accumulation buffer.

The *accumulation buffers* are realized using four separate on-chip SRAM components, one for each (even,odd) - (row,column) combination. This greatly simplifies the assignment of sampling coordinates to the individual rasterizers, and furthermore allows to accumulate four independent pixel contributions in parallel. Every buffer access is realized with a read-accumulate-write operation, and the final read-out operation is followed by clearing the respective data. Due to the interpolating nature of the Gaussian kernels, a final *output normalization* with the accumulated filter weight is necessary. The resulting output is streamed directly to the output video interface.

Since video interfaces usually require cycle-accurate pixel data output, the external video interface dictates the processing speed. To ensure a constant read-out, the accumulation buffer is designed to be always almost-full. To avoid overflows, the input to the rendering is stalled when the almost-full threshold is reached, whereas underflows are avoided by ensuring that the input rate and processing rates are higher or equal than the output rate.

5. Results and Limitations

We evaluated our architecture in terms of quality, as well as in terms of a dedicated FPGA implementation and a (simplified) version on a Tegra 2 embedded processor.

5.1. Quality Assessment

The goal of this work is to provide an efficient retargeting algorithm that is suitable for hardware integration. While

Figure 7: *Selected examples from the RetargetMe dataset [RGSS10], compared to related work and linear stretching and center cropping. Our algorithm is able to produce similar results at much lower computational complexity.*

we did not strive to outperform other video retargeting algorithms in terms of quality, we are able to achieve results comparable to much more complicated retargeting approaches.

We evaluate the results of our algorithm on the RetargetMe data set [RGSS10]. Figure 7 shows an excerpt of the results, the full data set together with video sequences of our running system are provided as additional material. Although we did not perform any user study, the results of our work are very similar to the top-performing approaches [KLHG09, PWS12], and it is safe to assume that our approach would score similarly high. Additionally, our algorithm fails only in cases in which the methods of [KLHG09, PWS12] also fail. Such cases occur when the saliency estimation produces incorrect results, or when certain geometric structures such as circles or diagonal lines are present. Due to the streaming nature of our algorithm, temporal artifacts similar to [KLHG09] can be noticable for video retargeting.

5.2. FPGA Implementation

The video retargeting architecture is implemented using VHDL, and is targeted for ALTERA FPGAs (Cyclone IV/Stratix IV). The employed TERASIC development board supports DVI/HDMI input and output, and features an external SDRAM component. In addition to the retargeting core, we implemented an interface to an external SDRAM device as well as standard-compliant DVI/HDMI receiver and transmitter. The SDRAM controller is interfaced with a multi-port interface, in order to arbitrate multiple accesses from different units to the same physical SDRAM component.

Table 1 summarizes the resource utilization of our architecture. The design is very small in terms of logic resources. We use the available on-chip SRAM blocks quite freely, as modern FPGAs usually contain an abundance of such resources. For a corresponding ASIC implementation, a considerable amount of the on-chip RAM could be saved by careful dimensioning of every SRAM component. Furthermore, parts of the on-chip caches such as the saliency blur cache or the QFT caches could be moved to an external memory. The buffer of the rendering part might also be reduced at the cost of sacrificing rendering quality.

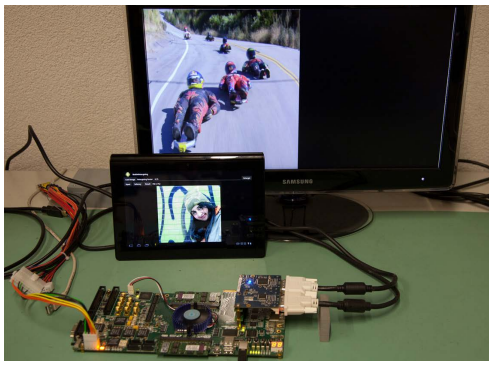

Figure 8: *Our FPGA prototype, connected to a LCD monitor, and our Tegra 2 prototype.*

Table 1: *ALTERA FPGA resource utilization, divided into FPGA logic look-up tables (LUTs), register bits (Regs), on-chip static block rams (BRAM), and embedded multiply-accumulate units (DSPs).*

	LUTs	Regs	BRAM (bit)	DSPs
downsc.	224	180	31K	19
saliency	8466	13391	816K	96
grid	993	562	87K	12
rendering	4071	2731	483K	80
total core	13762	16864	1416K	207
DVI	140	349	0	0
Mem IF.	5297	5169	337K	0
total infr.	5437	5518	337K	0

The maximum achievable clock frequency of the retargeting core is 135 Mhz, which is sufficient to achieve 1080p60 real-time performance. Note that a clock of at least 124 MHz would be necessary to achieve this throughput. The I/O blocks for DVI and SDRAM accesses run in separate clock domains, and are decoupled from the core clock. Our design is easily scalable to higher resolutions and framerates, since the core functionality could be parallelized to a good degree.

5.3. Tegra 2 Implementation

In order to compare our performance to embedded systems, a slightly modified version of the retargeting algorithm is

implemented on a Tegra 2 tablet. The implementation does not include the temporal filtering, but it features all other functionality of the hardware architecture. All parts of the algorithm but the rendering are executed on the ARM CPU, using one core only. For the rendering, we use the hardware OpenGL ES 2.0 acceleration with texture-mapped quads. The quad mesh is generated at the downsampled image resolution (512 quads horizontal). Linear upscaling of the warp is implicitly performed using the quad mesh.

The Tegra 2 tablet is able to retarget 1080p images in about 520ms. Most of the time is spent in the downsampling operation (49ms), the saliency computation (207ms), and the rendering block (263ms). Note, that the downsampling operation as well as FFT operation are performed using OpenCV, which might not be fully optimized for the ARM architecture. More recent Tegra 3 architectures and multicore processing would speed up the algorithm substantially, however, 1080p60 performance is still challenging.

5.4. Limitations

The quality of our retargeting algorithm mainly depends on the saliency estimation, similar to previous work. The employed QFT algorithm performs reasonably well in many situations, however, future algorithms might show better performance-quality trade-offs. Another possible extension could be face detection or other application-specific object detectors.

Our algorithm can lead to temporally unstable results, which are noticeable as low-frequency artifacts. This is mainly due to the employed filter sizes and IIR filter coefficients. If the filter sizes are small, the deformation grid can fluctuate over time for quasi-static scenes. On the other hand, if the filter sizes are too big, fast moving object or camera motion cannot be captured by the slower change of the deformation grid. Adding motion estimation could serve as an image analysis tool to adapt the filter coefficients, which would improve the result. Note that all streaming video retargeting algorithms including [ZHM08,SWDL09,KLHG09,KJJK11] have similar limitations on the maximum allowable scene motion.

6. Conclusion

In this work, we derived a hardware-efficient algorithm for content-aware video retargeting. By adapting and modifying state-of-the art video retargeting algorithms, we could considerably reduce the algorithmic complexity, and thus develop an efficient hardware architecture. We presented an FPGA prototype implementation that is able to retarget 1080p60 video streams at low hardware costs, and our architecture could easily be integrated into displays or mobile devices. Despite the reduced computational complexity, our results are comparable to state-of-the-art approaches.

As final observation, we conclude that state-of-the-art video retargeting is not fully ready for integration into consumer electronics, since artifacts cannot be avoided in all situations. As a step toward artifact-free retargeting, our architecture could include further image analysis components such as scene cut detection or optical flow. Furthermore, we think that with upcoming algorithmic improvements for temporal consistency [LWA*12] and with the advent of scalable video coding [WSL*11], real-time video retargeting is a valuable component for upcoming video applications.

References

[BCP*11] BAE S., CHO Y., PARK S., IRICK K., JIN Y., NARAYANAN V.: An FPGA implementation of information theoretic visual-saliency system and its optimization. In *IEEE International Symposium on Field-Programmable Custom Computing Machines* (2011), pp. 41–48. 2

[CS00] CHATTERJEE S., SEN S.: Cache-efficient matrix transposition. In *IEEE International Symposium on High-Performance Computer Architecture* (2000), pp. 195 –205. 3

[GMZ08] GUO C., MA Q., ZHANG L.: Spatio-temporal saliency detection using phase spectrum of quaternion fourier transform. In *IEEE Conference on Computer Vision and Pattern Recognition,* (2008), pp. 1 –8. 3

[GSH*12] GREISEN P., SCHAFFNER M., HEINZLE S., RUNO M., SMOLIC A., BURG A., KAESLIN H., GROSS M.: Analysis and VLSI implementation of EWA rendering for real-time HD video applications. *IEEE Transactions on Circuits and Systems for Video Technology (to appear)* (2012). 2, 5, 7

[GW08] GONZALEZ R. C., WOODS R. E.: *Digital Image Processing*. Pearson Education International, 2008. 3

[JDT10] JUDD T., DURAND F., TORRALBA A.: Fixations on low resolution images. *Journal of Vision 10*, 7 (2010), 142–142. 3

[JEDT09] JUDD T., EHINGER K., DURAND F., TORRALBA A.: Learning to predict where humans look. In *IEEE International Conference on Computer Vision* (2009), pp. 2106–2113. 4

[KJJK11] KIM J.-S., JEONG S.-G., JOO Y., KIM C.-S.: Content-aware image and video resizing based on frequency domain analysis. *IEEE Transactions on Consumer Electronics 57*, 2 (2011), 615 –622. 2, 9

[KLHG09] KRÄHENBÜHL P., LANG M., HORNUNG A., GROSS M.: A system for retargeting of streaming video. *ACM Transactions on Graphics 28*, 5 (2009). 1, 2, 3, 8, 9

[KSVC07] KNOCHE H.AND PAPALEO M., SASSE M. A., VANELLI-CORALLI A.: The kindest cut: Enhancing the user

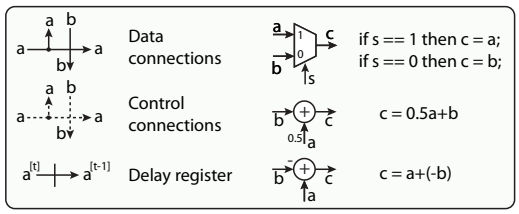

Figure 9: *Explanation of commonly used symbols. All block diagrams only show functional delay registers, and pipeline registers have been omitted for clarity.*

experience of mobile TV through adequate zooming. In *ACM Multimedia* (2007), pp. 87–96. 1

[LWA*12] LANG M., WANG O., AYDIN T., SMOLIC A., GROSS M.: Practical temporal consistency for image-based graphics applications. *ACM Transactions on Graphics (to appear)* (2012). 9

[PWS12] PANOZZO D., WEBER O., SORKINE O.: Robust image retargeting via axis-aligned deformation. *Computer Graphics Forum 31*, 2 (2012). 2, 3, 8

[RGSS10] RUBINSTEIN M., GUTIERREZ D., SORKINE O., SHAMIR A.: A comparative study of image retargeting. *ACM Transactions on Graphics 29*, 5 (2010), 160:1–160:10. 1, 2, 3, 8

[RSA08] RUBINSTEIN M., SHAMIR A., AVIDAN S.: Improved seam carving for video retargeting. *ACM Transactions on Graphics 27*, 3 (2008), 1–9. 1

[RSA09] RUBINSTEIN M., SHAMIR A., AVIDAN S.: Multi-operator media retargeting. *ACM Transactions on Graphics 28*, 3 (2009), 1–11. 1

[STR*05] SETLUR V., TAKAGI S., RASKAR R., GLEICHER M., GOOCH B.: Automatic image retargeting. In *International Conference on Mobile and Ubiquitous Multimedia* (2005), pp. 59–68. 1

[SWDL09] SHI L., WANG J., DUAN L., LU H.: Consumer video retargeting: context assisted spatial-temporal grid optimization. In *ACM Multimedia* (2009), pp. 301–310. 2, 9

[WFS*09] WANG Y.-S., FU H., SORKINE O., LEE T.-Y., SEIDEL H.-P.: Motion-aware temporal coherence for video resizing. *ACM Transactions on Graphics 28*, 5 (2009), 127:1–127:10. 2

[WGCO07] WOLF L., GUTTMANN M., COHEN-OR D.: Non-homogeneous content-driven video-retargeting. In *IEEE International Conference on Computer Vision* (2007), pp. 1–6. 1

[WLSL10] WANG Y.-S., LIN H.-C., SORKINE O., LEE T.-Y.: Motion-based video retargeting with optimized crop-and-warp. *ACM Transactions on Graphics 29* (2010), 90:1–90:9. 2

[WSL*11] WANG Y., STEFANOSKI N., LANG M., HORNUNG A., SMOLIC A., GROSS M.: Extending SVC by content-adaptive spatial scalability. In *IEEE International Conference on Image Processing* (2011), pp. 3493–3496. 9

[YSWL08] YU-SHUEN WANG CHIEW-LAN TAI O. S., LEE T.-Y.: Optimized scale-and-stretch for image resizing. *ACM Transactions on Graphics 27*, 5 (2008). 1

[YSWL11] YU-SHUEN WANG JEN-HUNG HSIAO O. S., LEE T.-Y.: Scalable and coherent video resizing with per-frame optimization. *ACM Transactions on Graphics 30*, 4 (2011). 2

[ZHM08] ZHANG Y.-F., HU S.-M., MARTIN R. R.: Shrinkability maps for content-aware video resizing. *Computer Graphics Forum 27* (2008), 1797–1804. 2, 4, 9

[ZPBG02] ZWICKER M., PFISTER H., BAAR J. V., GROSS M.: EWA splatting. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 223–238. 5