Spatially-Varying Image Warping: Evaluations and VLSI Implementations

Pierre Greisen, Michael Schaffner, Danny Luu, Val Mikos, Simon Heinzle, Frank K. Gürkaynak, Aljoscha Smolic

To cite this version:

Pierre Greisen, Michael Schaffner, Danny Luu, Val Mikos, Simon Heinzle, et al.. Spatially-Varying Image Warping: Evaluations and VLSI Implementations. 20th International Conference on Very Large Scale Integration (VLSI-SoC), Aug 2012, Santa Cruz, CA, United States. pp.64-87, 10.1007/978-3-642-45073-0_4 . hal-01456962

HAL Id: hal-01456962

https://inria.hal.science/hal-01456962

Submitted on 6 Feb 2017

Spatially-Varying Image Warping: Evaluations and VLSI Implementations

Pierre Greisen[1,2], Michael Schaffner[1,2], Danny Luu[1], Val Mikos[1], Simon Heinzle[2], Frank K. Gürkaynak[1], Aljoscha Smolic[1,2]

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

Abstract. Spatially-varying, non-linear image warping has gained growing interest due to the appearance of image domain warping applications such as aspect ratio retargeting or stereo remapping/stereo-to-multiview conversion. In contrast to the more common global image warping, e.g., zoom or rotation, the image transformation is now a spatially-varying mapping that, in principle, enables arbitrary image transformations. A practical constraint is that transformed pixels keep their relative ordering, i.e., there are no fold-overs. In this work, we analyze and compare spatially-varying image warping techniques in terms of quality and computational performance. In particular, aliasing artifacts, interpolation quality (sharpness), number of arithmetical operations, and memory bandwidth requirements are considered. Further, we provide an architecture based on Gaussian filtering and an architecture with bicubic interpolation and compare corresponding VLSI implementations.

Keywords: image-based rendering, EWA splatting, bicubic resampling, video processing, anti-aliasing, interpolation quality, complexity evaluation, caching, VLSI

1 Introduction

With the steadily increasing frame rates and resolutions, real-time video processing and graphics processing is becoming predominant in terms of computational requirements in mobile devices. Many application-specific hardware cores for video processing are currently being integrated onto mobile system-on-chips (SoCs) (e.g., NVIDIA Tegra). One upcoming application for mobile devices is video content adaptation: while a growing amount of content is watched on an increasing number of different mobile platforms, most content is captured with one acquisition system at fixed parameters. Examples for content adaption algorithms are content-aware video resizing (video retargeting) [13], non-linear stereoscopic 3D (S3D) adaption [14], 2D to S3D conversion and S3D to multiview generation [5,6,19]. Other content transformation applications are camera alignment for S3D video and panoramic shots.

As a first step, any display adaptation algorithm determines an image *warping function* that is dependent on the display characteristics. The input frames are then transformed to the output frames according to the given warping function using a *view rendering* algorithm based on spatially-varying warping. The generation of the warping function is application-specific, and can be separated from the view rendering. For instance in video retargeting, the warping function retains the aspect ratio of *salient* (i.e., visually important) parts of the image, while the image distortion is hidden in visually less important regions. In S3D to multi-view conversion, the warping function is derived from the 3D structure of the scene (obtained from a disparity estimation step) to generate in-between views.

Various view synthesis and image rendering architectures have been proposed in prior work. However, the majority of these architectures have been optimized for one particular rendering application, such as depth-image based rendering (DIBR) [3, 11], stereo rectification [7], or non-linear lens correction [2, 17]. In contrast, we consider warping with general transformations that can be used for global per-frame transformations such as (wide-angle) lens undistortion, but also for spatially varying per-pixel transformations such as in video retargeting. Due to the spatially-varying nature of the transformation and the high resolutions of video footage in current applications, special care has to be taken in algorithm and architecture design. That is, aliasing needs to be avoided, high-quality interpolation should be guaranteed, and high computational – and memory bandwidth requirements need to be addressed.

In this paper, we address hardware efficiency and VLSI architectures of non-linear warping for view synthesis applications. It is an extended version of our previous work [8, 9]: next to the *elliptical weighted average (EWA) rendering system* presented in [8, 9] we present *non-linear image warping through bicubic interpolation and adaptive super-sampling* and assess *image quality and hardware requirements* by comparing an extended set of non-linear warping strategies. An important hardware consideration is thereby the memory bandwidth requirements and the corresponding *cache simulations and VLSI designs*. Finally, using the obtained ASIC implementation results, we provide a comparison of different warping techniques in terms of VLSI performance.

2 Non-linear Image Warping

Non-linear image warping is the process of geometrically transforming an image with a general image transformation (warping) function. In the simplest case, the image warping function can be represented as a global per-image transformation such as a rotation or translation of all the pixel values. Such transformations are usually represented by simple, per-image arithmetic operations of the input pixel locations. Stereo rectification is a practical application example: two non-aligned camera images are rectified in order to eliminate any vertical offsets between the cameras, and a 3-by-3 matrix with 8 degrees of freedom is enough to specify the full image transformation.

While our setup is able to perform global per-image transformations, its strength lies in the ability to realize locally-adaptive non-linear deformation of

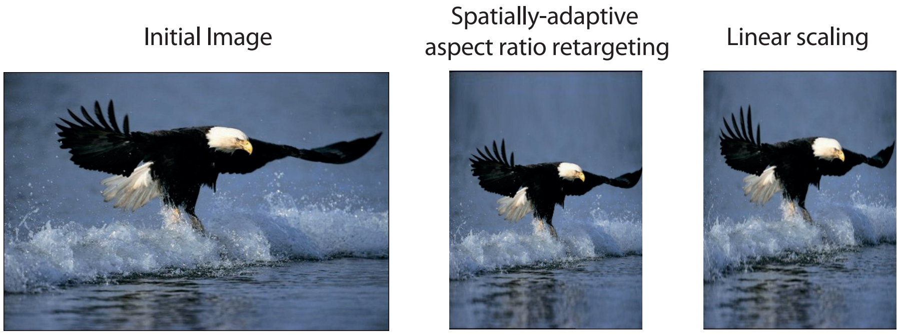


Fig. 1. Example of transformations possible with our non-linear image warping setup. In addition to global per-frame transformations such as rotations or linear scaling (right image) our system also allows arbitrary non-linear transformations (middle image). Such transformations are essential for spatially-adaptive retargeting applications. Image credits: the initial image (left) is in the public domain, the middle image is generated with the framework from [13] and the right image is a linear scaled version of the initial image.

the input video, which is required in modern video applications such as content-aware video retargeting. Our warping function can be specified by a per-pixel mapping function: any pixel in the source image is assigned its own destination pixel position in the target image. Figure 1 shows an example of transformations that are possible with the system presented in this work.

2.1 Warping Basics

In the following we briefly summarize the image resampling process, for a thorough derivation we refer to literature (e.g. [18, 21, 22]). Consider an input image with pixel values w_k, where k is the linearized image coordinate and $\mathbf{u}_k$ the corresponding 2D coordinate in source (image) space. Each (non-integer) source coordinate $\mathbf{u}$ is transformed via mapping $\mathbf{m}$ to a target coordinate $\mathbf{x}$

$$\mathbf{x} = \mathbf{m}(\mathbf{u}) \tag{1}$$

Further, let $f_{\mathrm{i}}()$ be a continuous source space interpolation filter and $f_{\mathrm{a}}()$ a continuous target space anti-aliasing filter. The general mapping function $\mathbf{m}$ then transforms an input image into an output image f_{out} according to

$$\begin{aligned} f_{\mathrm{out}}(\mathbf{x}) &= \sum_k w_k f_{\mathrm{i}}(\mathbf{m}^{-1}(\mathbf{x}) - \mathbf{u}_k) * f_{\mathrm{a}}(\mathbf{x}) \\ &= \int_{\mathbb{R}^2} \sum_k w_k f_{\mathrm{i}}(\mathbf{m}^{-1}(\boldsymbol{\tau}) - \mathbf{u}_k) h(\mathbf{x} - \boldsymbol{\tau}) d\boldsymbol{\tau}. \end{aligned} \tag{2}$$

The interpolation and anti-aliasing filter are crucial for obtaining good image quality in the resampling process, and omitting them can lead to aliasing or holes in the output image, especially for spatially-varying transformations. The final output image is obtained by evaluating f_{out} on the desired integer grid positions.

In practice, the general mapping function $\mathbf{m}(\mathbf{u})$ is linearly approximated with a first-order Taylor expansion around an integer grid position $\mathbf{u}_k$

$$\mathbf{x} = \mathbf{m}(\mathbf{u}) \approx \mathbf{m}(\mathbf{u}_k) + \mathbf{J}_k \cdot (\mathbf{u} - \mathbf{u}_k), \tag{3}$$

where $\mathbf{J}_k$ is the 2×2 Jacobian matrix of $\mathbf{m}$ at position $\mathbf{u}_k$. Also, the resampling equation (2) can be evaluated in two ways. The so-called backward mapping approach steps through the output pixel positions $\mathbf{x}_h$ and looks for the corresponding pixels in the input image. The forward mapping approach steps through the input grid positions $\mathbf{u}_k$ and calculates its contributions to the target pixels. In the following, we introduce practical backward and forward mapping techniques for non-linear warping.

2.2 Forward Mapping: EWA Splatting

An efficient forward mapping approach is elliptical weighted average (EWA) splatting [9]. In the EWA framework, 2D Gaussian filters are used for both interpolation and anti-aliasing filters with covariance matrices $V_{\{\mathrm{i,a}\}} = \sigma^2_{\{\mathrm{i,a}\}} I_2$, where I_2 is a 2-by-2 identity matrix. Two main advantages of Gaussian filters make the EWA framework very effective: first, a Gaussian filter remains Gaussian under linear transformations. Second, the convolution of two Gaussian filters results in another Gaussian.

Consider an input image with pixel values (intensities or RGB components) w_k, where k is the linearized 2D image coordinate corresponding to the 2D position vector $\mathbf{u}_k$ and an Taylor-approximated mapping $\mathbf{m}(\mathbf{u}_k) + \mathbf{J}_k(\mathbf{u} - \mathbf{u}_k)$. The complete EWA resampling process is then summarized as follows. First, the per-pixel covariance matrix is calculated from the warping grid Jacobian $\mathbf{J}_k$ and covariance matrices $\mathbf{V}_{\{\mathrm{a,i}\}}$

$$\mathbf{C}_k = \mathbf{J}_k \mathbf{V}_{\mathrm{i}} \mathbf{J}_k^T + \mathbf{V}_{\mathrm{a}}. \tag{4}$$

The technique from [9] further adapts the resulting co-variance matrix $\mathbf{C}_k$ on a per-pixel level and thereby optimizes the inherent blur-aliasing trade-off of Gaussian filters (see [9] for details). Next, for each input pixel k with position $\mathbf{u}_k$ and value w_k, we accumulate its contributions in the target image v_h on target grid positions $\mathbf{x}_h$ with linear index h

$$v_h \leftarrow \frac{w_k |\mathbf{J}_k|}{2\pi\sqrt{|\mathbf{C}_k|}} e^{-0.5(\mathbf{x_h} - \mathbf{m}(\mathbf{u}_k)^T \mathbf{C}_k^{-1} (\mathbf{x_h} - \mathbf{m}(\mathbf{u}_k))}. \tag{5}$$

The '$\leftarrow$' symbol denotes an update operation (accumulation). Due to non-idealities, a post-normalization step is necessary: v_h/ρ_h, where ρ_h are the accumulated weights

$$\rho_h \leftarrow \frac{|\mathbf{J}_k|}{2\pi\sqrt{|\mathbf{C}_k|}} e^{-0.5(\mathbf{x_h} - \mathbf{m}(\mathbf{u}_k)^T \mathbf{C}_k^{-1} (\mathbf{x_h} - \mathbf{m}(\mathbf{u}_k))}. \tag{6}$$

In theory, $\mathbf{x}_h$ is the complete target image grid; in practice, because of the fast decay of the Gaussian kernel, the range of $\mathbf{x}_h$ can be confined by a rectangular bounding box around the transformed center of the Gaussian $\mathbf{m}(\mathbf{u}_k)$ [9]

$$\mathbf{m}(\mathbf{u}_k) + \begin{pmatrix} \pm\sqrt{\mathbf{C}_k(1,1)} \\ \pm\sqrt{\mathbf{C}_k(2,2)} \end{pmatrix}. \tag{7}$$

2.3 Backward Mapping

Backward mapping approaches do not accumulate contributions from source pixels in the target image but, conversely, perform a direct look-up for each target pixel in the source image. In order to evaluate the analytical resampling expression without using Gaussian filters, the anti-aliasing filter is usually replaced by a practical anti-aliasing technique. The resampling expression simplifies to an interpolation in source space. There exist a variety of different filter kernels that can be used for the interpolation, such as nearest-neighbor, bilinear, bicubic, or windowed-sinc interpolation kernels. In the quality evaluation section, the performance of the interpolation kernels is evaluated and compared numerically to each other.

The general backward evaluation formula can be written as

$$\rho_h = \sum_k w_k f_i(\mathbf{m}^{-1}(\mathbf{x_h}) - \mathbf{u}_k). \tag{8}$$

For instance, in the simple case of nearest-neighbor interpolation, the expression becomes $\rho_h = w_{k'}$ with $k' = \text{argmin}_k |\mathbf{m}^{-1}(\mathbf{x}_h) - \mathbf{u}_k|^2$. Expressions for other interpolation filters can be derived similarly or looked-up in literature [21].

To add anti-aliasing on top of the practically efficient backward interpolation technique, different approaches exist. One is super-sampling and decimation, where image values are looked-up on a higher-resolution output grid and then decimated again to the actual required resolution. The decimation filter thereby serves as anti-aliasing filter. Another technique is mip-mapping, which is used in the texture mapping stage in current GPUs [1]. Mip-mapping keeps multiple resolution of the same image and, during look-up, uses the resolution that corresponds to the local downscale/upscale factor.

3 Evaluations

The general resampling framework described above allows many practical realizations, in particular when selecting interpolation filters and the anti-aliasing method. In this section we compare several common methods in terms of visual quality, and, more importantly, in terms of computational complexity. Finally, we evaluate memory accesses and design a cache to reduce memory bandwidth for non-linear image domain warping applications.

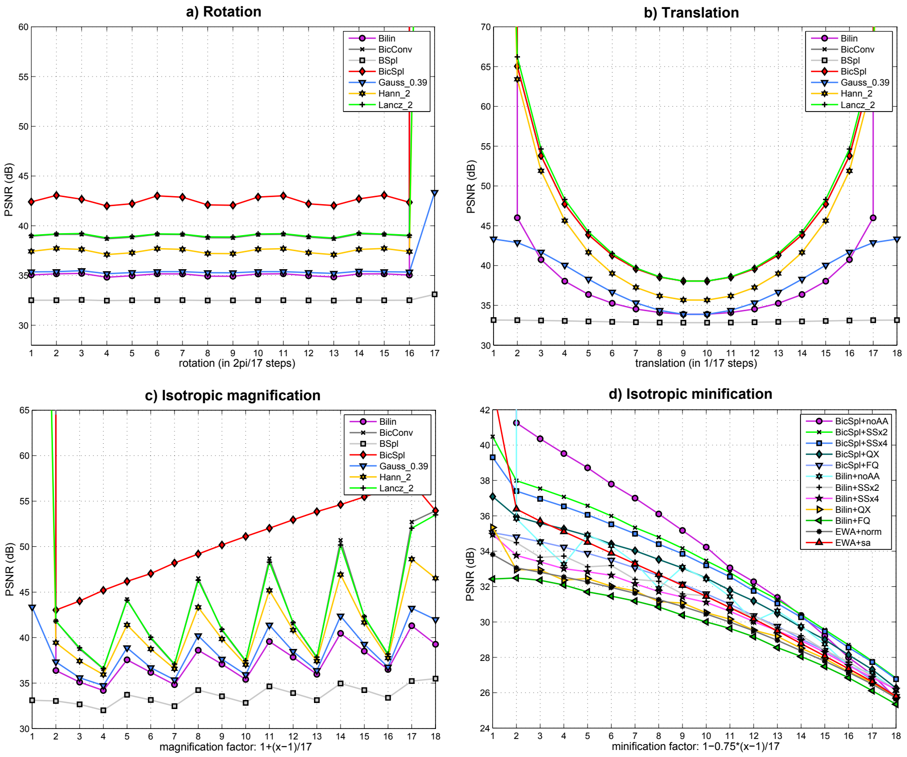


Fig. 2. Comparison of PSNR values for different kind of transformations. The evaluations have been performed by first applying a transformation and then the corresponding inverse transformation. The resulting image is compared with the original using the PSNR metric. The results plotted here are median values over a set of 16 natural 720p color images. The Gaussians are parameterized with $\sigma = 0.39$ in the above evaluation (this value was determined in [9]) and they are clipped at 2.

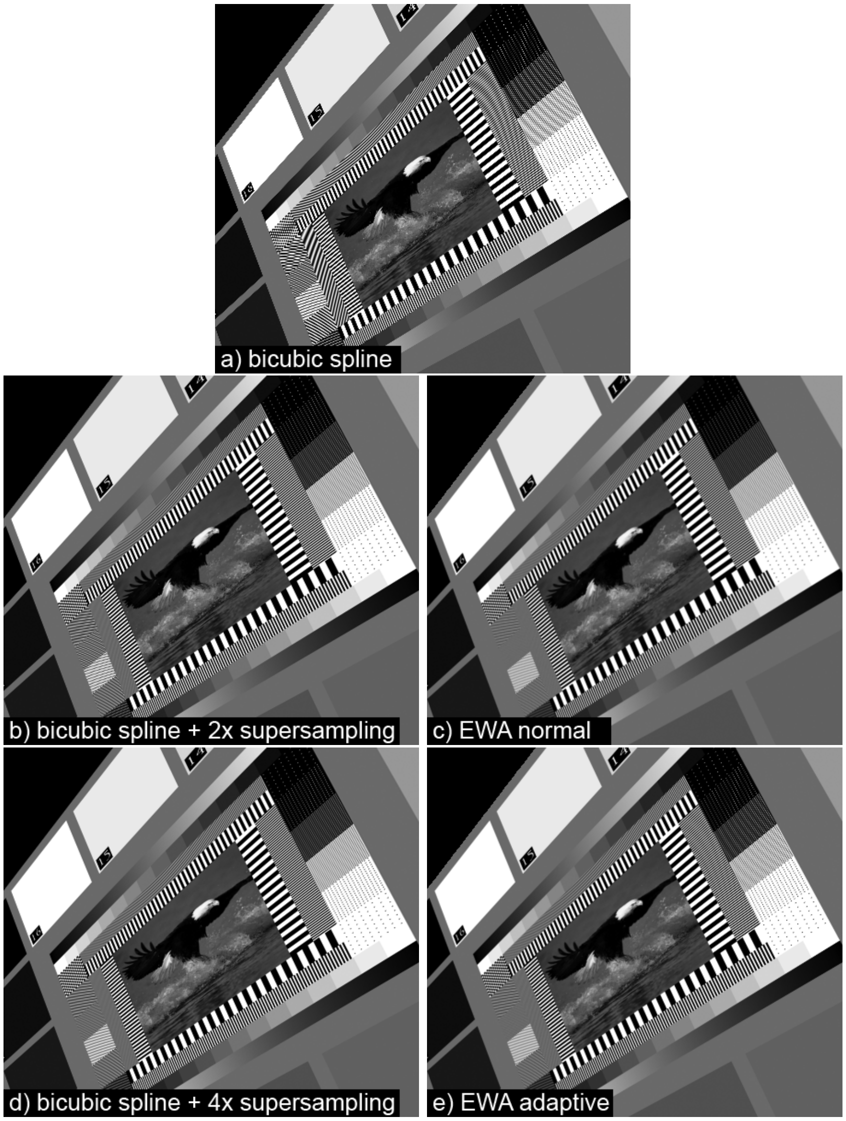

a) bicubic spline
b) bicubic spline + 2x supersampling
c) EWA normal
d) bicubic spline + 4x supersampling
e) EWA adaptive

Fig. 3. Visual comparison of anti-aliasing artifacts: without anti-aliasing visual distortions are obvious (a); twofold oversampling removes some of the aliasing in this example (b). However, fourfold oversampling is necessary to remove all aliasing (d). The two EWA splatting variants (c,e) hardly show aliasing artifacts. We also observe that adaptive EWA splatting (e, see Section 2.2) provides a sharper image than conventional EWA splatting (c).

3.1 Quality Comparisons

For the evaluations we use several well-known interpolation and anti-aliasing methods and apply them to aspect-ratio retargeting and stereo-to-multiview conversion examples. Interpolation kernels compared in this work are bilinear and bicubic interpolation, b-spline, bicubic spline interpolation, Gaussian, and windowed-sinc filters with Hann and Lanczos windows. For details on the different methods refer to e.g. [21]. Anti-aliasing kernels in forward mapping are evaluated using the EWA framework. Backward mapping methods are evaluated with various degrees of constant super-sampling (SS) or different sampling patterns such as quincunx and flipquad [1].

Figure 2 provides evaluation results on comparing interpolation filters and anti-aliasing methods in resampling applications. The different resampling methods are used to transform a set of 16 natural 720p color images according to simple parameterized transformations (e.g. rotation by a certain angle). In order to be able to assess the resampling quality using the PSNR measure, the images are resampled twice: once with the forward transform and once with the corresponding inverse transform. The original images can then be used as a reference. It should be noted that the PSNR is a good measure for the amount of introduced blurring, but it does not capture aliasing artifacts very well as can be seen in Figure 2d, which shows the numerical results for isotropic minification.

From these basic quality evaluation figures, several observations can be made. The first observation is that Gaussian interpolation shows similar quality compared to the typically used bilinear interpolation, even without any non-linear warping. Bicubic methods and windowed sinc filters are in general superior to Gaussian and bilinear interpolation, and B-splines show the worst performance.

Regarding anti-aliasing, one observes two things: the *numerical* evaluations show that all methods introduce a similar amount of additional blurring, except of course, when using no anti-aliasing method. The adaptive Gaussian shows less blurring than the fixed-width Gaussian, which has been already shown in Section 2.2. On top of that, *visual* evaluations show that using no anti-aliasing filter may introduce severe aliasing artifacts, as can be seen in Figure 3.

Our observations are in line with claims and evaluations from previous works [1, 10, 18, 20]. Quality-wise, we conclude that the adaptive EWA splatting approach is superior both to simple bilinear interpolation and bilinear interpolation with super-sampling. In terms of quality, a bicubic or windowed-sinc backward mapping approach with (sufficient) supersampling are superior to EWA splatting.

3.2 Computational Complexity

Beside the quality evaluations, we compare the computational complexity of bilinear and bicubic backward mapping as well as adaptive EWA splatting in image domain warping applications (e.g., aspect ratio retargeting). We consider the following scenarios, first an identity transformation of a 1920×1080 image,

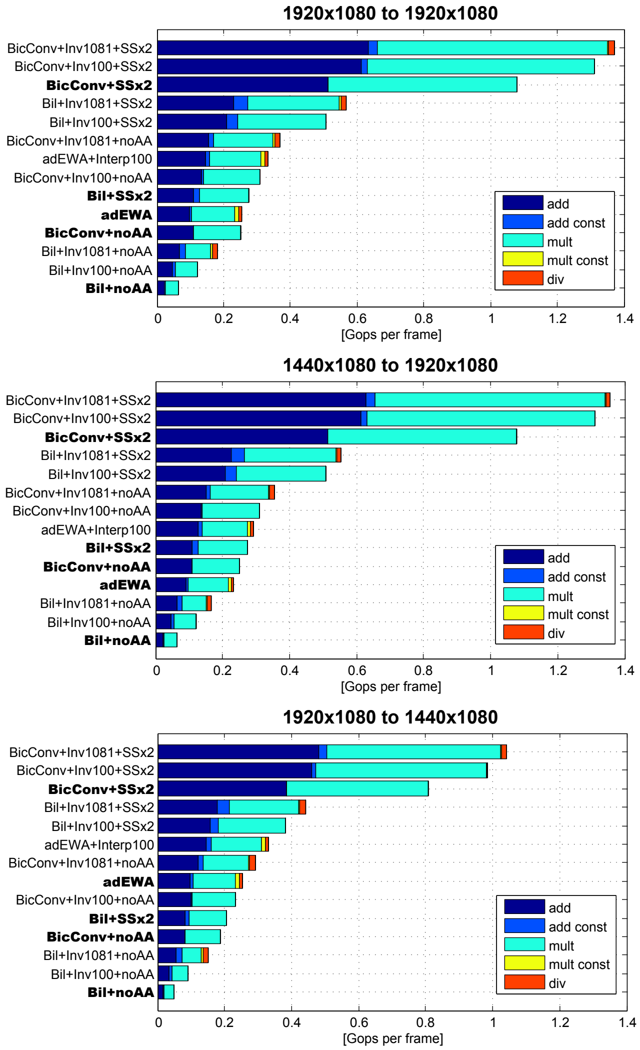


Fig. 4. Comparison of the computational complexity of different resampling techniques. Top: identity transformation; middle: 4:3 to 16:9 retargeting using linear scaling; bottom: 16:9 to 4:3 retargeting using linear scaling. The compared methods are bilinear interpolation (*Bil*), bicubic convolution (*BicConv*) and adaptive EWA splatting (*adEWA*). The postfix *SSx2* stands for twofold supersampling, whereas *noAA* stands for no anti-aliasing. The methods printed with a bold font do not include any warp preprocessing, whereas the methods with the postfix *WarpInv1081*, *WarpInv100* or *Interp100* include an additional warp inversion or warp interpolation step (the number denotes the vertical warp resolution - refer to the text for more details).

second linear scaling from 1440 × 1080 to 1920 × 1080, and third linear scaling from 1920 × 1080 to 1440 × 1080.

Using these three cases, the order of magnitude of the complexity can be estimated. More importantly, these cases allow us to compare the *relative* complexity of different methods. As can be seen in Figure 4, the lowest-cost technique is bilinear interpolation (*Bil+noAA*), followed by bicubic interpolation (*Bic+noAA*) and adaptive EWA (*adEWA*). We also see that supersampling significantly increases the complexity of the backward mapping methods. Note that the bicubic polynomials and the Gaussian kernel evaluations are approximated with linear interpolation between lookup table values. Further, the inverse square-root is calculated using the fast inverse square-root approximation [16].

Warp Interpolation and Inversion The numbers discussed above hide a practical issue: depending on the application, warp information is available in either forward or backward format and the conversion from one format to another requires additional computations. In typical image domain warping applications, warps are available in forward format. This has consequences for backward mapping approaches, where an additional inversion step becomes necessary (denoted as *Inv1081* in Figure 4). Further, the warp is usually available on lower resolution than the actual image such that an additional upscaling step using interpolation is necessary (e.g. bilinear interpolation). Warp inversion together with warp upsampling from 100 pixel to 1080 pixel in vertical direction is denoted as *Inv100* whereas, warp upsampling alone is denoted as *Interp100*. Note that the warp inversion also requires more memory (and associated bandwidth) for the storage of the inverted warp coordinates or the partly rasterized output image.

Conclusion Together with the quality evaluations from the previous section, we conclude that adaptive EWA splatting forms a good tradeoff between computational complexity, interpolation and anti-aliasing quality. Bilinear interpolation lowers the computational complexity but at the price of lower image quality. Finally, bicubic approaches provide more interpolation quality at a similar cost as EWA splatting, but they are very costly when super sampling is added. In image domain warping applications, forward warp coordinates are available, which makes EWA splatting an efficient solution for non-linear warping.

3.3 Memory Bandwidth Evaluation

In real-time VLSI implementations, the warp and image are processed in scanline order. Depending on the warp format, i.e., backward or forward transformation, the image needs to be stored in either input buffer or an output framebuffer, in order to support arbitrary image transformations. The buffers usually contain large parts or the entire image and therefore have to be stored in external off-chip memories (an uncompressed full HD image amounts to about 50 Mbit). Due to the limited pin and power budget it is important to minimize the required memory bandwidth.

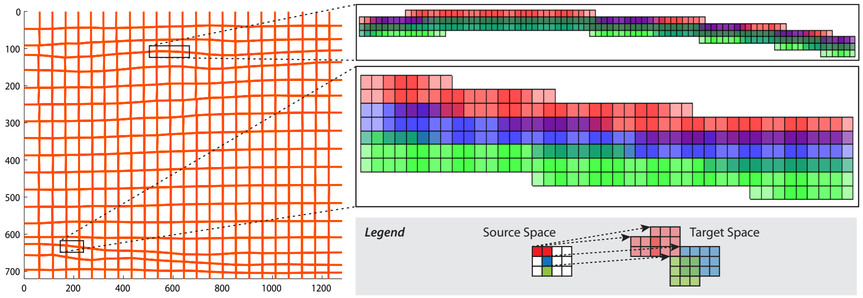
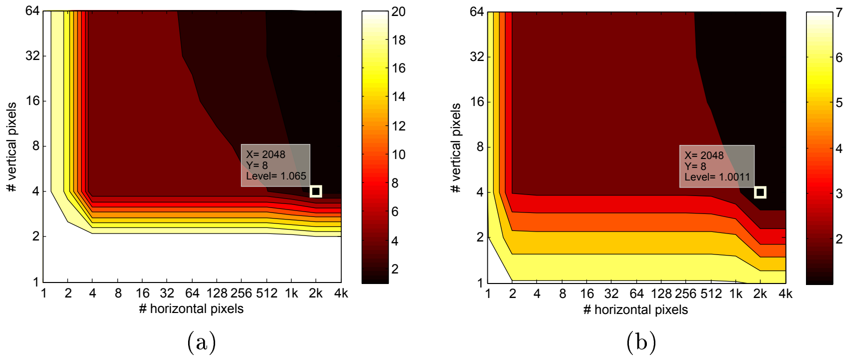


Fig. 5. This figure shows a typical frame buffer access pattern of a retargeting warp rendered with EWA splatting. The grid on the left hand side shows a typical retargeting warp of a 1280 × 720 image. In this application, the warp contains only limited image distortions. The excerpts on the right side show close-ups of two partially rendered regions of this warp, where the large spatial overlap among subsequent patches in horizontal and vertical directions can be observed.

Bandwidth Bottleneck In the case of backward mapping, each rendered output image pixel requires a small patch to be fetched from the input image in order to perform the interpolation. Similarly, in the case of a forward mapping method, each input image pixel leads to a patch of pixel contributions in the output image, which are then accumulated in a frame buffer. In both cases the size of those patches depends on the employed filter kernel, and ranges from 2×2 (bilinear and adaptive EWA), to 3×3 (normal EWA) to 4×4 pixels (bicubic).

Thus, the amount of memory accesses is a multiple of the image resolution and can easily lead to a bandwidth bottleneck. For instance, in the case of bicubic interpolation and a 1080p color output image we require a bandwidth of $1920 \times 1080 \times 3 \times 4 \times 4 \approx 796$ MByte/frame. Fortunately, if the warp is traversed in scanline order, consecutively accessed patches spatially overlap to a certain degree as illustrated in Figure 5. The vertical and horizontal *locality* of the patches can be leveraged to reduce the overall bandwidth by employing a *two-dimensional* image cache (similar to texture caches in GPUs).

Cache Evaluations In order to see how a two-dimensional image cache has to be parameterized in the case of image warping, we simulated different cache configurations for different resampling methods and calculated the required bandwidth for one frame. Figure 6 shows the averaged simulation results of a direct mapped cache with degenerate 1×1 blocks for 17 nonlinear retargeting warps (aspect ratio change from 4:3 to 16:9 for 1080p images). Note that the retargeting warps all show similar memory access patterns as they perform the same global linear upscaling with only local variations. For better readability, the bandwidth has been normalized with the size of the input image (a), or with the size of the output image (b) – depending on the mapping direction (forward or backward). We can see that as soon as the cache is large enough to store an image patch



(a)

(b)

Fig. 6. Simulation of different overall cache sizes of a direct mapped cache with degenerate 1 × 1 blocks (i.e. each pixel has its tag and valid bit). Both plots show the averaged results from 17 nonlinear retargeting warps (aspect ratio change from 4:3 to 16:9 of 1080p images).(a) shows the normalized bandwidth of the input image buffer for bicubic backward mapping; and (b) shows the normalized bandwidth of the frame buffer for (non-adaptive) EWA splatting. The normalized bandwidth is encoded in the color. Depending on the mapping direction of the method, the normalization factor is either the size if the input (a) or output image (b).

which is larger than the size of the employed filter kernel (4×4 in (a) and 3×3 in (b)), the bandwidth is beginning to drop significantly. In the ideal case, if the cache is large enough, data will be transferred only once between the main memory and the cache. This corresponds to transferring only one input image to the cache in a backward mapping architecture, and only one output image in a forward mapping architecture. In this optimal case the normalized bandwidth is equal to one. As can be seen in Figure 7 and Figure 8, the optimum can be reached if the cache is large enough to hold several image rows.

A cache with a block size of 1×1 pixel is adapts well to the geometric variations in the warp function, but such a cache configuration requires a huge amount of overhead (i.e., valid-bit and address-tag entries have to be stored for each pixel as well). Increasing the block size reduces the memory required to store the tags and the valid-bits, but also comes at the cost of cache performance, as can be seen in Figure 7 and Figure 8. However, if the cache is large enough, it is possible to reach the optimum with large cache blocks.

4 Hardware Architectures

Based on the findings from the previous section, we introduce two hardware architecture for spatially-varying image warping. The first architecture uses forward mapping with adaptive EWA, the second architecture uses backward mapping with bicubic interpolation and, to reduce computations, with an *adaptive super-sampling* technique.

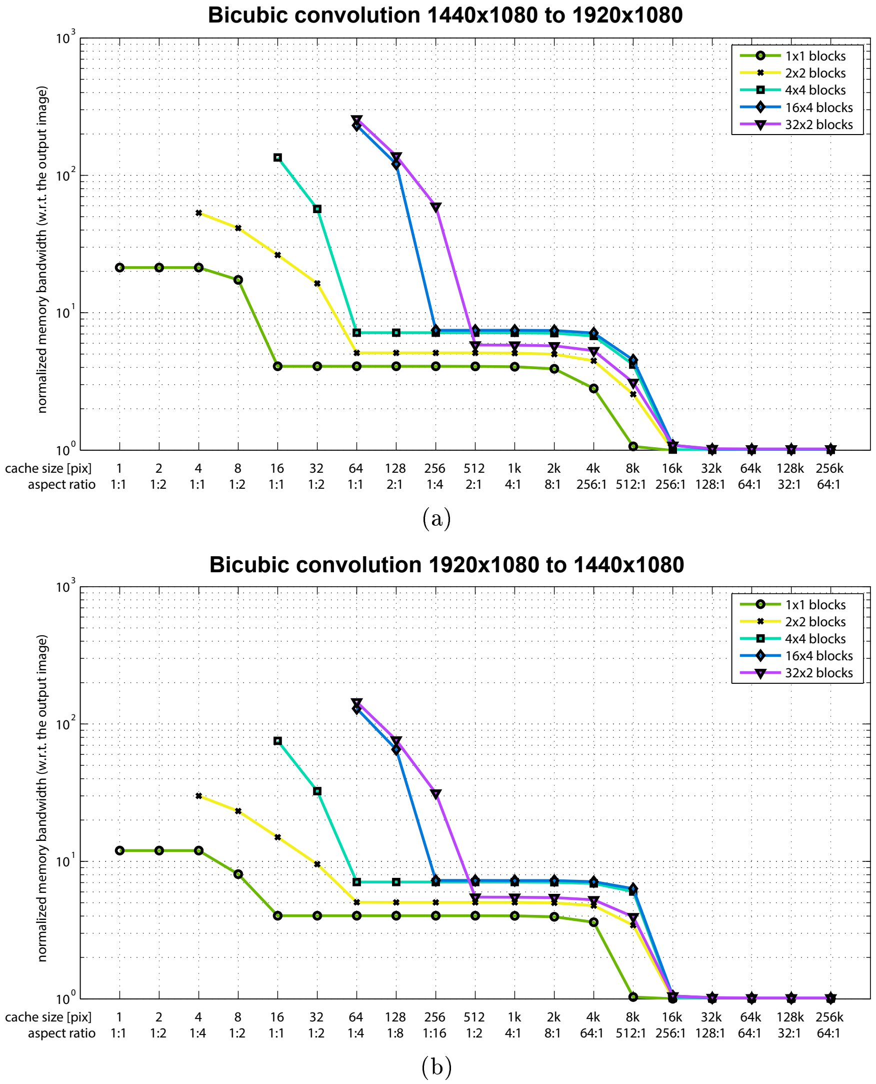
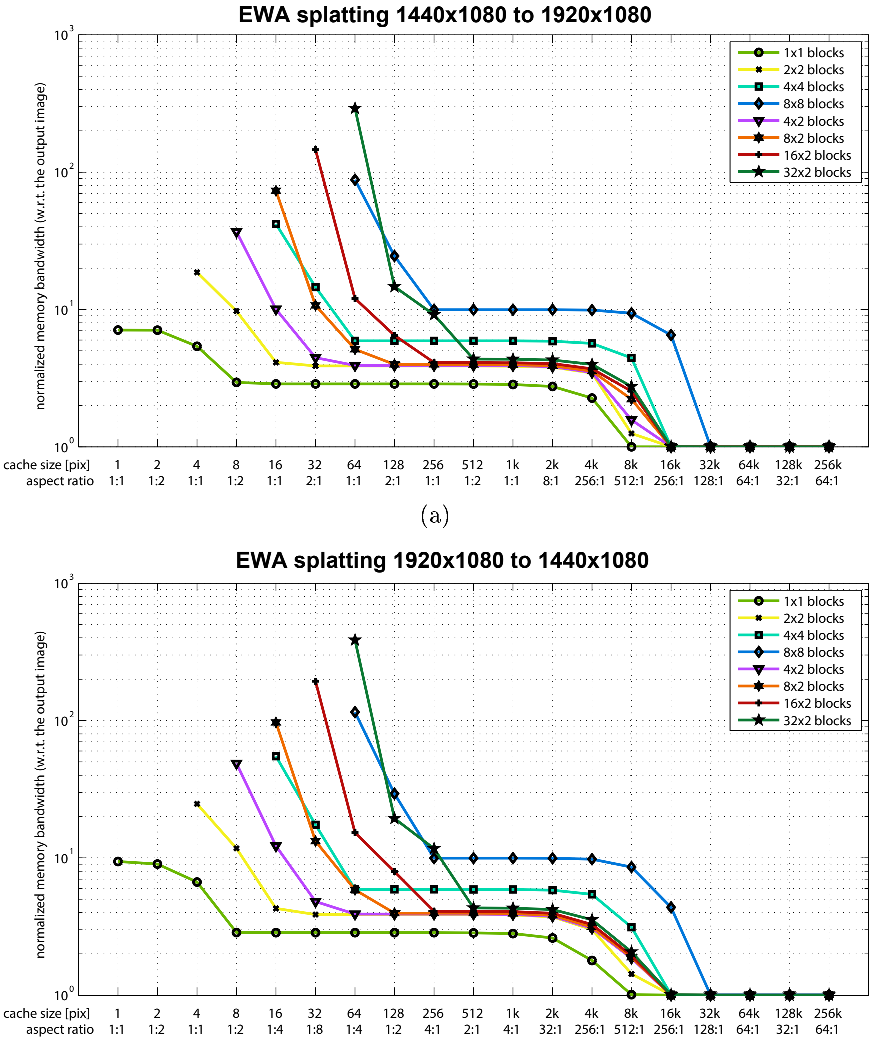


Fig. 7. Simulation of different cache configurations for the input image buffer used in bicubic backward mapping. (a) shows the results for a nonlinear aspect ratio change from 4:3 to 16:9 of a 1080p image; and (b) shows the results for a nonlinear aspect ratio change from 16:9 to 4:3 of a 1080p image. The results are mean values over 17 warps generated for natural testimages.

4.1 EWA Splatting Architecture

The top-level diagram of the EWA architecture is given in Figure 9. The core accepts streaming pixel color information, given in an 24 bit RGB format. In addition to the color information, a deformation grid describing the pixel mapping



(a)

(b)

Fig. 8. Simulation of different cache configurations for the frame buffer used in EWA splatting. (a) shows the results for a nonlinear aspect ratio change from 4:3 to 16:9 of a 1080p image; and (b) shows the results for a nonlinear aspect ratio change from 16:9 to 4:3 of a 1080p image. The results are mean values over 17 warps generated for natural testimages.

m is streamed in. In the quadrilateral deformation grid format, the deformation of each pixel is described by transformation of the pixel's bounding box. More specifically, the four corner positions of a *quad* describe the new pixel center as well as the pixel deformation. The horizontal and vertical gradients necessary for constructing J_k can be easily deduced from the quads.

We assume that the image transformation m is locally smooth, and that neighboring pixels share their adjacent quad grid corners. Therefore, in compact form, the quad grid representation only requires $(W+1) \times (H+1)$ grid points, if W and H are the input video width and height, respectively. Note that we chose this representation in order to disallow transformations that would result in image holes. Furthermore, since neighboring grid points and pixels are typically strongly correlated, we add a lossless differential compression/decompression scheme at the input interface to reduce the input bandwidth and I/O power. Note that temporal compression across frames could further reduce the input bandwidth since the warp typically varies slowly over time.

From the input quad grid, the pixel position $m(\mathbf{u}_k)$ (mean of adjacent corner positions) and the Jacobi matrix J_k (horizontal and vertical gradients computed from the corner positions) are calculated and stored in an on-chip FIFO buffer. A dispatcher unit then distributes positions, Jacobian, and pixel values to multiple arithmetic units that perform the splatting operation. The processing time of each splatting operation strongly depends on the deformation, as one input pixel can possibly be stretched to multiple output pixels. To handle the variable throughput requirements, several arithmetic splatting chains are used in parallel, and the dispatcher unit distributes the input pixels depending on the workload. A FIFO buffer can absorb incoming pixels when all splatting units are occupied during performance peaks. To handle prolonged peaks, the FIFO fires a back-pressure system that allows to stall the data source to avoid data loss.

Due to mathematical properties of the summation operation in (5), we can rearrange the operation: instead of evaluating the sum for each output position, we forward-transform all individual Gaussian kernels and perform an accumulation of the Gaussian *contributions* in the temporary output image. To avoid excessive memory bandwidth requirements between the chip and the external frame buffer, we employ a two-level cache structure, in accordance with the cache configuration simulations presented in the previous section. The cache exploits the spatial coherence of image transformations, which in general map neighboring input pixels to neighboring output pixels.

When all input pixels have been processed, the temporary output image can be streamed to a normalization unit, where the accumulated pixels are then normalized by the sum of the filter weights. Note that this final normalization step is necessary due to the fact that Gaussian filters, and in particular their truncations, are non-ideal interpolation filters.

Input Interface The system requires the pixel information and the deformation grid as the input. Since the deformation grid has to be determined by another computation block, a simple custom interface has been designed that can easily be adapted for different applications.

Compression The input interface consists of 24 bit RGB values and 2×24 bit pixel coordinates, resulting in a bandwidth of 4.5 GBit/s for 1080p30. To

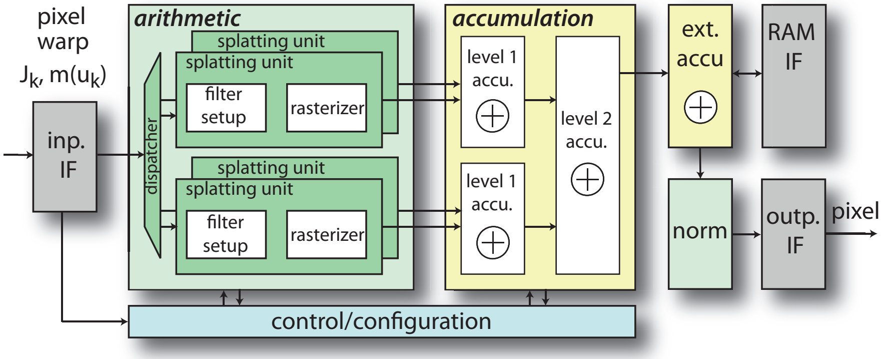


Fig. 9. Top level block diagram of the EWA rendering architecture.

reduce the input bandwidth, we propose an optional differential compression scheme. The compression exploits the fact that neighboring pixel colors and coordinates usually exhibit strong spatial correlation, and will therefore result in small incremental changes only. The purpose of the compression is to transmit the small incremental changes only. The input fixed-point words are decomposed into several sub-words, i.e., an n bit word is decomposed into n/m m bit words. This decomposition relies on the observation, that the upper bits (MSBs) of pixels and pixel positions change very rarely compared to the lower bits (LSBs). With this, as only sub-words that change are transmitted, one MSB sub-word can be transmitted with several LSB sub-words plus control bits that indicate the number of lower sub-blocks per upper sub-block. Evaluation on actual data has shown a bandwidth reduction of 35% on average. Note that the compression is completely lossless and comes at negligible hardware overhead.

Dispatcher The dispatcher unit is responsible for load-balancing between multiple subsequent splatting units. A simple round-robin based priority scheme is used for the scheduling.

Arithmetic Processing Elements In a nutshell, the splatting units implement the EWA equation (5) in a fixed-point format. For each input pixel, a Gaussian kernel is calculated from the pixel color w_k and the linearized approximation J_k of the warp grid. The Gaussian kernel is then resampled to determine its contribution to all output pixels. The resampling is evaluated within a small bounding box of the Gaussian only, i.e., the Gaussian will be truncated to zero as soon as its energy falls below a very small threshold. The contributions of the individual Gaussians are then accumulated, and finally normalized.

The datapath is implemented using custom fixed-point arithmetic. The accumulated color channels are calculated with 11 bits each, and the accumulation values are calculated with 12 bits, resulting in data words of 45 bits in total for each pixel. This number has been chosen both for accuracy reasons as well as to match the word-width of the external memory.

Adaptive EWA The splatting cores can be configured to work in 'adaptive' mode, which means that the Gaussian resampling covariance matrix is adapted per-pixel to reduce the amount of blurring. The adaptive mode has been introduced in [9] and its impact on overall area is negligible.

Throughput Each splatting unit has a fixed throughput $\Theta = f/n_{\text{cycles}}$, determined by the clock frequency f and the number of cycles required to evaluate one input pixel n_{cycles}. The current architecture is optimized for $n_{\text{cycles}} = 20$, which is matched to the average number of output pixels times the number of cycles it takes to evaluate one output pixel (9×2 plus overhead). A throughput of 9 MPixels/s per splatting unit at a clock frequency of 170 MHz can be achieved. Therefore, 1080p30 video (63 MPixels/s) can be achieved with some margin by employing 8 parallel splatting units.

Accumulation, Caching, and Memory Interface Each input pixel produces several output contributions that need to be weighted by a Gaussian kernel and accumulated at the output sampling locations. As described earlier, the Gaussian kernels are truncated at the bounding box boundary, and simulations have shown that bounding boxes of 4 to 9 pixels in size are sufficient to capture the majority of the non-zero contributions. Hence, the accumulation bandwidth is between 2×4 and 2×9 times larger than the input bandwidth, as each accumulation is performed using a read-modify-write operation. To reduce the external bandwidth, our on-chip caching architecture exploits the horizontal and vertical overlaps of neighboring Gaussian kernels. In a first stage (denoted L1), contributions with spatial proximity are collected and accumulated into larger blocks. The L1 blocks are then efficiently accumulated to a second stage (L2 blocks). The L2 cache is able to store several lines of the image, and once a line is removed from the L2 cache it is accumulated to the external frame buffer memory. Our two-stage caching architecture reduces the resulting bandwidth considerably: the L1 cache is implemented using register arrays that support the highest bandwidth, and the L2 cache implemented using block RAMs that reduce the bandwidth to external memory further.

Throughput Each accumulated data word has 45 bits, the required bandwidth for 1080p30 can be calculated as

$$bw_{\text{full}} = 45 \times 1920 \times 1080 \times 30 \times 2 \times (1 + n_{\text{pps}}),$$

where the factor 2 comes from the read-modify-write operation. n_{pps} denotes the number of pixels per splat, i.e., the bounding box size. Additionally, the final read out requires one more read from the memory. If we assume a conservative value of $n_{\text{pps}} = 9$, the overall bandwidth equals $bw_{\text{full}} = 56\,\text{Gbit/s}$. Our cache architecture exploits the inherent spatial overlap between neighboring pixels,

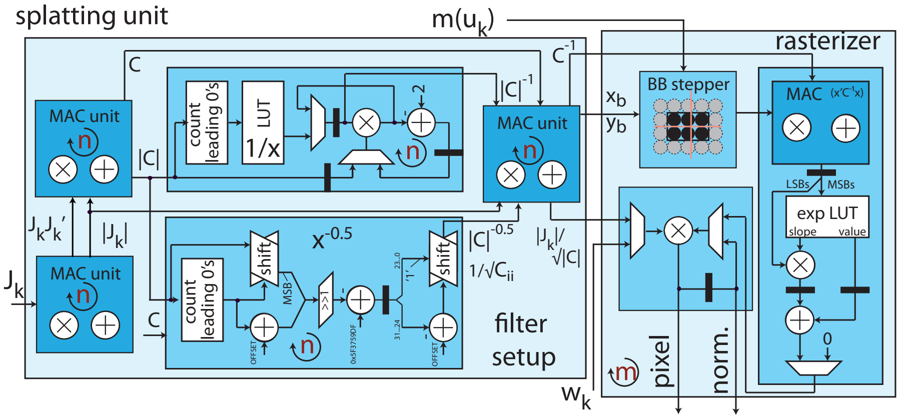


Fig. 10. Data path of the EWA splatting core.

and shifts the bandwidth burden to the on-chip buffers, reducing the effective n_{pps}. In simulations, a cache efficiency resulting in $n_{\mathrm{pps}} = 3$ is always achieved, and the required bandwidth is reduced to 22.4 Gbit/s.

Due to the read-modify-write operation, we choose a QDR-type memory interface to efficiently support the accumulation. QDR memories are static RAMs that have separate read and write ports, which can be accessed in parallel. Moreover, the data is transmitted in double edge mode. A 9 bit QDR RAM port therefore has 3 Gbit/s read and 3 Gbit/s write bandwidth, at a clock frequency of 170 MHz. Our architecture employs 5 instances of such 9 bit RAM interfaces, and the resulting 45 bit memory interface matches our data word size. The overall available bandwidth therefore amounts to 30 Gbit/s.

Scheduling and Control Flow Due to the varying bounding box sizes of the input Gaussian kernels, the run-time of the individual rendering cores is non-deterministic during operation. However, on a per-frame basis the varying per-pixel run-times are averaged out and thus approximately constant, which can be used for dimensioning the number of cores and the required memory bandwidth. Short-term fluctuations of throughput are then regulated using a back-pressure system. Moreover, an efficient scheduling strategy distributes the input pixels to individual rendering units.

Output Interface The final step of the rendering pipeline consists of reading out the image from the frame buffer and interfacing it to a standard display chip. Since display interfaces must adhere to a very strict timing, the read-out from the frame buffer is always prioritized over the read-modify-write accumulation operations. In case of collision, the accumulation can be stalled via the back-pressure mentioned before. The normalization block contains a divider producing

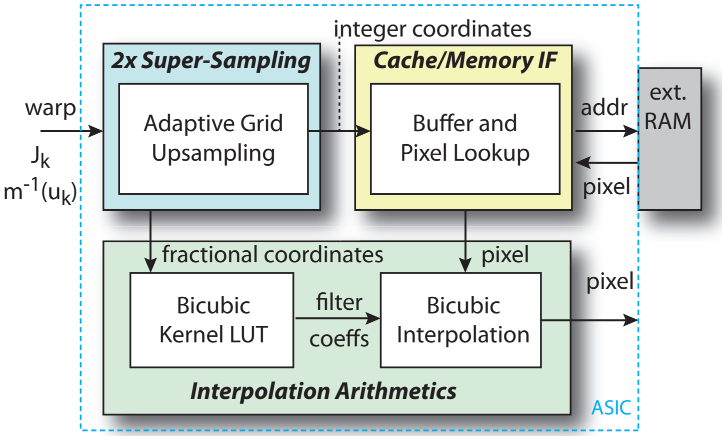


Fig. 11. Block diagram of bicubic interpolation with adaptive 2× super-sampling. The caching setup is similar to the EWA rendering architecture cache, since a similar access pattern is assumed. The key difference is that the bicubic backward mapping cache is a read-buffer whereas the EWA forward mapping cache uses (read-modify)-write accesses.

the final 24 bit RGB values, by normalizing the accumulated RGB values with their weights.

4.2 Bicubic Warping Architecture

Figure 11 provides a top-level architecture overview of backward mapping using bicubic interpolation and two-times adaptive super-sampling. The high-level architecture is conceptually similar to the EWA architecture shown in Figure 9. The backward warping grid represented by Jacobian and backward coordinate lookup values is streamed line-by-line into the warping core (no warp inversion is performed here). However, contrary to the EWA forward architecture, the pixel intensities are not streamed together with the warp grid, but are accessed through an external buffer.

Adaptive Supersampling The adaptive super-sampling block decides for each warp input whether super-sampling is necessary or not, by detecting if the transformation is locally minifying. Minifications are detected by checking if the determinant of the Jacobian is greater than one, or the determinant of the inverse Jacobian is larger than one, depending on which format is available at the input. In the case of supersampling, the locations of the additional sampling points are linearly approximated using the Jacobian and the lookup coordinate. Note that applying super-sampling in an adaptive way has two benefits over applying super-sampling everywhere: first, the amount of computations is significantly reduced, and second, blurring due to super-sampling with non-ideal decimation kernels is avoided where no anti-aliasing is necessary.

The currently employed supersampling strategy could be further improved by introducing directional super sampling, i.e., by applying super sampling only in the direction where the potential aliasing appears. This extension is able to provide a higher throughput in cases where the image transformation is demagnifying and anisotropic (e.g. an aspect ratio change from 16:9 to 4:3).

Bicubic Interpolation The adaptive super-sampling block outputs backward coordinates to the memory interface block responsible for fetching the corresponding pixels from the external buffer. In parallel, the interpolation arithmetics block gets the fractional part of the backward coordinates to set up the bicubic filter kernel coefficients. For each output sample, an area of 4×4 pixels has to be multiplied with the bicubic kernel. Note that this kernel is separable and thus can be implemented with four vertical- and one horizontal application of the one-dimensional bicubic convolution kernel. This architecture uses fixed point arithmetic throughout. The coefficients of the bicubic kernel can therefore be directly obtained from two lookup tables (LUTs), where we the indices consist of two integer bits and the fractional bits of the x and y coordinate differences, respectively. If the fractional precision is not too large, no additional refinement of the indexed values (e.g. using linear interpolation) has to be performed as the LUTs already cover the whole index range. The throughput of the bicubic interpolator is one pixel per 4 cycles. In order to produce a super-sampled output pixel, four such samples have to be calculated and averaged and thus 16 cycles are required in that case. Thus, the effective throughput of the interpolator depends on the fraction of supersampled pixels per frame.

Caching and Memory Interface Analogue to the forward mapping architecture, the amount of accessed input image pixels is very large as the bicubic interpolator always accesses a 4×4 neighbourhood in order to calculate an output pixel. Thus even in the case of no supersampling, we would have to load 16 times more pixels than there are in one frame, which translates into a very large external bandwidth. An on-chip read cache which is able to hold several lines of the image is therefore employed to reduce the bandwidth. As the bicubic interpolator requires 4 cycles to produce one output sample, the cache memory is divided into four column interleaved RAM macros such that four parallel accesses are possible.

5 Implementation Results

Several ASIC implementations of non-linear warping architectures have been realized which allow for a comparison and conclusion on hardware performance of the different techniques. In the following, we discuss some of the implementation results of the different ASICs.

5.1 EWA Splatting: Esper

The EWA architecture described previously was implemented in VHDL and was fabricated in 180 nm (1P6M) CMOS technology (Figure 12(a)). The design supports image resolutions up to 2048×2048 and works on gray-level 8-bit pixels. It employs four splatting units to support 720p25 in splatting performance. Due to die size limitations the cache is reduced to eight lines of gray-valued 576p (i.e. 8x1024 pixels). The ASIC has been successfully tested at 123 MHz where a power consumption of 300 mW has been measured. Core voltage is 1.8 V and I/O pad voltage is 3.3 V. Core area is 6 mm^2 which corresponds to 660 kGE. There are 64 data I/O pins and 56 power/ground pins. This chip is a prototype of the EWA core architecture and does not possess a real-world memory interface. The normalization block is also not included. The accumulation precision is set very conservatively to 16 bit per entry.

Detailed Throughput Figures For the following throughput figures, a nominal clock frequency of 133 MHz is assumed. One splatting unit has a throughput of 6.65 MPixels/s. A 720p25 video stream requires a throughput of 23 MPixels/s and thus four splatting units. The necessary external memory bandwidth without caching is $2 \times 9 \times 23 = 414$ MPixel/s which amounts to $414 \times 4 \approx 1.66$ GByte/s for 4 bytes per pixel entry (accumulated value plus normalization weight). The factor 2 comes from the read-modify-write operation of the accumulation. The cache has an efficiency of about 83% which reduces the external bandwidth to 282 MByte/s, i.e. the normalized bandwidth is around 1.5 (normalized to the bandwidth it takes to read, modify and write one output image). The optimum normalized bandwidth cannot be reached, as the cache is only 1024 pixels wide. In order to reach the optimal cache efficiency for 720p25 video, the cache should be extended to 8 lines with 1280 pixels per line (see Section 3.3). Note that in addition to the above bandwidth, a read/clear operation to the memory is further necessary to account for the final read-out and clearing.

5.2 EWA Splatting: Vesper

The Vesper chip is an extended version of Esper, and it is designed to render full HD color images at 30 frames per second. In contrast to Esper this chip has been fabricated in 130 nm CMOS and it is equipped with fully-functional display- and memory interfaces (Figure 12(c)).

The DVI interface requires a fixed input bandwidth and clock frequency, which usually is dependent on the display resolution and frame rate. To decouple the arithmetics and accumulation from the DVI interface, we separate the design into two clock domains. While the rendering core should run as fast as possible, the DVI core is running at the specific DVI pixel clock. The asynchronous data interface is implemented using an asynchronous FIFO, see [4]. The chip also provides clock signals for the external RAM components and the DVI transmitter. In order to provide a flexible timing at the corresponding interfaces, those output clocks can be phase shifted relative to the internal clock signals.

The I/O bandwidth, the throughput of the splatting units and the caching have been dimensioned for very pessimistic and demanding scenarios, such that 1080p30 performance is achieved in a practical system that supports a wide variety of warping applications. In turn, the actual performance for typical applications will be higher, and therefore also higher frame rates are possible. Under typical conditions, our architecture reaches 1080p48. Furthermore, smaller resolutions are always possible and would increase the frame rate further (e.g. 720p60).

The chip area is largely dominated by the number of input and output pins, as well as the required power distribution for the high speed I/O interfaces. VESPER supports 175 data I/O pins, of which 115 pins are used for the external QDR-II interface. Due to the prototype nature of the chip, a more conventional "around the core" I/O has been employed, instead of a more area efficient flip-chip I/O. For a commercial implementation, the area could be significantly reduced since the overall logic area (including on-chip SRAM) is 9 mm^2 which is much smaller than the $5 \times 5\,\text{mm}^2$ the chip currently occupies.

5.3 Bicubic Interpolation: EVA

A bicubic backward mapping ASIC named EVA has been fabricated in 180 nm CMOS technology (Figure 12(b)), and it has been designed to roughly match the specifications of the ESPER ASIC such that the two implementations are comparable. It is able to support enough throughput for 576p25 when 10% of the calculated pixels are super-sampled. Typical case post-layout simulations have shown a maximum operating frequency of 135 MHz and a power consumption of 60 mW. The core area is around 1 mm^2 which correspond to 110 kGE. The chip contains a similar cache configuration as ESPER (8×1024 pixel) with approximately half of the access bandwidth: backward mapping only requires a read operation whereas the accumulation process in forward mapping requires a read-modify-write operation. Thus, a single port memory is sufficient which reduces SRAM size by half. Further, the entries are only 8 bit wide, and no accumulation weights have to be stored. Thus, in total, the SRAM memory macro is about four times smaller in size. Note however, that this factor of four reduces to a factor of around 1.83 if RGB pixels are stored (EWA shares the weight entry among the three color subpixels), and if the accumulation precision is changed to a more realistic value of 11 bit.

5.4 Comparison

In Table 1, we list several warping VLSI architectures to compare computational resources. As can be seen, the EWA warping core is significantly larger than comparable backward architectures such as EVA and [12,15]. In particular, the cache of the EWA implementations is larger due to the higher fixed-point precision, the additional weight entry, and the read-modify-write type operation. Another reason for the higher gate count of the EWA implementations is that the EWA arithmetics are fully evaluated, whereas a significant portion of computations in

Table 1. Comparison of different warping CMOS implementations. The rather large differences in size originate from the type of transformation that are supported: arbitrary (arb.) or simple linear scaling (lin. scal.). In addition, EWA and bicubic feature full or partial (super sampling (SS)) anti-aliasing support. Note that for the EWA architecture, the interpolation (interp.), anti-aliasing (AA), and transformation (transf.) blocks cannot be separated for the area numbers. A '–' means that the architecture does not contain such a block, N/A means that the values are not available. The external bandwidth figures are typical case values, normalized to reading/writing an image once. The maximum throughput of our implementations are evaluated valid for non-linear resizing (change of aspect ratio from 4:3 to 16:9). Results marked with (*) are obtained through postlayout simulations.

	EWA	Bicubic	Ext. Bil.	Ext. Bil.
	Esper	Eva	[15]	[12]
Anti-Aliasing	yes	2×SS	no	no
Transformation	arb.	arb.	lin. scal.	lin. scal.
Mapping direction	fwd.	bwd.	bwd.	bwd.
Image resolution	576p	576p	1080p	WQSXGA
Color channels	1	1	1	1
Technology [nm]	180	180	130	130
Max. clock freq.[MHz]	123	135*	267	278
Max. throughput [fps]	40	47 (0%SS)* 28 (10%SS)* 12.5 (100%SS)*	N/A	30
Power [mW]	300	62*	18.1	11.7
Transformation [kGE]	≈ 115	6	N/A	N/A
Interpolation [kGE]	≈ 115	10	N/A	N/A
Total w.o. memory [kGE]	230	16	26	13
Buffer size [Bit]	$1024 \times 8 \times 2 \times 16$	$1024 \times 8 \times 8$	–	4 lines
Buffer [kGE]	410	90	–	N/A
Memory type	DP SRAM	SP SRAM	–	SRAM
Normalized ext. BW	1	1	4	N/A

the bicubic implementation are optimized by using LUT-based approximations. Note that the EWA architecture could also be optimized by replacing some of the arithmetic units with look-up tables. Thus, in applications where warping information is available in backward format and aliasing is of limited concern, one should clearly prefer a backward mapping architecture, e.g. bicubic or bilinear interpolation.

6 Conclusions

Arbitrary transformations of high-definition videos can be efficiently rendered using a non-linear warping VLSI architecture. The proposed VLSI cores can be used in an end-user device and enable image warping for current and upcoming content-adaptive applications. Due to the separation of the rendering core into several sub-units, the computational capabilities are easily scalable to higher

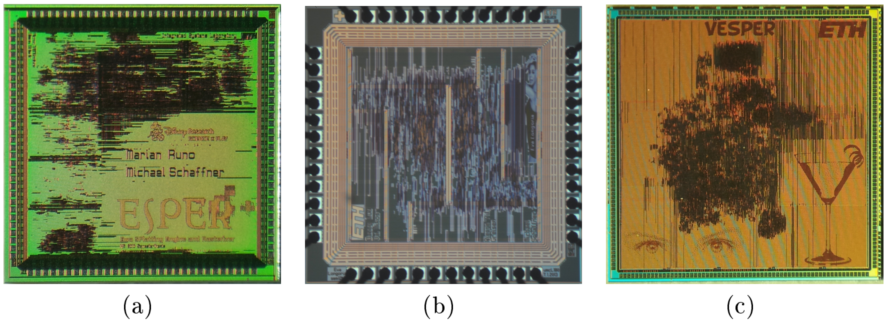


Fig. 12. Photo and CAD rendering of our VLSI implementations: (a) Esper (180 nm), (b) Eva (180 nm) and (c) Vesper (130 nm). The pictures are not to scale.

resolutions and frame-rates, such as the upcoming quad HD standards (2160p) or the high-frame rate (HFR) standards.

References

1. Akenine-Moller, T., Haines, E., Hoffman, N.: Real-time rendering. AK Peters (2008)
2. Asari, K.V.: Design of an efficient vlsi architecture for non-linear spatial warping of wide-angle camera images. Journal of Systems Architecture 50(12), 743 – 755 (2004), http://www.sciencedirect.com/science/article/pii/S1383762104000682
3. Chang, F.J., Tseng, Y.C., Chang, T.S.: A 94fps view synthesis engine for HD1080p video. In: Visual Communications and Image Processing (VCIP), 2011 IEEE. pp. 1 –4 (November 2011)
4. Cummings, C.: Simulation and synthesis techniques for asynchronous fifo design. In: SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers (2002)
5. Do, M., Nguyen, Q., Nguyen, H., Kubacki, D., Patel, S.: Immersive visual communicatio n. Signal Processing Magazine, IEEE 28(1), 58 –66 (jan 2011)
6. Farre, M., Wang, O., Lang, M., Stefanoski, N., Hornung, A., Smolic, A.: Automatic content creation for multiview autostereoscopic displays using image domain warping. In: Multimedia and Expo (ICME), 2011 IEEE International Conference on. pp. 1–6. IEEE (2011)
7. Greisen, P., Heinzle, S., Gross, M., Burg, A.: An FPGA-based processing pipeline for high-definition stereo video. EURASIP Journal on Image and Video Processing 2011(1), 18 (2011)
8. Greisen, P., Emler, R., Schaffner, M., Heinzle, S., Gurkaynak, F.: A general-transformation EWA view rendering engine for 1080p video in 130 nm CMOS. In: VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on. pp. 105 –110 (oct 2012)
9. 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. Transactions on Circuits and Systems for Video Technology accepted (2012)

10. Heckbert, P.: Fundamentals of Texture Mapping and Image Warping. Masters thesis, Univ. of California, Berkeley, Dept. of Electrical Eng. and Computer Science (1989)
11. Horng, Y.R., Tseng, Y.C., Chang, T.S.: VLSI architecture for real-time HD 1080p view synthesis engine. IEEE transactions on circuits and systems for video technology 21(9), 1329–1340 (September 2011)
12. Huang, C.C., Chen, P.Y., Ma, C.H.: A novel interpolation chip for real-time multimedia applications. Circuits and Systems for Video Technology, IEEE Transactions on 22(10), 1512–1525 (oct 2012)
13. Krähenbühl, P., Lang, M., Hornung, A., Gross, M.: A system for retargeting of streaming video. ACM Transactions on Graphics (TOG) 28(5), 1–10 (2009)
14. Lang, M., Hornung, A., Wang, O., Poulakos, S., Smolic, A., Gross, M.: Nonlinear disparity mapping for stereoscopic 3D. ACM Trans. on Graphics (Proc. SIGGRAPH) 29(3) (2010)
15. Lin, C., Sheu, M., Chiang, H., Wu, Z., Tu, J., Chen, C.: A low-cost VLSI design of extended linear interpolation for real time digital image processing. In: Embedded Software and Systems, 2008. ICESS'08. International Conference on. pp. 196–202. IEEE (2008)
16. Lomont, C.: Fast inverse square root. Tech. rep., Purdue University (2003), http://www.lomont.org/Math/Papers/2003/InvSqrt.pdf
17. Oh, S., Kim, G.: Fpga-based fast image warping with data-parallelization schemes. Consumer Electronics, IEEE Transactions on 54(4), 2053 –2059 (november 2008)
18. Szeliski, R., Winder, S., Uyttendaele, M.: High-quality multi-pass image resampling. Tech. rep., Microsoft Research (2010)
19. Tanimoto, M., Tehrani, M., Fujii, T., Yendo, T.: Free-viewpoint tv. Signal Processing Magazine, IEEE 28(1), 67 –76 (jan 2011)
20. Triggs, B.: Empirical filter estimation for subpixel interpolation and matching. In: International Conference on Computer Vision (ICCV). vol. 2, pp. 550–557 (2001)
21. Wolberg, G.: Digital image warping, vol. 3. IEEE Computer Society press (1990)
22. Zwicker, M., Pfister, H., Baar, J.V., Gross, M.: EWA splatting. IEEE Transactions on Visualization and Computer Graphics 8(3), 223–238 (2002)