A General-Transformation EWA View Rendering Engine for 1080p Video in 130 nm CMOS

Pierre Greisen*†, Richard Emler*, Michael Schaffner*, Simon Heinzle†, Frank Gürkaynak*
*ETH Zurich, 8092 Zurich, Switzerland
†Disney Research Zurich, Switzerland

***Abstract*—Current digital video pipelines are progressing towards ever higher image resolutions and frame rates, a trend which increases computational requirements of mobile end-user devices. Moreover, due to the wide variety of devices with display sub-systems, video streams often need to be adapted to the capabilities of the respective platform. In this work, we present a rendering core that is able to perform spatially-varying geometrical transforms with implicit anti-aliasing in real-time on high-definition video. The rendering is realized with a high-quality elliptical weighted average (EWA) splatting algorithm. The ASIC implementation is fabricated in a 130 nm CMOS technology, and is equipped with a standard display interface and a QDRII RAM interface. The ASIC achieves at least 1080p30 (full HD) video I/O, and is able to perform per-pixel transformation on the video stream in real-time and at low latency.**

I. 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) [1], non-linear stereoscopic 3D (S3D) adaption [2], 2D to S3D conversion and S3D to multi-view generation [3] [4] [5]. 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. 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.

In this work, we present a general purpose *view rendering* engine supporting arbitrary video warping function. Our engine employs a high-quality elliptical weighted average (EWA) rendering approach, which implicitly incorporates anti-aliasing without excessive blurring, and is based on our previous work [6]. In contrast to [6], our rendering core supports full color video formats of full HD video at more than 30 frames per second (1080p30).

Related work: In recent years, various view synthesis and image rendering architectures have been presented. However, the majority of these architectures has been optimized for one particular rendering application, such as depth-image based rendering (DIBR) [7] [8], stereo rectification [9], or non-linear lens correction [10], [11]. In contrast, our rendering engine can process any type of geometrical image transformations. In particular, it can be either used for global per-frame transformations such as (wide-angle) lens undistortion, but also for spatially varying per-pixel transformation such as in video retargeting. Most previous rendering architectures reported in literature employ inferior resampling filters based on bilinear interpolation, and have very limited anti-aliasing filter support compared to the Gaussian EWA filters employed in this work.

Summary of contributions: We present a view rendering system for a wide range of deformations of full color 1080p video, derived from our previous work [6]. In contrast to [6], we re-designed the pipeline for the increased throughput required for 1080p video with at least 30 fps, and included interfaces for video input and output, as well as an interface to external memory components. Our corresponding VLSI architecture is fabricated in a 130 nm CMOS process. The resulting ASIC is able to operate as a component within a system, and our VLSI architecture could alternatively be integrated as a processing core for a mobile SoC.

II. System Overview

In this work, we address the problem of view rendering given an image 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 usually can be 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.

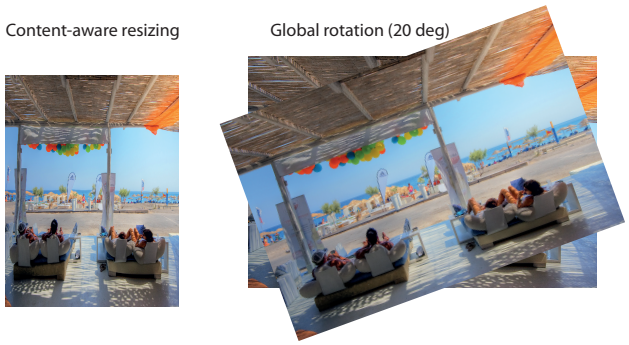

Fig. 1. Examples of transformations that are possible with our view rendering system. In addition to global per-frame transformations such as rotations or zoom (right), our system also allows arbitrary non-linear transformations for each pixel. Such transformations are essential for content-aware resizing applications (left).

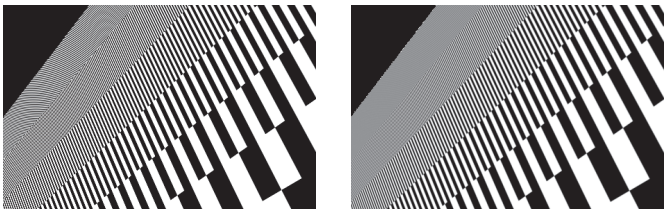

Fig. 2. Quality comparison of bi-cubic spline interpolation (left) vs. EWA resampling (right). The EWA framework reduces aliasing artifacts greatly.

While our setup is able to perform global per-image transformations easily, its strength lies in the ability to realize locally-adaptive non-linear deformation of 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 two examples of transformations that are possible with the system presented in this work. Figure 2 highlights the benefit of using an EWA approach compared to alternative schemes such as bicubic spline interpolation; bilinear interpolation shows even more artifacts.

A. Image Resampling

Transforming an image with a per-pixel mapping function is equivalent to 2D image resampling. First, the input image is transformed into continuous domain using an interpolation function. Then, the per-pixel mapping is realized as a coordinate transformation into the continuous source image. Finally, the transformed source image is resampled on the regular output pixel grid. To avoid aliasing, an additional anti-aliasing filter is applied in target space. We refer to [12] or [13] for more details on general image resampling or image warping. The derivation of EWA splatting is given in [14] or [6].

B. Image Resampling via EWA Splatting

In this work, we employ the elliptical weighted average (EWA) splatting framework [6] to perform image resampling. 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 $\mathbf{u}_k$. Let m be an arbitrary spatially-varying (pixel) mapping function, which is approximated by a first order Taylor expansion around $\mathbf{u}_k$: $m(\mathbf{u}_k) + J_k(\mathbf{u} - \mathbf{u}_k)$, where J_k is the 2D Jacobian of $m()$. The complete EWA resampling process is then summarized as follows. First, the per-pixel covariance matrix is calculated from the warping grid Jacobian J_k and covariance matrices

$$C_k = J_k V_{\mathrm{i}} J_k^T + V_{\mathrm{a}}. \tag{1}$$

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

$$v_h \leftarrow \frac{w_k |J_k|}{2\pi\sqrt{|C_k|}} e^{-1/2(\mathbf{x_h} - m(\mathbf{u}_k))^T C_k^{-1} (\mathbf{x_h} - m(\mathbf{u}_k))}. \tag{2}$$

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

$$\rho_h \leftarrow \frac{|J_k|}{2\pi\sqrt{|C_k|}} e^{-1/2(\mathbf{x_h} - m(\mathbf{u}_k))^T C_k^{-1} (\mathbf{x_h} - m(\mathbf{u}_k))}. \tag{3}$$

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 $m(\mathbf{u}_k)$ [6]

$$m(\mathbf{u}_k) + \begin{pmatrix} \pm\sqrt{C_k(1,1)} \\ \pm\sqrt{C_k(2,2)} \end{pmatrix}. \tag{4}$$

C. View Rendering System

The system-level data flow for our rendering architecture is illustrated in Figure 3. First, input video data and per-pixel warp information is streamed to our video rendering core. The rendering core then performs EWA rendering presented in the previous section. During rendering, external memory is accessed through a memory controller. The transformed output video is then streamed to an external DVI interface.

The external RAM serves as output frame buffer as well as intermediate buffer to collect the contributions of the Gaussian kernels. The external buffer furthermore allows for arbitrary mapping functions, since we can transform the incoming row-by-row pixel stream into an arbitrary order at the output. Note that approaches based on on-chip line buffers only can support arbitrary deformations to a limited degree: the maximum deviation from the vertical input coordinates is fixed by the number of lines in the on-chip buffer.

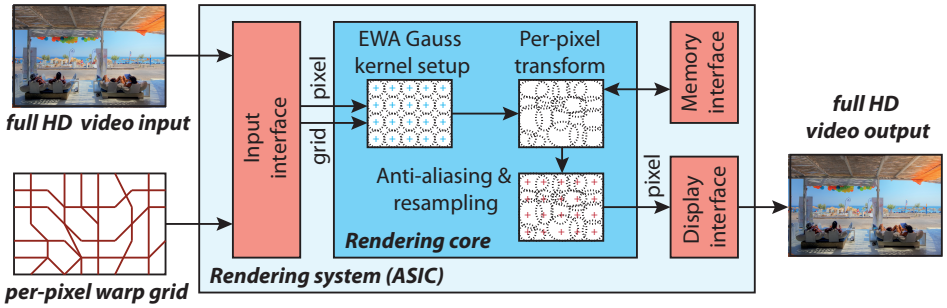

Fig. 3. System overview of the view rendering engine and interfaces.

While our previous work [6] presents a similar architecture, it focuses merely on algorithmic and architectural concepts, and the implementation neither supports HD color video nor employs an I/O interface designed to be used in an actual system.

III. VLSI Architecture

In the following section, we describe the VLSI architecture of the rendering core in detail.

A. Top-level Data Flow

A top-level diagram is given in Figure 4. The ASIC 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 m is streamed in parallel. 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. As a benefit of the quad representation, 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 $(W+1)\cdot(H+1)$ 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 neighoring 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. The 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.

Our system uses separate clock domains for the core and the display interface, and the clock rate of the splatting blocks can be adjusted to match the throughput of the available resources to the requirements of the (expected) image transformations. Note that our system keeps track of the utilization of all available splatting units, which can be queried by the data source through a service interface.

Due to mathematical properties of the summation operation in (2), 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 an extremely high memory bandwidth to the output image in the external frame buffer, we employ a two-level cache structure. 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.

B. 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 2x24 bit pixel coordinates, resulting in a bandwidth of 4.5 GBit/s for 1080p30. To reduce the input bandwidth, we employ a simple differential compression scheme. The

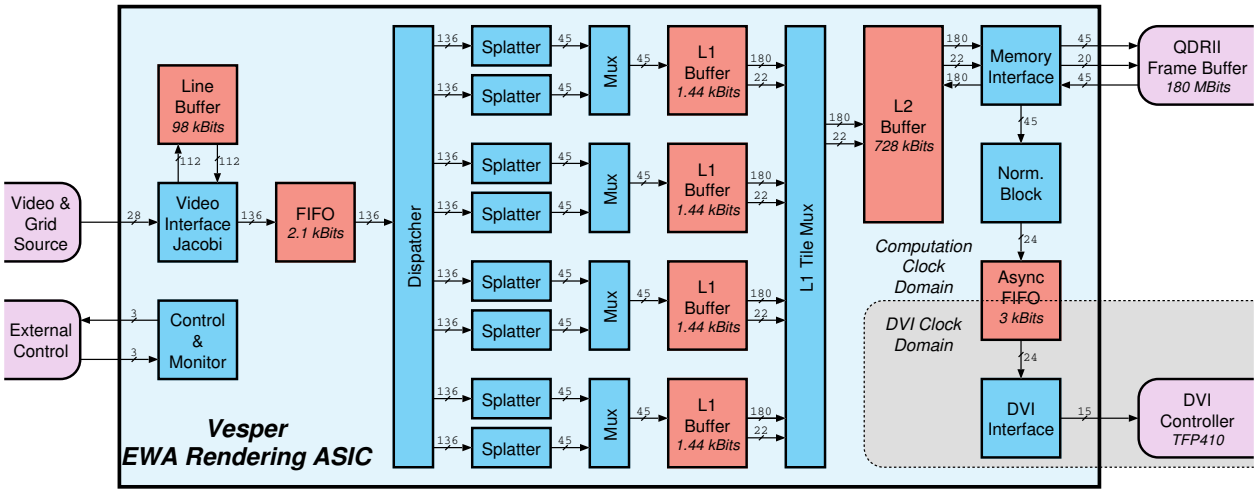

Fig. 4. Top level block diagram of the VESPER EWA chip.

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 values are decomposed into several sub-words, i.e., the MSBs and LSBs are separated. Only sub-words that change are then transmitted. The principle relies on the observation, that the MSBs of pixels and pixel positions change very rarely compared to the LSBs. 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.

Input Interface: We dedicate 28 pins for the data input, resulting in an available bandwidth of 4.8 GBit/s at 170 MHz, the maximum achieved clock frequency for our design. At this clock rate, the system would be able to receive an uncompressed 1080p30 video stream. While our compression scheme is not strictly required for 1080p30, doing so gives some margin for processing higher data rates (1080p48) and allows for operating the core at lower clock frequencies reducing the overall power consumption. Due to the quad grid representation, all four quad-points adjacent to each pixel need to be determined. Therefore, a small line buffer storing the previous quad line is required.

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.

C. Arithmetic Processing Elements

The core splatting units are similar to the ones presented in [6]. In a nutshell, the splatting units implement the EWA equation (2) 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 re-sampling covariance matrix is adapted per-pixel to reduce the amount of blurring. The adaptive mode has been introduced in [6] and its impact on overall area can be neglected.

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.

D. 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. To achieve practical performance, the number of contributions per input pixel is limited by a bounding box, and thus the Gaussian weights are truncated at the bounding box limits. Simulations have shown that bounding boxes of 4 to 9 pixels are sufficient to capture the majority of the non-zero contributions of the Gaussian kernel. Hence, the accumulation bandwidth is between 2×4 and 2×9 times larger than the input bandwidth, as each accumulation is

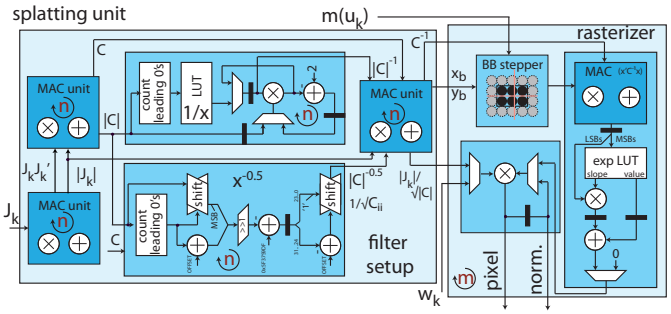

Fig. 5. Data path of the EWA splatting core.

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

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

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, and shifts the bandwidth burden to the on-chip buffers, reducing the effective n_{pps}. In simulations, a cache efficiency resulting in $n_{\text{pps}} = 3$ is always achieved, and the required bandwidth is reduced to 22.4Gbit/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 a separate read and write port, which can be accessed in parallel. Moreover, the data is transmitted in double edge mode. A 9-bit QDR RAM port therefore has 3Gbit/s read and 3Gbit/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 30Gbit/s.

E. 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.

F. 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 very strict timings, 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 the final 24 bit RGB values, by normalizing the accumulated RGB values with their weights. An asynchronous FIFO is used for clock domain crossing and for ensuring that pixels are always available to the DVI interface.

The DVI interface on the ASIC can be configured to different modes, depending on the current application. That is, common resolutions and frame rates as well as progressive or interlaced modes can be selected. Note that the DVI standard only supports a discrete set of configurations. For instance, 1080p30 is not a supported mode, which is why we convert 1080p30 to 1080p60 at the output interface by interleaving it with zeroes.

IV. Implementation Details

The implemented ASIC is denoted VESPER[1] and is fabricated in a 130 nm CMOS process. Table I summarizes the key features of VESPER.

A. Area, Inputs and Outputs

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, 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 9mm^2 which is significantly smaller than the 5x5 mm^2 the chip currently occupies.

B. Clocking

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 DVI specific clock. The asynchronous data interface is implemented using

[1] VESPER: VLSI EWA SPlatting Engine and Rasterizer

TABLE I
KEY FEATURES OF THE VESPER CHIP

Functional Characteristics	
Technology	UMC 130 nm, 8Metal
Core Voltage	1.2 V
Package	CPGA 256
Core Area	5 mm × 5 mm
Circuit Complexity	1.7 MGE (8.8 mm^2)
Logic (std. cells)	0.81 MGE (4.1 mm^2)
On-chip Memory (835 kbits)	0.84 MGE (4.7 mm^2)
Maximum Clock Frequency	170 MHz
Performance	
I/O Format	24 bit RGB
Maximal Input Resolution	2048 × 2048
Maximal Output Resolution	2048 × 2048
Performance (1080p), min – max	30 fps – 48 fps
Rendering Input Bandwidth (1080p30)	1.5 Gbit/s
Rendering Output Bandwidth (1080p30)	56 Gbit/s
External Memory Bandwidth (1080p30)	12.1 Gbit/s – 22.4 Gbit/s
Available Memory Bandwidth	30 Gbit/s @ 170 MHz

an asynchronous FIFO, see [15]. In addition to the two internal clocks for DVI and core related logic, we feed the actual chip clock for the DVI transmitter and RAM chips through the ASIC, for timing analysis and delay balancing reasons. We can therefore add a phase shift on the clocks to the external components relative to the internal clocks.

C. Obtained Throughput

The I/O bandwidth, throughput of the splatting units and 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 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 has been fabricated, and the functional tests have verified that VESPER is fully functional at 170 MHz.

V. CONCLUSIONS

Arbitrary transformations of high-definition videos can be efficiently rendered using a VLSI EWA splatting architecture. The proposed VLSI core can be used in an end-user device and enables 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 resolutions and frame-rates, such as the upcoming quad HD standards (2160p) or the high-frame rate (HFR) standards.

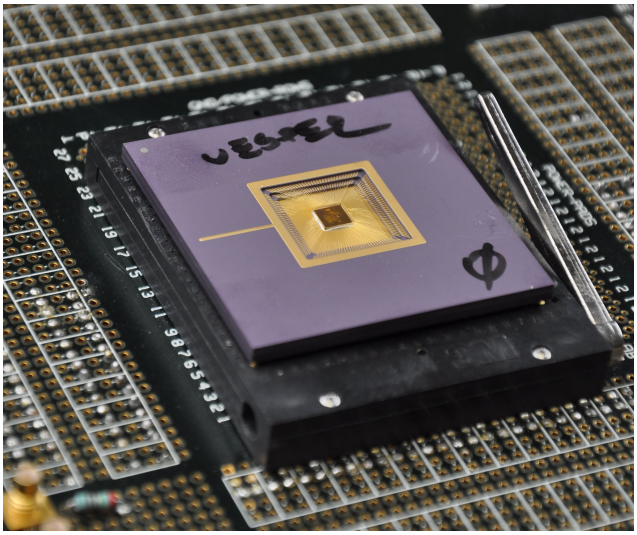

Fig. 6. The VESPER chip on the ASIC tester.

REFERENCES

[1] P. Krähenbühl, M. Lang, A. Hornung, and M. Gross, "A system for retargeting of streaming video," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 5, pp. 1–10, 2009.

[2] M. Lang, A. Hornung, O. Wang, S. Poulakos, A. Smolic, and M. Gross, "Nonlinear disparity mapping for stereoscopic 3D," *ACM Trans. on Graphics (Proc. SIGGRAPH)*, vol. 29, no. 3, 2010.

[3] M. Farre, O. Wang, M. Lang, N. Stefanoski, A. Hornung, and A. Smolic, "Automatic content creation for multiview autostereoscopic displays using image domain warping," in *Multimedia and Expo (ICME), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–6.

[4] M. Tanimoto, M. Tehrani, T. Fujii, and T. Yendo, "Free-viewpoint tv," *Signal Processing Magazine, IEEE*, vol. 28, no. 1, pp. 67 –76, 2011.

[5] M. Do, Q. Nguyen, H. Nguyen, D. Kubacki, and S. Patel, "Immersive visual communicatio n," *Signal Processing Magazine, IEEE*, vol. 28, no. 1, pp. 58 –66, jan 2011.

[6] P. Greisen, M. Schaffner, S. Heinzle, M. Runo, A. Smolic, A. Burg, H. Kaeslin, and M. Gross, "Analysis and vlsi implementation of ewa rendering for real-time hd video applications," *Transactions on Circuits and Systems for Video Technology*, vol. accepted, 2012.

[7] Y.-R. Horng, Y.-C. Tseng, and T.-S. Chang, "VLSI architecture for real-time HD 1080p view synthesis engine," *IEEE transactions on circuits and systems for video technology*, vol. 21, no. 9, pp. 1329–1340, September 2011.

[8] F.-J. Chang, Y.-C. Tseng, and T.-S. Chang, "A 94fps view synthesis engine for HD1080p video," in *Visual Communications and Image Processing (VCIP), 2011 IEEE*, November 2011, pp. 1 –4.

[9] P. Greisen, S. Heinzle, M. Gross, and A. Burg, "An FPGA-based processing pipeline for high-definition stereo video," *EURASIP Journal on Image and Video Processing*, vol. 2011, no. 1, p. 18, 2011.

[10] K. V. Asari, "Design of an efficient vlsi architecture for non-linear spatial warping of wide-angle camera images," *Journal of Systems Architecture*, vol. 50, no. 12, pp. 743 – 755, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762104000682

[11] S. Oh and G. Kim, "Fpga-based fast image warping with data-parallelization schemes," *Consumer Electronics, IEEE Transactions on*, vol. 54, no. 4, pp. 2053 –2059, november 2008.

[12] G. Wolberg, *Digital image warping*. IEEE Computer Society press, 1990, vol. 3.

[13] R. Szeliski, S. Winder, and M. Uyttendaele, "High-quality multi-pass image resampling," Microsoft Research, Tech. Rep., 2010.

[14] M. Zwicker, H. Pfister, J. V. Baar, and M. Gross, "EWA splatting," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 3, pp. 223–238, 2002.

[15] C. Cummings, "Simulation and synthesis techniques for asynchronous fifo design," in *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, 2002.