Hybrid ASIC/FPGA System for Fully Automatic Stereo-to-Multiview Conversion using IDW

Michael Schaffner, Frank K. Gürkaynak, Pierre Greisen, Hubert Kaeslin, Luca Benini, Aljosa Smolic

***Abstract*—Recently, multiview autostereoscopic displays (MADs) have become commercially available which enable a limited glasses-free 3D experience. The main problem of MADs is that they require several (typically 8 or 9) views, while most of the 3D video content is in stereoscopic 3D (S3D) today. In order to bridge this gap, the research community started to devise automatic multiview synthesis (MVS) methods. These algorithms require real-time processing and should be portable to end-user devices to develop their full potential. To this end, we revisit an algorithmic solution based on image domain warping (IDW) and devise a hardware architecture of a complete synthesis pipeline, provide insights on where the computationally challenging parts are, and present implementation results of a hybrid FPGA/ASIC prototype - which is the first hardware implementation of a complete, IDW-based MVS system. Based on these results, we also estimate the complexity and energy efficiency of a fully integrated solutionin 65 nm and 28 nm CMOS technology and show that a full-HD real-time solution on a single chip is within reach. The proposed architecture could be used as a co-processor in a system-on-chip (SoC) targeting 3D TV sets, thereby enabling efficient content generation with limited user interaction (e.g. depth range adjustment) in real-time.**

***Index Terms*—video processing, real-time, stereoscopic 3D (S3D), multiview synthesis (MVS), image domain warping (IDW), FPGA, ASIC, VLSI**

I. Introduction

Over the last couple of years, video capture, post-processing and distribution technologies for *stereoscopic 3D* (S3D) content have become mature enough for broad commercialization [1], [2], [3]. Coupled with the box office success of S3D movies, this has brought on renewed interest in the development of S3D-capable consumer-electronic devices such as TV sets. However, most such devices require the viewers to wear some sort of shutter- or polarization glasses, which is often regarded as an inconvenience [4]. Recently, so-called *multiview autostereoscopic displays* (MADs) [5], [6] have become commercially available. These are able to project several views of a scene simultaneously - enabling a glasses-free 3D experience and a limited motion parallax effect in horizontal direction. However, appropriate content for such displays is largely inexistent since storage and transmission of *high definition* (HD) content with more than two views is impractical and even infeasible in some cases. The fact that each MAD has different parameters exacerbates the problem. In order to bridge this *content-display gap*, *multiview synthesis* (MVS) methods [7], [8], [9], [3] have been devised over the past couple of years. These methods are able to generate M virtual views from a small set of N input views, as shown in Figure 1.

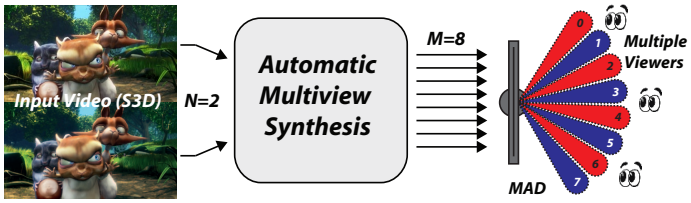

Fig. 1: MV synthesis generates a large number of synthetic views (e.g. $M = 8, 9$ for current MADs) from a small number of input views ($N = 2$ for S3D). Images © copyright 2008, Blender Foundation / www.bigbuckbunny.org.

Common MVS methods are based on *depth image based rendering* (DIBR) [10], [3], where a dense depth map of the scene is used to reproject the image to new viewpoints. Although physically correct, this approach requires accurate depth maps and additional inpainting steps. Our work uses an alternative conversion concept suggested by [11] which is based on *image domain warping* (IDW) [12]. The IDW framework allows to locally distort image regions via a non-linear, two-dimensional transformation which is obtained by solving a *least-squares* (LS) problem. The constraints for this problem are formulated using image features extracted from the input images. This technique is promising as it does not rely on pixel dense depth, but only on robust, sparse point correspondences. Further, no inpainting is required which is still an algorithmically difficult step of DIBR based MVS [13].

Multiview synthesis, using IDW methods as well as alternative approaches, are computationally intensive - yet they should run efficiently in real-time and should be portable to end-user devices to develop their full potential. To this end, we make the following contributions:

- We design and implement the first complete hardware system for IDW-based, automatic MVS. While several subcomponents have already been addressed in more detail in previous work (feature points [14], saliency [15], warp rendering [16], [17]) the system integration aspect is new and represents one of the main contributions.
- The IDW processing pipeline for MVS [11] is revisited and the algorithms involved in the substeps are selected such that an efficient hardware implementation is possible. New w.r.t. to the existing algorithmic solution are the hardware specific design decisions.
- Based on linear solver architectures we have developed

M. Schaffner is with ETH Zurich, Switzerland, and Disney Research, Zurich, Switzerland. Frank. K. Gürkaynak, P. Greisen, and H. Kaeslin are with ETH Zurich, Switzerland. L. Benini is with ETH Zurich, Switzerland, and University of Bologna, Italy. A. Smolic is with Disney Research, Zurich, Switzerland. {schaffner,kgf,greisen,kaeslin,benini}@iis.ee.ethz.ch {smolic}@disneyresearch.com

previously [18], we design an improved solver which contains a programmable matrix assembly stage and makes use of custom, fused floating-point (FP) arithmetic.
- Using these results, we estimate the implementation cost and energy-efficiency of a completely integrated solution and show that our proposal is viable as an accelerator in a mobile SoC. Also, we provide a comparison with related components from DIBR-based systems.

II. Background and Related Work

A. Multiview Synthesis Methods

The IDW method used in this work relies heavily on prior work on warping approaches for video content adaption such as aspect-ratio retargeting [19], S3D retargeting [20] and non-linear disparity mapping [21]. In these applications, the video is warped in a content-aware manner in order to fulfill certain constraints. Image features (such as edges) and visual importance maps (saliency) are used in order to determine which parts of the image are important and should be preserved. Unavoidable distortions are moved to visually unimportant regions. This IDW framework has been extended for automatic MVS by [11]. The artifacts caused by geometrical incorrectness are minor and visually hardly noticeable. In fact, exhaustive and formal subjective experiments performed by MPEG [22], [11] revealed that IDW as proposed here performs at least as well as DIBR methods, even if those used pre-computed and hand-tuned depth maps. Based on these results, MPEG adopted warp coding for the 3D extension of the new HEVC standard [23], thereby enabling IDW-based MVS.

Related methods [7], [10], [9], [3] for MVS and *free-viewpoint television* (FTV) rely on DIBR in order to generate new viewpoints. These methods use dense disparity maps to re-project pixels into new virtual views using the relations from epipolar geometry [24]. Although physically correct, DIBR still has two major limitations that remain fundamentally unresolved. First, DIBR-based rendering always introduces disocclusions which have to be filled using inpainting techniques. Second, a dense disparity map of the scene is required which is difficult to obtain in practice, and in real-time implementations often noisy and incomplete.

B. Image Features and Saliency Estimation

A crucial ingredient for multiview synthesis using IDW is establishing robust sparse point correspondences between the two input images. Numerous algorithms and variations thereof have been devised over the past decade. Especially so termed *binary* descriptors such as BRIEF [25] and *Semantic Kernels Binarized* (SKB) [26] are attractive for hardware implementations since they are inexpensive to compute, require less storage, and can be matched very efficiently. The hardware architecture implemented in this work is based on an efficient hardware implementation of SKB by [14].

Further, the IDW pipeline relies on so-called *saliency maps* that indicate visually important regions in the images. Algorithms for the extraction of saliency information are usually based on psycho-visual attention models [27], and can greatly vary in computational complexity. E.g., a very recent, high-quality saliency for stereoscopic video was proposed by [28]. Their model not only considers low-level cues such as luminance and chrominance, but also high-level cues. The algorithm, involves many substeps such as optical-flow calculation, segmentation, and scene-type classification, and therefore is less suited for a real-time hardware implementation. Here, we use an efficient implementation [15] of the Fourier-transform-based saliency proposed by [29] due to its good tradeoff between computational complexity and quality.

C. Linear Solvers

At the core of most IDW algorithms lies a quadratic energy minimization problem which is discussed in more detail in Section III. Such problems can be tackled by solving a large, linear system with typically banded sparsity structure. Since this system is positive definite and symmetric, *conjugate gradient* (CG) and *Cholesky decomposition* (CD) based solvers are usually the algorithms of choice [30], [31]. The design of hardware architectures for these iterative and direct solver types and related work thereof have been discussed in detail in [18], [32]. These two serve as a basis for the CD-based solver implemented in this work.

D. Resampling for MADs

The MAD specific interleaving process essentially represents a resampling step onto a possibly non-orthogonal or completely irregular sampling lattice. This matter is discussed in detail in [5], [6]. Due to the imperfect optics of the MADs, the views often suffer from crosstalk which can be mitigated using appropriate filtering [6]. Further, MADs only have a limited depth budget, and the content to be viewed can be optimized in order to reduce spatial aliasing artifacts [33], [34].

In this work, we use an adapted version of *elliptical weighted average* (EWA) splatting [16] for the warping and interleaving processes. This is a *forward-mapping* algorithm and its mathematical properties allow to analytically combine interpolation and anti-aliasing filters for the transformation with the display pre-filter.

E. Real-time Systems and Hardware Architectures

Most real-time systems for MVS are based on DIBR, and the depth estimation and view synthesis steps are usually treated separately in literature. There are only a few publications where both parts have been combined, such as the work by [35] where one of the first complete real-time systems for MVS has been implemented using a high-end workstation. In [36], an FPGA-based system able to synthesize one synthetic view from 1080p S3D input at 60 fps is presented.

FPGA-based hardware accelerators for depth estimation have been developed by [37], [38], [39], [40], [41]. A high-quality, global stereo matching algorithm, achieving a throughput of 720p@2.6 fps is implemented by [37]. The design by [38] outputs CIF depth maps @42 fps whereas the designs by [39] and [40] reach resolutions of up to XGA@60 fps. The architecture from [41] is based on the algorithm used in the MVS system [35] and shows very promising performance of up to $2\times$1080p@30 fps (S3D).

DIBR-based view synthesis engines based on FPGAs and CMOS integrated circuits are developed in [42] and [43], [44],

[45], [46], respectively. In [44], [46] and [45], single view single view synthesis engines with throughputs of 1080p@32 fps, 1080p@94 fps and 4 k@216 fps, are developed. MVS rendering solutions for 1080p@60 fps and SXGA@29 fps are presented in [42] and [43].

In contrast to mentioned work, we devise a complete, IDW-based hardware system that takes a 1080p S3D input stream and that outputs interleaved 1080p video for display on a MAD. A more detailed comparison is given in Section V-D.

III. Algorithmic Flow

This section is a summary of the algorithmic flow of the implemented MVS scheme, and is based on work presented previously in [11], [18]. Further, we explain the specific selection and parametrization of the involved methods with respect to hardware efficiency. As shown in Figure 2, the input to the IDW processing pipeline is the S3D footage (left and right images) which is analyzed in order to reveal image features such as point correspondences, edges and saliency information in a first step. Those features are then used to formulate a global energy minimization problem the solution of which results in two warps — one for each input image. These warps describe the (nonlinear) transformation of the input images to a viewing position centered between the two original views. The new views are then generated by first inter- and extrapolating the two warps to the desired view positions; and then by resampling the S3D input according to those interpolated warps. Finally, the generated views are interleaved in such a way that they can be displayed on the MAD. The individual steps are explained in more details below.

A. Sampling Lattices, Domains and Warps

In the following, we will use the terms *sampling lattice* Λ [5] and *domain* of the image/warp. The former is a generalization of sampling grids, and may also describe regular, non-orthogonal arrangements. The latter describes the physical dimensions that a certain image or warp spans (measured in pixels). This distinction allows to define coordinate quantities (e.g. warps) or feature maps (e.g. saliency) over a certain physical dimension, but with a different amount of sampling positions (e.g. for subsampled quantities). Non-orthogonal sampling arrangements are important to describe the interleaving patterns of MADs. A sampling lattice is defined as all linear combinations of a set of (not necessarily orthogonal) basis vectors $\Lambda = \{\mathbf{u} : \mathbf{u} = n_1\mathbf{v}_1 + n_2\mathbf{v}_2 + ... + n_N\mathbf{v}_N | n_i \in \mathbb{Z} = V\mathbf{n}\}$. In order to emphasize the distinction between orthogonal and non-orthogonal sampling arrangements, we will refer to the former as a grid, and to the latter as a lattice. Both are defined by a sampling matrix, but in case of a grid we have the restriction that V is diagonal. A domain of an image or warp will be denoted as $\mathcal{D} = \{w \times h\} \subset \mathbb{R}^2$ where w and h are the dimensions in pixels.

An image warp can be described using a non-linear, two-dimensional *forward* mapping $\mathbf{w}(\mathbf{u}) : \mathbf{u} \in \mathcal{D}_{in} \rightarrow \mathbf{w}(\mathbf{u}) \in \mathcal{D}_{out}$, where $\mathbf{u}$ is the coordinate in $\mathcal{D}_{in}$. Since it is infeasible to express the image warps as analytical expressions in general, they are stored as discretized functions defined over a two-dimensional grid Λ_{wrp} and with $\mathcal{D}_{in}$. Linearized indices, e.g. k, are used to enumerate the warp vertices $\mathbf{w}_k$ sampled at discrete sampling points $\mathbf{u}_k$ on Λ_{wrp}. These may coincide with the pixel positions in the input image in the case of pixel dense warps. However this is usually not the case since the warps are often subsampled by around $10\times$ for efficiency reasons (subsection III-C3). Superscript letters indicate to which view and coordinate-dimension a particular quantity belongs. For example, ^{l} and ^{r} denote the left and right input views, and ^{x} and ^{y} the first and second dimension of a two-dimensional quantity. The two S3D input images will be denoted as $\mathbf{i}^l_{in}$, $\mathbf{i}^r_{in}$. They are defined over $D_{in} = \{w_{in} \times h_{in}\}$ and the sampling grid Λ_{in} is defined as $V_{in} = \mathrm{diag}\,(1, 1)$.

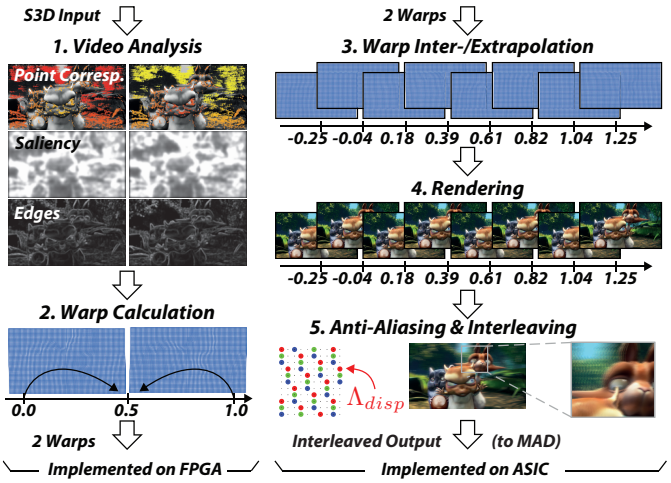

Fig. 2: IDW pipeline for MVS (refer to the text for more details). As indicated, the first part is implemented on an FPGA, and the second part on an ASIC. Images © copyright 2008, Blender Foundation / www.bigbuckbunny.org.

B. Video Analysis

III-B1 Sparse Point Correspondences and Disparities. In MVS, the disparities are the most important features since they reveal the 3D geometry of the observed scene. Yet these have to be robust in order to get good results. As opposed to DIBR-based methods, the IDW approach works on sparse disparities. In this work we adopted features based on SKB [26], since they work very well in the setting of almost ideally rectified stereo video. As shown in Figure 3, SKB features a low outlier rate and, therefore, it is possible to use the features without additional RANSAC filtering, which would be costly in hardware. Occasional outliers can be tolerated since the warp calculation process enforces spatial and temporal smoothness. Furthermore, since SKB is a binary descriptor, it can be calculated and matched very efficiently in hardware. Descriptors containing fixed-point or even FP entries such as SURF or SIFT are much more costly in this respect.

Feature points are given in the form of two lists $\mathbf{p}^l_i \in \mathcal{D}_{in}$ and $\mathbf{p}^r_i \in \mathcal{D}_{in}$ with $i \in \{0, 1, ..., n_{pts} - 1\}$. In addition, there is confidence value c_i associated with each pair $\mathbf{p}^l_i$, $\mathbf{p}^r_i$, and we have that $0 \leq c_i \leq 1.0$.

III-B2 Saliency Estimation and Edge Maps. A saliency map identifies the visually important regions in the image, and is used to guide the warp calculation such that deformations are hidden in unimportant regions (e.g. homogeneous parts such as blue sky). Extracting visual saliency is difficult since

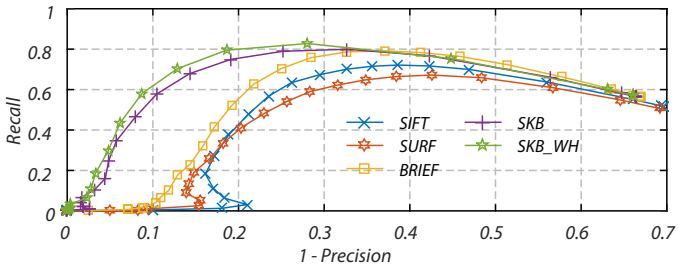

Fig. 3: Performance of a greedy nearest-neighbor matching with a small window, as it is implemented in hardware (test set from [47]). The *recall* is the ratio of correct matches and existing correspondences between left and right image, and *1-precision* is the fraction of false matches. The 'SKB' curve is for the original 256 bit descriptor described in [14], [26], and 'SKB_WH' represents the improved version used in this project where the original SKB kernels have been replaced by binary Walsh-Hadamard basis functions.

it is a subjective measure that depends to some extent on video content, viewer, and application [27]. Here we use the *quaternion-Fourier-transform*-based (QFT) algorithm from [29] which exhibits a good tradeoff between computational complexity and quality. The algorithm leverages the *phase-spectrum* of a video sequence, which carries information on where discernible objects are located in an image. Note that the QFT can be efficiently calculated using two separate 2D *Fast Fourier Transforms* (FFTs) [15]. In the following, the saliency maps extracted from $\mathbf{i}_{in}^{l}$ and $\mathbf{i}_{in}^{r}$ are denoted as $\{\mathbf{s}^l, \mathbf{s}^r\} \in [0.0, ..., 1.0]$ and are defined on $\mathcal{D}_{in}$, Λ_{wrp}.

Very salient lines should also be preserved in order to avoid bending them in the warped images. This can be done using several methods, e.g. by extracting straight lines using the Hough transform. However, since this feature is not as important as the saliency and the point correspondences, simple gradient-magnitude maps without Canny edge post-processing are used. Such gradient-magnitude maps can be extracted very efficiently using 3×3 Sobel filters. The edge maps will be denoted as $\{\mathbf{e}^l, \mathbf{e}^r\} \in [0.0, ..., 1.0]$ and are defined on $\mathcal{D}_{in}$, Λ_{wrp}.

C. Warp Generation

The warps are calculated by solving a quadratic energy minimization problem of the form

$$\min_{\mathbf{f}} \left(E\left(\mathbf{f}\right)\right) = \min_{\mathbf{f}} \left(E_{data}\left(\mathbf{f}\right) + E_{smooth}\left(\mathbf{f}\right)\right) \qquad (1)$$

where the data term E_{data} enforces function values at certain coordinate positions, and the smoothness term E_{smooth} is a regularizer that propagates these known values to adjacent sampling positions. The vector $\mathbf{f}$ holds the samples of the unknown warp vertices on Λ_{wrp}. The data term itself is composed of the energy term E_{pt} containing the desired vertex positions in the target image, and the energy term E_t which enforces temporal consistency:

$$E_{data} = \lambda_{pt} E_{pt} + \lambda_t E_t.$$

λ_{pt} and λ_t are weighting parameters to set the relative importance of a particular constraint. Analogously, the smoothness term comprises the two constraints E_{sal} and E_{edge} which determine the local smoothing strength:

$$E_{smooth} = \lambda_{sal} E_{sal} + \lambda_{edge} E_{edge}.$$

Again, λ_{sal} and λ_{edge} are relative importance weights. Recall that we solve for two warps, each of which maps either the left or the right image to a virtual view at position 0.5 on the normalized baseline. This asks for setting up four quadratic energy functionals E^{lx}, E^{ly}, E^{rx} and E^{ry} (two coordinate dimensions for each input view) which can all be minimized independently. In the following we will only describe the constraints assembly for the x-problem of the left warp E^{lx} - the formulations for all other problems follow analogously.

III-C1 Data Constraints. The term E_{pt} contains the desired positions of the warp vertices in the warped image. As illustrated in Figure 2, the target view for both warps is centered between both input views. The sparse point correspondences are used to formulate these constraints by setting

$$E_{pt}^{lx} = \sum_{i=0}^{n_{pts}-1} \left(s_k^l \cdot c_i^l\right)^2 \cdot \left(f_k^{lx} - 0.5\left(p_i^{lx} + p_i^{rx}\right)\right)^2, \qquad (2)$$

where s_k^l is the saliency value at coordinate position $\mathbf{p}_i^l$, f_k^{lx} is the warp vertex at position $\mathbf{p}_i^l$ and c_i is the associated confidence value. Note that these constraints are only defined for the warp vertices corresponding to $\mathbf{p}_i^l$. There may be many warp vertices without any point constraints. In the case where the warps are sub-sampled w.r.t. to the image resolution, the point constraints have to be distributed to the surrounding four warp vertices using bilinear weighting [18]. The temporal constraint is defined using the warp of the previous frame $\mathbf{g}$:

$$E_t^{lx} = \sum_{k \in \Lambda_{wrp}} \left(f_k^{lx} - g_k^{lx}\right)^2.$$

III-C2 Smoothness Constraints. The saliency constraints

$$E_{sal}^{lx} = \sum_{k \in \Lambda_{wrp}} {s_k^l}^2 \cdot \left(\left(f_{k+1}^{lx} - f_k^{lx}\right)^2 + \left(f_{k+h}^{lx} - f_k^{lx} + d_x\right)^2\right),$$

penalize deformations of the regular warp grid (assuming a column-major enumeration k of the warp vertices). d_x denotes the sample spacing, and is related to the warp grid sampling matrix $\Lambda_{wrp} = \mathrm{diag}\left(d_x, d_y\right)$. The edge constraints penalize deformations orthogonal to the edge directions:

$$E_{edge}^{lx} = \sum_{k \in \Lambda_{wrp}} {e_k^l}^2 \cdot \left(f_{k+1}^{lx} - f_k^{lx}\right)^2.$$

III-C3 Sparse Linear System. Since the $E(\mathbf{f})$ is quadratic in the elements of $\mathbf{f}$, the solution of (1) is a LS solution. As shown in [18], the terms of $E(\mathbf{f})$ can be arranged such that $E\left(\mathbf{f}\right) = ||A\mathbf{f} - \mathbf{b}||^2$, where matrix A and vector $\mathbf{b}$ both encode the constraints explained before. The global minimum is then achieved if $\nabla E\left(\mathbf{f}\right) = 2A^T\left(A\mathbf{f} - \mathbf{b}\right) = 0$, i.e. the solution can be found by solving the *normal equations* $\left(A^T A\right)\mathbf{f} = A^T\mathbf{b}$, where $\left(A^T A\right)$ is symmetric, quadratic and positive definite. As shown in [18], it is possible to deduce analytical expressions for $A^T A$ and $A^T\mathbf{b}$, which makes the assembly of these equations more efficient.

Since the constraints are defined on small, local neighborhoods on Λ_{wrp}, the matrix $\left(A^T A\right)$ is very sparse and only contains one main- and nine off-diagonals. Further, its dimension $n = w_{wrp} \times h_{wrp}$ is in the order of tens of thousands to millions - depending on the resolution of Λ_{wrp}.

Current video content is predominantly in 1920×1080 format, resulting in 4 problems (E^{lx},E^{ly},E^{rx} and E^{ry}) with nearly two million variables for each video frame. Solving such large systems at frame rates of up to 30 fps is computationally very demanding and infeasible for real-time applications. Also, such pixel dense solutions are not necessary as even 10× sub-sampled grids result in sufficient synthesis quality on many HD S3D sequences [11]. This results in realistic grid sizes of about 180×100, which corresponds to 18 k variables.

III-C4 Solution Methods. Linear solver algorithms fall into two main categories, namely *direct* and *iterative* ones. Direct solvers employ a matrix decomposition such as LU or CD in order to compute an exact solution, whereas iterative solvers successively refine an approximate solution. The choice of the solver is dependent on several factors such as the mathematical properties and structure of $\left(A^T A\right)$, convergence and numerical behaviour, and the complexity of arithmetic operations and memory accesses. The two most widely used algorithms for positive-definite systems are direct CD- and iterative CG solvers [30][31].

As discussed in [18], [32], iterative solvers require an extremely large memory bandwidth. For IDW problems with $n = 18\,\mathrm{k}$ variables, a single-precision CG solver would have to traverse nine n-dimensional vectors per iteration, and well defined problems require in the order of 200 iterations. This would translate into a memory bandwidth in the order of $30\,\mathrm{fps} \times 4 \times 18\,\mathrm{k} \times 200 \times 9 \times 4\,\mathrm{Bytes} \approx 15.6\,\mathrm{GByte/s}$ and memory requirements of around $18\,\mathrm{k} \times 9 \times 4\,\mathrm{Byte} \approx 5.2\,\mathrm{MBit}$. Such a large memory bandwidth is infeasible to realize with external memory on most embedded platforms, which means that the whole solver would have to be implemented with on-chip resources only, which is very costly in this case.

Instead a direct CD-based solver only requires two passes: in the first pass the matrix is decomposed into a lower triangular matrix L, a *forward substitution* $\mathbf{h} = L^{-1}\mathbf{b}$ is performed, and L and $\mathbf{h}$ are written to the memory. In a second pass, L is read in reverse order to perform the *backward-substitution* $\mathbf{f} = L^{-T}\mathbf{h}$. In this case, L is a banded matrix with 102 nonzero (off-)diagonals, so the external memory bandwidth would be $30\,\mathrm{fps} \times 4 \times 2 \times 18\,\mathrm{k} \times (102+1) \times 4\,\mathrm{Bytes} \approx 1.7\,\mathrm{GByte/s}$, and the memory requirements for local buffering amount to $325.1\,\mathrm{kbit}$. Based on these numbers we decided to use a CD-based solver in our hardware implementation. However note that direct methods do not scale well to larger systems beyond a few hundred-thousand variables, and more elaborate solvers based on iterative methods are required in such scenarios [32].

D. Warp Interpolation and Rendering

In the rendering step, the two warps $\mathbf{w}^l$ and $\mathbf{w}^r$ are first bilinearly upsampled from Λ_{wrp} to Λ_{in} in order to form pixel dense warps $\tilde{\mathbf{w}}^l$ and $\tilde{\mathbf{w}}^r$. These are then linearly inter- and extrapolated to all desired view positions α on the normalized baseline. As shown in Figure 2, eight views are generated in this case: four views from the left image $\mathbf{i}^l$, and the remaining four views from the right image $\mathbf{i}^r$. Note that this view interpolation can be used to linearly scale the depth range of the scene at runtime. The resampling is performed using EWA splatting, which is a *forward-mapping* method. Although more complex than traditional bilinear backward mapping [48], EWA has the advantage that no warp inversion is required since these are in calculated in forward format in this application. The EWA framework uses Gaussian filter kernels and the Jacobian of the image warp as a local deformation measure in order to calculate the footprint of an input image pixel in the output image. The input pixels thus correspond to Gaussian *splats* in the output image, which are rasterized within a bounding box and accumulated in a frame buffer. Since Gaussians are closed among themselves and under affine transformations, an anti-aliasing filter for the output image sampling grid can be easily incorporated analytically. A short summary is given below – for a complete derivation see [16].

III-D1 The EWA Filter Kernel. $\tilde{\mathbf{w}}(\mathbf{u})$ can be any of the pixel dense image warps. Let J_k be the Jacobian of the warp at pixel position $\mathbf{u}_k$. The EWA kernel is characterized by the covariance matrix $\Sigma_k = J_k W_i J_k^T + W_{aa} = C_k + W_{aa}$ in the target image domain, where the first term is the transformed interpolation kernel, and the second term is the anti-aliasing kernel. $W_i = \mathrm{diag}(\sigma_i^2, \sigma_i^2)$ and $W_{aa} = \mathrm{diag}(\sigma_{aa}^2, \sigma_{aa}^2)$ are diagonal matrices that parameterize the interpolation and anti-aliasing filters. The weight of the Gaussian filter at the position $\mathbf{v}_j \in \Lambda_{out}$ on the output sampling lattice is calculated as

$$\rho_{jk} = |J_k| \left(2\pi\sqrt{|\Sigma_k|}\right)^{-1} e^{-0.5(\mathbf{v}_j - \tilde{\mathbf{w}}(\mathbf{u}_k))\Sigma_k^{-1}(\mathbf{v}_j - \tilde{\mathbf{w}}(\mathbf{u}_k))},$$

and is multiplied with the pixel value $\mathbf{i}_{in}(\mathbf{u}_k)$. The weights ρ_{jk} are accumulated along with the pixel values $\rho_{jk} \cdot \mathbf{i}_{in}(\mathbf{u}_k)$. Finally, the output pixels $\mathbf{i}_{out}(\mathbf{v}_j)$ are calculated by dividing the accumulated values by the corresponding weights.

III-D2 Filter Parametrization and Display Anti-Aliasing. In [16], it is shown that for a regular, quadratic sampling grid the filter parametrization $\sigma_i \approx 0.39$ leads to the optimal L2 fit of a Gaussian to the ideal low-pass filter in the frequency domain. It is also shown how σ_{aa}^2 can be chosen adaptively such that anti-aliasing is only performed when needed. In our application, the covariance matrices are diagonally dominant, and therefore we can use the *simplified adaptive* scheme.

The resampled views are interleaved according to a special interleaving pattern (like in Figure 2), such that they can be displayed simultaneously on a MAD. Proper care must be taken in order to prevent aliasing, as shown by [5]. The filters are generally non-separable, and in theory a high order is required to approximate the intricate shape of the passband. But [5] also noted that for natural images, the benefit of such filters is rather small. As a result, simpler separable filters that lead to visually pleasing results could be used as well. In this work we use the closedness of Gaussians in order to incorporate a Gaussian display pre-filter analytically into the EWA kernel. Instead of using $\sigma_{aa} = 0.39$ we adapt this value with the *density* $1/|V_{disp}|$ of the display sampling lattice for one particular view as $\sigma_{disp}^2 = \sigma_{aa}^2 \cdot |V_{disp}|$, where σ_{disp}^2 is now used in place of σ_{aa}^2. The sampling lattices of all views and colors of the display we used for experiments have a density of $1/8$, which results in $\sigma_{out}^2 \approx 1.22$.

IV. Hardware Architecture

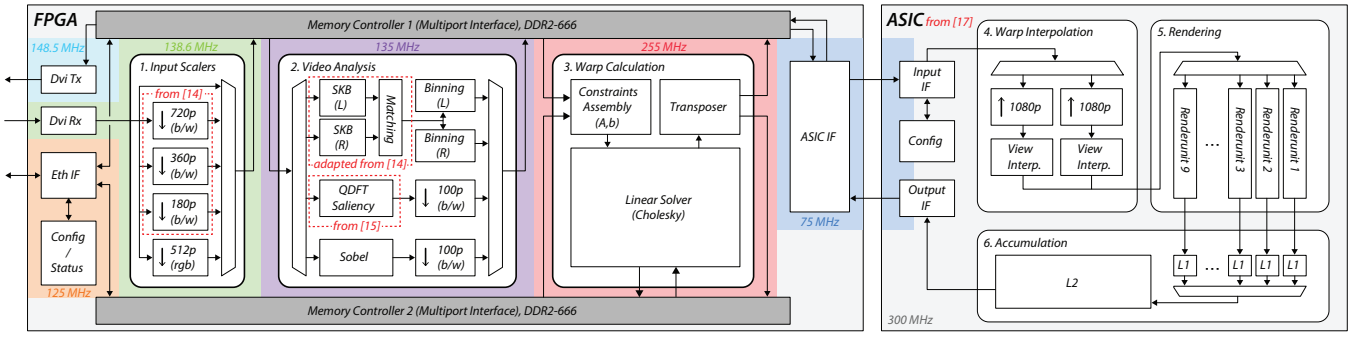

Fig. 4: Top-level blockdiagram of the MVS system. It consists of 5 main parts, where the first 3 reside on the FPGA and the remaining ones on the ASIC.

The complete MVS system shown in Figure 4 has been developed in several steps. Initial work concentrated on the design of an efficient multiview rendering core in hardware which was fabricated as a custom ASIC in 65 nm. In later stages of the project, the development was moved to the FPGA-based prototyping system Terasic DE4-530. The fabricated ASIC was connected via one of the HSMC ports using a custom extension PCB, and the missing parts have been developed and implemented on the FPGA.

Figure 4 provides a high-level block diagram, where the core view synthesis components are numbered from 1 through 6. These components are, in principle, device-independent and can be ported to other FPGA/ASIC technologies. The system processes the input according to the algorithmic flow presented in Section III: The input scaler (1) block scales the input video to the different resolutions that are required later on, and stores them in the memory. The video analysis block (2) extracts the saliency- and edge maps, and calculates the point correspondences using these scaled video frames. The warp calculation block (3) contains a constraints assembly core and a Cholesky-decomposition-based linear solver. The former is a µcode-programmable unit that builds the LS-problem matrices, and the latter is a fixed function solver. The warp interpolation stage (4) upsamples the two warps to 1080p, and interpolates them to the desired view positions. The rendering block (5) prepares a filter kernel for each input pixel/warp-vertex pair, evaluates it on the sampling lattice Λ_{disp} of the assigned view, and sends it to the accumulation stage (6) which fuses all subpixels to form the interleaved output image. The system has been designed to provide enough throughput to process 1080p S3D input video @30 Hz using 180×100 warps.

The main contributions of this paper are the complete MVS system shown in Figure 4, the programmable matrix assembly stage and the improved linear solver (3). The SKB subsystem in the video analysis block (2) has been ported from [14] and enhanced with better semantic kernels (Walsh-Hadamard functions, see Figure 3) and with feature point sorting stages ('binning' blocks in Figure 4). The saliency implementation in (2) has been taken from [15], and the rendering ASIC comprising (4-6) is the one from [17]. In the following description we will concentrate on the new components of the system, and the remaining parts will be summarized briefly.

A. Interfaces

The in- and output video streams are both transferred via standard DVI interfaces, referred to as DVI RX and TX. The DVI RX interface is configured to receive S3D video in top/bottom format at 30 Hz with a pixel clock of 138.6 MHz. This allows to conveniently mount the MVS system on any PC or laptop, and play the S3D content using a standard media player. The DVI TX interface is configured to transmit a standard 1080p stream at 60 Hz with a pixel clock of 148.5 MHz. The frame rate of 60 Hz is a requirement of the Alioscopy display we used for testing, since 1080p at 30 Hz is not a supported video standard. Since the system is designed for a throughput of 30 fps, the output of the rendering stage has to be temporally upsampled and resynchronized. This is achieved by duplicating frames, and a triple buffering scheme is employed in order to guarantee synchronized frame changes.

In the current configuration, two memory controllers interfacing to a DDR2-666 DIMM are used - providing a theoretical maximum throughput of 5'333 MByte/s each. Each controller is attached to a command-based multi-port interface which employs a simple round-robin (RR) arbitration policy. The multi-port interface works at $666.7\,\text{MHz}/4 \approx 166.7\,\text{MHz}$ and with 256 bit wide words such that the throughput is matched. Dual-clock FIFOs are employed in order to facilitate synchronization between the clock domains shown in Figure 4.

The ASIC rendering core is operating at 300 MHz in order to achieve enough throughput for 30 fps. The IO between the FPGA and the ASIC is running at a phase-synchronous clock which is four times slower (75 MHz) than the core frequency. The ASIC has three 24 bit RGB ports – two at the input and one at the output. The two input images and warps are streamed in through the input ports in an interleaved manner.

External memories and status/configuration registers can be accessed from a MATLAB environment via Ethernet in order to facilitate development and debugging.

B. Schedule and Memory Maps

A frame-pipelined architecture was chosen for this project as each hardware component has a different requirement in accessing the memory. Some of the blocks process the two images of one S3D frame sequentially, and some in parallel. Further, the solver works on transposed problems which follow column-major order as opposed to the row-major order of the image pixel streams. Also, some of the matrix arrays in the solver have to be accessed in reverse order (backwards). In such a setting, it is almost impossible to connect all blocks directly to each other. As shown in Figure 5, each hardware component (1- 6) works independently on a particular frame, resulting in an overall processing latency of five frames. The

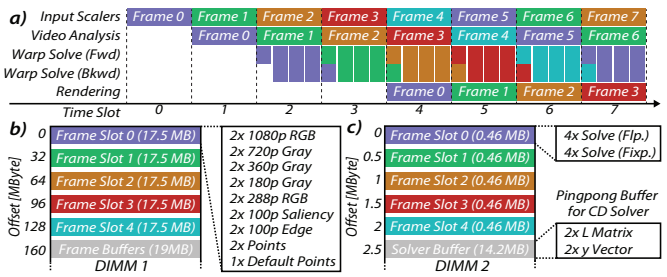

Fig. 5: Processing schedule (a), and memory maps for both DIMMS (b,c).

associated data is stored in frame slots and the rendered frames are stored in a triple frame buffer for temporal upsampling. Temporary decomposition data of the CD solver (L and $\mathbf{y}$) are stored in a ping-pong solver buffer.

The memory traffic generated by the individual components is listed in Table I. Since the DE4-530 platform provides two external memories (DIMM1 and DIMM2), the traffic has been distributed to both memory controllers for convenience. Solver related traffic is allocated to DIMM2, whereas everything else is allocated to DIMM1. This minimizes memory pattern interference, and allows to use simple controllers and multi-port interfaces with *round robin* (RR) scheduling. As shown in Table I, the theoretical bandwidth utilization is only around 30 % of the maximum bandwidth of the DDR2-666 interfaces. Even if bandwidth is in high demand, this is in a completely feasible range for modern mobile SoCs which usually have even faster interfaces such as the LPDDR3-1600 interface, providing a maximum bandwidth of 12'800 MByte/s.

TABLE I: Required memory bandwidth (including alignment overheads).

	DIMM1			DIMM2		
[MByte/s]	R	W	R+W	R	W	R+W
Scalers	-	497	497	-	-	-
Analysis	99	36	136	-	-	-
Warp Calc.	36	-	36	907	913	1'820
Rendering	398	199	597	6	-	6
DVI TX	398	-	398	-	-	-
Total	932	733	**1'664**	913	913	**1'826**

C. Stereo Video Analysis

IV-C1 SKB Subsystem. The SKB subsystem works on an image pyramid of 720p, 360p and 180p grayscale images. Correspondences are detected in three steps: *interest point detection*, *descriptor calculation* and *descriptor matching*. The first is basically a filter bank that approximates a Laplacian scale-space using box-filters. Extremal points in this space representing well localized regions in the image are identified using non-maximum suppression. In the descriptor calculation step, small support regions around these extrema are convolved with 16 *semantic kernels*, whose responses are binarized by comparing them to zero – thereby producing binary 256 bit descriptors. In the matching stage, the descriptors coming from the left and the right view are matched using a greedy, windowed matching procedure. For each descriptor coming from the left image, the Hamming distances of all correspondence candidates in a small matching window in the right image is calculated. The lowest-distance candidate is accepted if the value lies below the matching threshold. The SKB subsystem is able to extract and match up to 25 k descriptors per frame and the coordinate resolution is 1 pixel in the 720p domain. All data formats are fixed-point and have been chosen such that the performance is not compromised [14].

IV-C2 Point Correspondence Binning. The matching block can not guarantee that the correspondences are output in-order, and therefore an additional sorting stage is needed at this point. Further, several correspondences may affect the same warp-quad (the square formed by four warp vertices). We accomplish this by allocating a coordinate bin for each quad, and the correspondences are sorted into these bins on-the-fly. Note that the binning is not equal for the left and the right view, and therefore two such blocks are used. Since the SKB subsystem works in scan-line fashion, the sorting blocks only have to keep a sliding window of around 5 quad rows.

In our evaluations we observed that the number of correspondences per bin does usually not exceed 4. Therefore we align our data structure to DDR memory bursts (256 bit), and hence it is able to hold a maximum of 6 correspondences per bin (superfluous ones are discarded). This leads to a $180 \times 100 \times 256\,\text{bit} \approx 0.55\,\text{MByte}$ data structure for each of the two views. Clearly, there exist sparse data structures more memory-efficient than this (such as quad-trees), but the simplicity and regularity of this binning approach is very convenient for a hardware implementation. Further, since the warp-generation follows column-major order (Section IV-D), this burst alignment also simplifies the transposition and associated address generation in the constraints assembly stage.

IV-C3 Saliency and Edge Extraction. The saliency is calculated by first converting the RGB information into a quaternion representation, followed by a transformation to frequency space using a quaternion Fourier transform (QFT). In the frequency domain, the phase information is extracted by normalizing each quaternion with its magnitude. An inverse QFT transforms this information back into the image domain. Since the QFT is separable, one QFT can be implemented using two 2D FFTs which in turn can be split into four 1D FFTs, where two are along the rows and two are along the columns of the image. This allows to use an iterative datapath containing one 1D FFT core which performs the QFT by 4 successive applications. However, this also implies several transposition, which can be costly in terms of memory bandwidth – especially if the 2D array does not fit on-chip entirely. Therefore, the algorithm has been modified to compute a block-wise saliency, where the image is partitioned into p stripes spanning the full width, but only $1/p$-th of the height. The factor p-can be chosen depending on the on-chip cache size of the target architecture, and has been set to $p = 16$. Border artifacts are mitigated by overlapping the stripes by 25%, and high frequency noise in the saliency map is removed using a 9-tap separable box filter.

The edge maps are calculated using two 3×3 Sobel masks and the vector magnitude approximation from [49], Table 3.8. Clearly, it does not make sense to extract saliency and edge maps from the full resolution 1080p images, since the warp resolution is only 180×100. However, extracting the features from such a low resolution results in poor feature

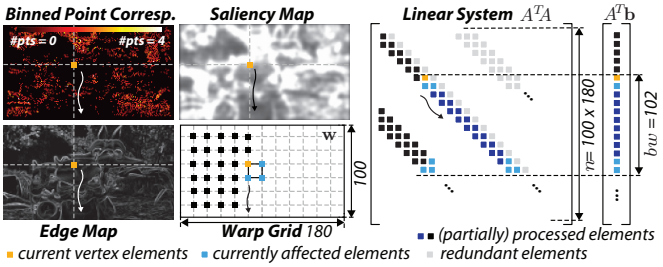

Fig. 6: Constraints assembly: A^TA, $A^T\mathbf{b}$ are built along the warp columns. Images © copyright 2008, Blender Foundation / www.bigbuckbunny.org.

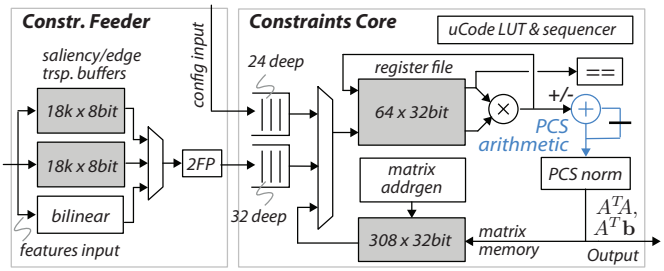

Fig. 7: Architecture of the constraints assembly block. It contains a feeder unit which loads the image features from the external memory, and feeds them to the µcode-programmable constraints core.

quality – especially in the case where FFTs are involved. Therefore, both the saliency and edge maps are extracted on an intermediate resolution aligned to powers of two (512×288). Then, they are bilinearly downsampled to the warp resolution.

D. Warp Generation

The assembly of A^TA matrices and $A^T\mathbf{b}$ vectors is sequential in nature and is not completely regular due to the sparse point correspondences. With this in mind, it makes sense to implement this using some sort of programmable architecture. Not only is the development of a program more convenient, but it also simplifies modifications and extensions of the constraints later on. Therefore we implemented a µcode-programmable mini-processor which is tailored exactly to the needs of the constraints-assembly step. It works on an image feature stream, assembles the constraints on-the-fly, and outputs finished elements of A^TA and $A^T\mathbf{b}$ to the solver.

As discussed in Section III, we use a sparse CD-based solver, since it offers a good compromise between on-chip memory resources and off-chip bandwidth for problem sizes around $n = 180 \times 100 = 18\,k$. Our previous implementation of such a solver by [18] could not reach the required performance of 120 solve/s because of latency overheads incurred by the FP pipeline. Since the CD has many data dependencies, a FP pipeline deeper than the matrix bandwidth bw (which is 102 for the 180×100 problems at hand) causes many idle cycles where the whole datapath essentially has to wait until all data has propagated to the end of the pipeline. To this end, we make the following two improvements which both aim at reducing the latency below bw and at improving the operating frequency:

- First, the modified LDL^T decomposition is used instead of using the standard LL^T decomposition. This has the advantage that no square-root operator is required [50], which has a high latency (~30 cycles). Further, no divisions are required during backward substitution.
- Second, fused FP arithmetic with partial carry save (PCS) [51] adders is used to implement the large adder tree of the scalar product in the CD – similarly as in other implementations for small, dense matrices [52].

IV-D1 Constraints Assembly. The architecture of this block is shown in Figure 7: it consists of a feeder unit that loads the image features from off-chip memory, converts them to FP, and feeds them to the µcode-programmable constraints core in *column-major* order. The saliency and edge maps do not occupy a lot of memory, and therefore are loaded at once into the on-chip memory. Note that these maps can be reused since the x- and y- direction warps of the same view are generated consecutively. The correspondence bins are loaded one after another from external memory, and the bilinear weights are determined on-the-fly before being converted to FP.

The constraints core is essentially a FP MAC unit with a 64-entry-register file, a comparator, and a matrix memory. The accumulator is capable of single-cycle accumulation thanks to PCS arithmetic. The µcode instruction set provides a couple of basic operations, such as *mult*, *mv*, etc. It also supports comparisons and jumps for flow control. Load instructions allow to load data either from the configuration and data FIFOs (*ldin*), or from pre-defined relative address offsets in the matrix memory (*ldmat*, only the address base changes from vertex to vertex). Move instructions allow to move FP values from register to register, or from register into the accumulator. Accumulated values can be written back to the matrix memory or output to the solver using store instructions. In order to use the two ports of the register file efficiently, one arithmetic instruction can be parallelized with one *load*/*store* instruction.

Once the features have been loaded, the feeder entity loads them into the data FIFO of the core and triggers the execution of the µcode program. This produces one matrix column, for which the constraints core outputs exactly six values: the main diagonal- and the four off-diagonal elements of A^TA, as well as an element of $A^T\mathbf{b}$. The execution time of the µcode program has been optimized such that it takes 107 cycles to process one warp vertex in the case where 3 point correspondences are present (solver processing time for one column is 111 cycles). If no points are present, the program only takes 38 cycles, and for 6 points it takes 176 cycles. Latency variations are averaged by the input FIFO of the solver. Constants and the weighting parameters λ can be written to the configuration FIFO which is checked in the initialization phase at the beginning of each frame.

IV-D2 Cholesky based Linear Solver. The CD computes a factorization of the form $AA^T = LDL^T$, where D is a diagonal matrix, and the diagonal elements of L are ones. In order to obtain the solution, we first have to solve $L\tilde{\mathbf{y}} = \mathbf{b}$ for $\tilde{\mathbf{y}}$, then we perform the divisions $\mathbf{y} = D^{-1}\tilde{\mathbf{y}}$ and finally get $\mathbf{x}$ by solving $L^T\mathbf{x} = \mathbf{y}$. The architecture of the CD solver is shown in Figure 8, and consists of two parts: The first part performs the decomposition and the forward substitution in interleaved manner, and the the second part performs the backward substitution. Although the backward pass is similar to the forward pass, the matrix L and the vector y have to be accessed in reversed order. Thus this task can only be executed

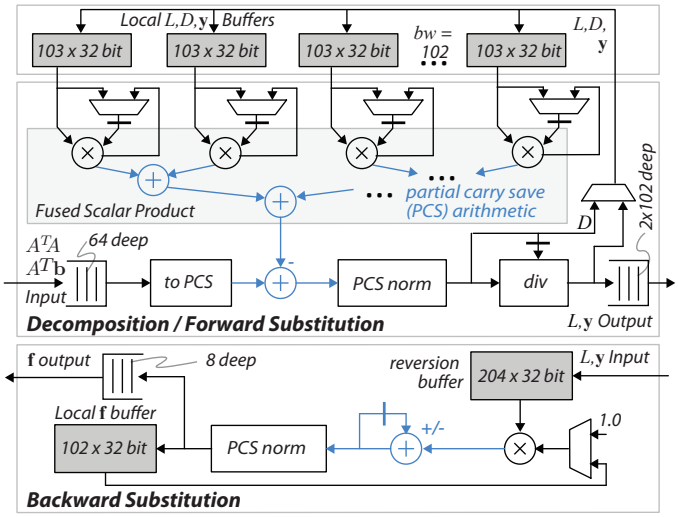

Fig. 8: Architecture of the Cholesky LDL^T solver. The forward substitution part (top) basically consists of a large scalar product, implemented using a PCS adder tree to minimize datapath latency.

once the decomposition and forward pass are finished. Using a separate unit is more convenient and allows to perform both passes of two subsequent matrices in parallel.

The decomposition stage contains a wide scalar product, which is used to calculate the inter-row products in column-major order. The parts which use PCS arithmetic have been highlighted. The decomposition is a sequential process since each element L_{ij} depends on all it's neighbours to the left. However, due to the banded shape of L, the values required to compute another column of L all lie within a window of size bw^2. These are buffered locally - together with the past bw elements of the $\mathbf{y}$ and D vectors. Parallelization is easy up to a degree corresponding to bw. Beyond that, the strong dependencies on previous results impedes further parallelization. Here, we use a scalar product width of $bw = 102$ in order to meet the throughput requirements. Together with all overheads, the solver requires 111 cycles to process one matrix column, which corresponds to $30\,\text{fps} \times 4 \times 18\,\text{k} \times 111\,\text{cycles} \approx 239.8\,\text{MCycle}$ per second. Our implementation has some margin and is clocked at 255 MHz.

IV-D3 Numerical Precision and PCS Arithmetic. The employed FP format has been tailored to the precision requirements of this application. Since we calculate coordinate values, the result should be precise to at least ~0.5 pixel so that no artifacts are visible in the rendered images. Using numerical evaluations (Figure 9), we decided to use an asymmetrically biased format (exponent is offset by +8), with 6 exponent and 24 mantissa bits, and an explicitly coded zero bit. One FP word is therefore aligned to 32 bits, and the accumulator length of the PCS adder tree is $24 + 2^6 + 1 + 7 = 96$ bit (mantissa, exponent range, sign bit and 7 overflow bits). No rounding is performed in any of the FP operators. Test syntheses revealed that 9 bits or less should be used per PCS segment such that 255 MHz can be reached on the target FPGA. With an accumulator length of 96 bits, this corresponds to 11 segments, and therefore PCS normalization requires around 13 cycles.

E. Warp Interpolation, Rendering and Accumulation

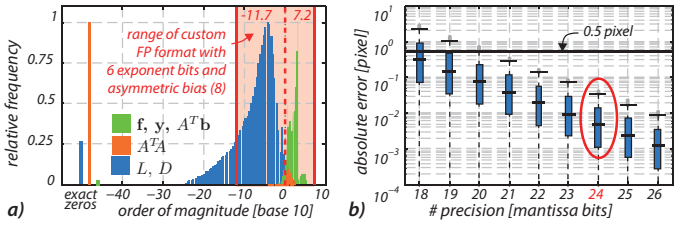

Fig. 9: The custom FP format has been defined by first fixing the exponent range, then the mantissa precision. The histograms in a) show typical magnitude distributions for the involved quantities. The boxplots in b) show the error statistics in dependency of the mantissa precision. The exponent and mantissa have been chosen to have 6 bits and 24 bits, respectively.

The two input warps $\mathbf{w}^l$, $\mathbf{w}^r$ are first upsampled bilinearly before being interpolated to the desired view positions. After view interpolation, the warp Jacobians are calculated using finite differences. Each warp coordinate is then grouped into packets (*splat jobs*) with its Jacobian, its view number and its associated input image pixel. The warp interpolation stage can deliver two splat jobs in each cycle with 8 views enabled.

Each render unit contains a filter setup stage, which iteratively prepares the filter kernels, and three rasterization units that evaluate the filter at programmable sampling points on the sampling lattice of the corresponding view. Only the required subpixels of the target image are evaluated, and the rasterizers are designed such as to evaluate one subpixel on the sampling lattice per cycle. Each render unit is able to process one splat job in four cycles, which translates into a throughput of 75 Msplat/s per second. This is sufficient to resample 1080p images at 30 fps as this amounts to 62.21 Msplat/s.

Only the subpixels required are evaluated in the render units and around 2 Gsubpixel/s need to be accumulated per color channel with eight views enabled. This corresponds to ~6.7 subpixel values per color channel and cycle. Fortunately, the large overlap among subsequent splats *of the same view* can be leveraged to reduce this number by placing small, fully-associative subpixel-caches right after the rasterizers (*Level-1* (L1) caches in Figure 4). These L1 caches reduce the required accumulations by a factor of 5.6 which means that the L2 cache now needs to accumulate around 1.2 subpixels per colour channel and cycle. The L2 cache is the actual framebuffer and is implemented as a sliding window that automatically adjusts its position depending on the incoming addresses. Assumptions on the geometric arrangement (i.e. almost rectified views) of the views allow to store only a small excerpt (25 rows) of the whole output image on chip, and therefore no external memory is required.

V. Results

A. Multiview Synthesis Results

Rendered results are depicted in Figure 10a-o), and an example for (real-time) depth-volume adjustment is shown in Figure 10p). All depicted results have been processed using our hardware system. During informal subjective tests using full-length S3D movie footage, we found that the system works well on a broad range of synthetic, as well as live action content using the same set of parameters. In general, the spatial artifacts of the IDW method are rather subtle in nature since the image texture is transformed as a whole, and

Fig. 10: a1-b6 show two examples with all feature maps of the left view and the rendered ouput. c1-c4 show four first viewing zones (starting from the left). d-e show two additional scenes where the IDW method works well. f-o4 show results with annotated warping artifacts, and p1-p2 show an example for depth-volume adjustment – the percentages indicate the scaling factor of the original baseline. The rendered images have all been grabbed from the hardware system and are in anaglyph format - use a document viewer to zoom in and view with red/blue glasses. Images (a,b,c,f,g,h,i,m,n,p) © copyright 2008, Blender Foundation / www.bigbuckbunny.org and images (d,e,j,k,l,o) © 2006, Blender Foundation / Netherlands Media Art Institute / www.elephantsdream.org.

no discontinuities are produced. This is why even rendered images with artifacts can be visually pleasing. Typical artifacts are described in more detail below, and for exhaustive and formal subjective experiments we refer to the study performed by [22], [11], which revealed that fully automatic IDW as proposed here performs at least equally well as DIBR methods.

V-A1 Spatial Artifacts Typical spatial IDW artifacts are excessively bended image regions, which are usually due to large changes of perspective, large antagonistic disparities in adjacent image regions, multiple disocclusions, or too few point correspondences. Figure 10 f-h depict cases where fore- and background have large opposite disparities. In such cases, a change of perspective would usually lead to disocclusion of large image areas, and the IDW method has to handle this by streching/compressing the transition regions.

Enough accurate point correspondences are essential for a good performance. Usual failure cases are small or thin foreground objects which do not yield enough correspondences, and therefore are 'smeared' into the background, e.g. Figure 10 c1, i-n. Although we do not perform a RANSAC filtering step, there are only very few artifacts due to false correspondences, since the smoothness constraints in the warp generation establish a certain outlier tolerance. Artifacts are sometimes visible in repetitively or homogeneously textured regions, such as in Figure 10 o. As shown in Figure 10 o4, increasing the smoothness weight λ_{sal} can help to reduce such artifacts - however at the price of increased warp stiffness.

The current implementation does not perform any blending of textures coming from left and right views. This has the advantage that less views need to be rendered, and that these views do not exhibit any ghosting artifacts due to warp misalignment. A series of four adjacent viewing zones from an 8 view setup is shown in Figure 10. Note that these views are all slightly different and enable a limited, horizontal head-motion parallax when viewed on a MAD.

V-A2 Temporal Artifacts Without any temporal consistency constraint, the warps can exhibit disturbing temporal jittering artifacts (Figure 11 a) , since the extracted image features are slightly different from frame to frame. In order to overcome this issue, a temporal regularizer is used which basically acts as a smoothing filter. However, as can be seen in Figure 11 a-b there's obviously a tradeoff to be made between jittering artifacts and over-smoothing which, for large values λ_t, can be visible as temporally lagging depth adjustments. For smaller, more reasonable values of λ_t, the effect of temporal regularization can only be noticed around fast moving image regions or at scene cuts[1].

[1]A scene-cut detector could be employed in order to alleviate this.

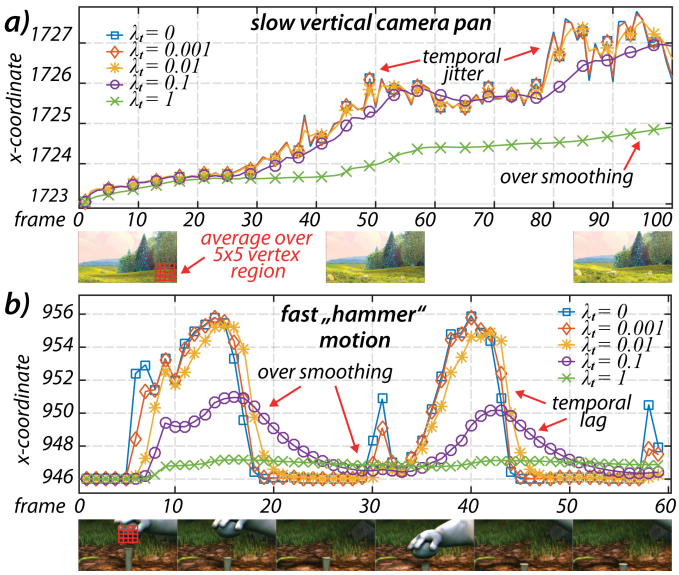

Fig. 11: The effect of temporal regularization is illustrated using a slowly moving scene (a) and a scene with fast motion (b). The plots show how the average x-coordinate of a 5×5 warp patch evolves over time. For small values of λ_t, one can observe temporal jittering artifacts, whereas for large values of λ_t we can observe over smoothing and a temporal lag in image regions with fast motion (with $\lambda_{pt} = 1$, $\lambda_{sal} = 10$, $\lambda_{sal} = 100$). Images © copyright 2008, Blender Foundation / www.bigbuckbunny.org.

V-A3 Warp Resolution and Parameterization The impact of the warp resolution is illustrated with the example in Figure 12 b,c. Clearly, there is a tradeoff to be made: while small resolutions are computationally preferable, they deliver poor performance since the essential scene geometry is not captured accurately enough. High-resolution warps perform visually better, but they are also much more expensive to compute. A resolution around 180×100 has therefore been found to be a good compromise between visual quality and computational complexity. For higher fidelity, warps in the order of 360×200 could be used in future designs.

There is a range of good parameter combinations and the specific choice is up to the user. We found that setting the (relative) weights to $50 \times \lambda_t = 1 \times \lambda_{pt}/c_{pt} = 0.1 \times \lambda_{sal} = 0.001 \times \lambda_{edge} = 1$ provides good results on most content, where the factor $c_{pt} = (180 \times 100)/n_{pts}$ accounts for the fact that all constraints except the point correspondences are defined using 2D feature maps.

B. Functional Characteristics and Performance

Table II provides a summary of the resource utilization of the multiview system. The input video is in top/bottom tiled S3D format with 1080p resolution per view and 30 fps. The output is a 1080p 60 Hz video stream, where the individual frames contain 8 interleaved images for a MAD (currently for the Alioscopy HD 47" LV, but can be adapted to similar displays). The system is able to convert S3D video at 30 fps which allows for real-time operation. Further, the warp generation parameters, number of views (up to 9), the position of the views on the normalized baseline and the display parameters (interleaving pattern and filter parametrization) are fully programmable at runtime. This allows, for example, to adjust the displayed depth volume in real time (linear scaling of the camera baseline) as shown in Figure 10p). Further, our system

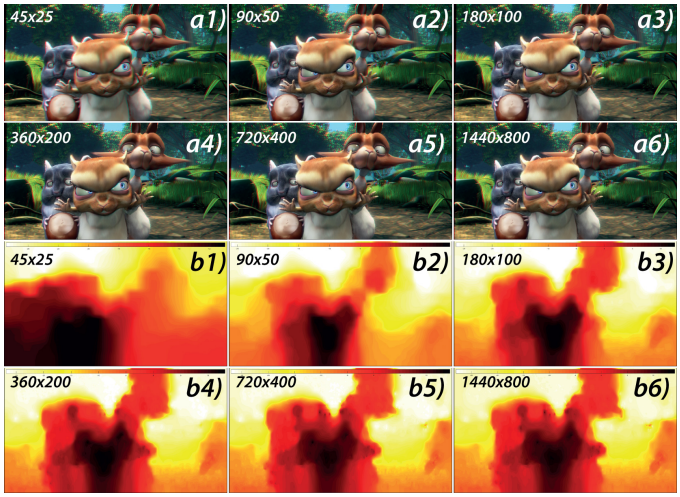

Fig. 12: a1-6 example rendering using different warp resolutions. b1-6 are the corresponding x-warps after bilinear upsampling. Results are in anaglyph format, use a document viewer to zoom in and view with red/blue glasses. Images © copyright 2008, Blender Foundation / www.bigbuckbunny.org.

does not assume completely rectified input content. In fact, a Keystone [1] distortion of up to ±11 pixel can be tolerated. Note that the same rendering architecture implemented in the ASIC could easily support quad-full-HD (3840 × 2160) output resolution at the same framerate by increasing the I/O bandwidth of the L2 cache. Only practical I/O limitations (no flip-chip packaging) have prevented us from doing so.

C. Estimated ASIC Complexity and Power Consumption

In order to get an idea of the area and power requirements of a completely integrated system, the major blocks have been ported to 65 nm and 28 nm technology. The results are shown in Table III and with the exception of the saliency estimation block, all results are based on gate-level synthesis and simulation (Synopsys DC, Mentor Modelsim and Synopsys Power Analyzer). Due to many FPGA macro dependencies (FFT cores), the values for the saliency block have been estimated using the FPGA and ASIC results of the Cholesky solver (which has similar characteristics in terms of DSP, LUT and register usage). Note that the estimates do not include the infrastructure and interface blocks from the FPGA part.

It is important to keep in mind that the power consumption due to the external memory accesses is not negligible. Using the estimation procedure described in [32], we calculated that an LPDDR3 PC6400 memory subsystem would consume around 700 mW supplying a bandwidth of 3.5 GByte/s – including memory controller, PHY, LPDDR3 component and I/O switching power. Adding this to the power estimate for 28 nm results in a total power consumption of ≈ 1.3 W (not accounting for additional I/O and infrastructure circuitry). Note that at this technology node, the power consumption of the external memory interface is even larger than the power consumption of the hardware accelerator.

D. Comparison with Related Work

There are relatively few published real-time systems which implement a complete MVS pipeline with video analysis and rendering units. Both parts are usually treated as separate subproblems in the literature. Therefore, we first compare our

system against complete systems, and then we compare the two subsystems with related depth image based architectures.

V-D1 Complete Systems. Complete real-time systems with similar functionality are the work by [35] and [36] (see Table IV top). [35] implemented a system based on a dual processor workstation with two Intel Xeon 5690 CPU's and two NVIDIA GTX 590 graphics cards. It has almost the same performance as ours and is able to synthesize 8 interleaved views from 1080p S3D content at 24 fps. Clearly, the size of this system, the cost, and power consumption make it unsuitable for integration into consumer devices. Similarly to our system – their rendering engine could also support larger 4 k displays without any decrease in performance, since the views are rendered at higher resolution internally. In [36] a single-view synthesis pipeline on a Stratix III FPGA is implemented, which is able to produce depth-adjusted S3D content generated from 1080p S3D input video at 60 fps. The system synthesizes one virtual view and bypasses one of the input views. Since no detailed FPGA results where published, we compare their complexity estimates in 250 nm CMOS technology against our ASIC estimates in Table IV. At first sight the implementation by [36] looks to be almost a factor 10 more area-efficient. This has several reasons. First of all, their implementation only renders one virtual view and not up to nine views as in our work and [35]. They also make the assumption that the input video is perfectly rectified, i.e. there is no y-disparity between the left and right image, which simplifies all 2D-problems to 1D-problems in the depth estimation and rendering parts. Yet such rectification is not easily achieved, and therefore systems should be designed to tolerate y-disparities (Keystone [1]). For example, our system is designed to support up to ± 11 pixels of y-disparity.

TABLE II: Physical characteristics of the multiview system. The FPGA is a Stratix IV (EP4SGX530KH40C2), and the ASIC has been fabricated in 65 nm technology. Frequency values represent the current parametrisation.

FPGA	Logic			RAM		Freq.
	LUTs	Regs	DSPs	LUTs	9/144K	[MHz]
Scalers	4.1 k	7.1 k	24	2.3 k	10/0	138.6
Analysis	43.3 k	47.3 k	177	2.7 k	434/24	135
1 SKB Core	11 k	11.3 k	32	0	11/0	135
Matching	6.2 k	2.8 k	1	0.3 k	249/4	135
1 Binner	0.5 k	0.9 k	0	0.3 k	31/0	135
Saliency	8.3 k	13.2 k	96	1.1 k	95/0	135
Sobel	0.3 k	0.3 k	0	0	2/0	135
Warp Calc.	51.9 k	67.4 k	420	0.7 k	148/0	255
Assembly	4.8 k	6.6 k	8	0.2 k	38/0	255
Solver	45.8 k	58.9 k	412	0.5 k	104	255
Transposer	1.3 k	1.9 k	0	0	6/0	255
Subtotal	99.3 k	121.8 k	621	5.7 k	592/24	↑
IO/Infra.	22.1 k	35.5 k	0	15.7 k	14/4	N/A
Total	**121.4 k**	**157.3 k**	**621**	**21.4 k**	**606/28**	↑
ASIC	Logic		SRAM			Freq.
	[mm^2]	[kGE]	[MBit]	[mm^2]	[kGE]	[MHz]
IO/Infra.	0.28	195	-	-	-	75, 300
Warp Interp.	0.314	218	0.76	1.023	710	300
Rendering	2.317	1'609	-	-	-	300
Renderunit	0.257	179	-	-	-	300
Accumulator	0.385	267	3.6	5.436	3'775	300
Total	**3.296**	**2'289**	**4.36**	**6.459**	**4'485**	↑

TABLE III: Estimated complexity and power consumption[2] of the major blocks in 65 nm (LVT, TT @1.3V 25°) and 28 nm (LVT, TT @1V 25°).

ASIC Estimates	Logic		SRAM		Power
	[mm^2]	[kGE]$^\diamond$	[mm^2]	[kGE]$^\diamond$	[mW]
Scalers*	0.136	94	0.406	282	19
Analysis*	1.099	763	9.414	6'537	343
Warp Calc*	1.841	1'278	1.794	1'245	791
MADMAX ASIC†	3.296	2'289	6.459	4'485	752
Total (65 nm)	**6.372**	**4'424**	**18.073**	**12'549**	**1'905**
Total (28 nm)*	**1.444**	**3'966**	**6.929**	**15'069**	**605**

* Gate level synthesis and simulation † Measurements
$^\diamond$ 1 GE = 1.44 µm^2(65 nm), 1 GE = 0.364 µm^2 (28 nm)

V-D2 Video Analysis and Depth Estimation Cores. In DIBR frameworks, depth estimators such as the implementations by [41], [40], [39], [37], [38] can be viewed as the equivalent part of the video analysis block in an IDW-based system. A quantitative comparison is shown in Table IV in the middle. Note that the design by [37] uses a global, high-quality technique which is extremely memory bound, and therefore not suited for real-time frame rates. The other designs are all local methods using either a variation of the census-transform [38], [39], a combination of absolute differences and the census-transform [40], or normalized cross-correlation [41]. These local methods are more attractive for real-time systems due to their higher throughput and lower bandwidth requirements. However, this comes at the cost of more artifacts in the depth maps. Among the listed designs, [41] is the most recent and is best suited for a 1080p MVS system. In fact, it implements a hardware friendly version of the algorithm used in Riechert's system [35], and is thus tailored to the needs of MVS.

Note that there are several reasons why it is difficult to directly compare the video analysis stage of our IDW system with related depth extraction cores, although both the image warps and depth maps essentially capture the scene geometry. First of all, the depth maps and image warps used in DIBR and IDW lead to inherently different artifacts which are not similarly perceived. Also in terms of hardware complexity a direct comparison is difficult due to the fact that most DIBR methods are local methods, whereas IDW is a global method requiring an LS-solve. In addition, some of the methods use internal subsampling in order to be more efficient, and convert the calculated maps to full resolution in a final post-processing step. E.g. [41] work with 4× subsampling, and our method works on 180×100 warps which are later upsampled to 1080p in the MVS chip.

V-D3 View Synthesis Cores. A comparison of the rendering ASIC with related DIBR rendering architectures is given in Table IV at the bottom. [44], [46] and [45] develop single (non-interleaved) view synthesis engines with performances of up to 1080p@32.4 fps, 1080p@94 fps and 4 k@216 fps, respectively. In contrast, [42] and [43] develop architectures which are able to directly render interleaved views. As we can

[2]For comparison, the GPU subsystems of Apple's A7 and NVIDIA's Tegra K1 occupy around 21 mm^2 and 40 mm^2, respectively, and the platform power consumption (with display) of an A7-based tablet is ~6 W when running a graphics benchmark (www.fool.com/investing/general/2014/06/04/just-how-big-is-nvidia-corporations-tegra-k1.aspx, www.anandtech.com/show/7460/3).

TABLE IV: Comparison with other view synthesis systems, video analysis and rendering cores. The MBit values represent block ram bits in the case of FPGA designs, and SRAM bits (macros) in the case of ASIC designs.

Full Systems	Type	Input	Maps	Output	Technology	LUT	DSP	Reg	Ram [MBit]	Ext Bw [GB/s]	f [MHz]	Fps
This work	IDW	2×1080p	2×100p	8mix 1080p	ASIC 65 nm		4'424 kGE*		10.5*	3.1‡	135-300	30
Liao [36]	DIBR	2×1080p	2×1080p	2×1080p†	ASIC 250 nm		470 kGE*		1.27*	0	?	60
Riechert [35]	DIBR	2×1080p	2×1080p	8mix 1080p	GPU+CPU	2×Xeon 5690		2×GTX 590		?	?	24
Video Analysis	Type	Input	Map Output (max disp)		Technology	LUT	DSP	Reg	Ram [MBit]	Ext Bw [GB/s]	f [MHz]	Fps
This work	Warp	2×1080p	2×100p (128)		Stratix IV	142.8 k	621	157.3 k	6.56	3.1‡◇	135-255	30
Werner [41]	Depth	2×1080p	2×1080p (?)		Stratix V	70.7 k	625	132.9 k	3+6.5□	1.6◇	150	30
Akın [40]	Depth	2×XGA	1×XGA (128)		Virtex 5	48 k	0	43 k	2.3+	0.57	190	60
Zhang [39]	Depth	2×XGA	1×XGA (64)		Stratix III	80.6 k	257	94.9 k	3.77	?	100	60
Perez [37]	Depth	2×720p	1×720p (96)		Virtex 5	23.7 k*	120*	17 k*	0*	19.2+	200*	2.6
Chang [38]	Depth	2×CIF	1×CIF (64)		ASIC 90 nm		562 kGE		0.17	0.06+	95	42
View Synthesis	Type	Input	Maps	Output	Technology	LUT	DSP	Reg	Ram [MBit]	Ext Bw [GB/s]	f [MHz]	Fps
This work	IDW	2×1080p	2×100p	8mix 1080p	ASIC 65 nm		2'289 kGE		4.36	0	300	33.1
Tsung [45]	DIBR	?	?	1×4 k	ASIC 40 nm		1'416 kGE		0.16	?	240	216
Horng [44]	DIBR	2×1080p	2×1080p	1×1080p	ASIC 90 nm		269 kGE		0.55	0.96+	200	32.4
Chang [46]	DIBR	2×1080p	2×1080p	1×1080p	ASIC 90 nm		143 kGE		0.44	1.27+	200	94
Fan [43]	DIBR	?	?	8mix SXGA	ASIC 180 nm		104 kGE×		0.008	0	71.4	29.1
Chen [42]	DIBR	1×360p°	1×360p°	9mix 1080p	Cyclone III	1.4 k△	0	1.3 k	0.47	0	148.5	60

+ Calculated using published values. ◇ Including bandwidth for S3D bypassing □ ROM + RAM △ 4-input LUTs × Including SRAM
† One bypassed view, one rendered view * Estimates, not accounting for memory controller and other I/O infrastructure ° 640×360
‡ DVI TX bandwidth due to temporal upsampling has been omitted, since 30 fps could be directly output from the rendering core.

see, the hardware complexities of these DIBR architectures vary significantly. A main reason for this is that the designs [46], [43], [42] make the same assumptions as [36] and work with perfectly rectified content. This allows to collapse the 2D warping operations to 1D problems which can be implemented very efficiently with streaming architectures. Additional reasons for the varying complexity are the different image and depth map resolutions.

When comparing our chip with these related DIBR designs we observe that it is has much larger logic and SRAM area than all other designs. A main reason for this is that we use a forward mapping algorithm, and its computational complexity is defined by the input image resolution. In terms of throughput, our design could therefore easily generate interleaved output images up to the point where each subview has 2 MPixel resolution (interleaved 4 k would therefore be possible as well). Only practical I/O limitations prevented us from doing so. Also, we do not make the assumption of perfectly rectified content, which would allow us to simplify the rendering architecture significantly as well. The large SRAM area is due to the warp (0.76 MBit) and frame buffers (3.6 MBit). The former has been incorporated for convenience during development and debugging, and is not strictly required since the design works in scan-line fashion. The latter has been added such that no off-chip memory is required.

VI. Limitations and Extensions

Although all three stages (analysis, warp solver, and rendering) seem to have similar complexity, the computational bottleneck of IDW-based MVS is clearly the linear solver since its complexity does not scale as well as for the other parts when moving to higher resolutions. There are several alternative approaches that could be explored for improving the efficiency of the warp solver stage such as using an iterative CG solver which requires only on-chip resources as shown in [32]. The system has – in principle – already enough throughput to realize a 4 k system. Modifications only have to be made at the output of the rendering subsystem. As proposed in [53], the rendering part could be optimized further by performing the large part of the display anti-aliasing using a *pre-filter*.

An additional RANSAC step could improve the quality of the point correspondences – yet we found that this is not required on the vast majority of S3D content (only repetitive and homogeneous textures sometimes have issues). Since most IDW pipelines are very similar, the system could in principle be extended to support other applications such as video retargeting [19], non-linear disparity mapping [21], retargeting of S3D content [20], or optimum multiview content creation methods [34]. The main part that would need modifications is the assembly of the LS system since these applications basically just differ in the way the constraints are assembled. This underlines the need for a programmable warp assembly stage we have implemented.

VII. Conclusions

To our knowledge, this is the first hardware implementation of a complete, IDW-based MVS system. The only other published complete system comprising similar functionality in terms of video analysis and MVS has been implemented on a high-end workstation. When comparing subcomponents of our system to related DIBR-based modules, we observe that some of these DIBR components are more compact in terms of logic and SRAM area. However, these comparisons should be interpreted with caution, since the involved algorithms are completely different and the synthesis quality has not been part of this comparison.

Our system is able to synthesize high quality, full-HD content with up to 8 views in real-time, and supports a keystone distortion of up to $\pm$ 11 pixels. Further, no inpainting

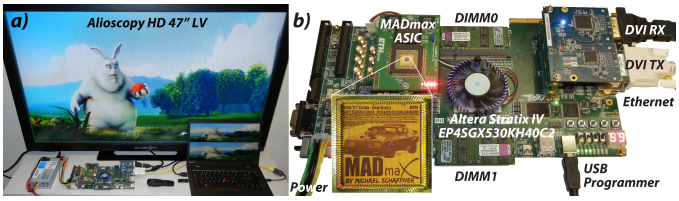

Fig. 13: a) photograph of the running system, and b) close up of the hardware prototype, which can be mounted as an external display with video format 1920x2160@30Hz on any pc or laptop with DVI or HDMI capability. Image © copyright 2008, Blender Foundation / www.bigbuckbunny.org.

steps are required due to the use of IDW. The current test-bed is designed using an FPGA platform combined with a custom ASIC, and our results show that a monolithic integration of the developed hardware IP into a SoC fabricated in 28 nm is completely feasible. With an estimated power consumption of ~1.3 W the hardware accelerator enables portable and energy efficient MVS, which are both essential properties when considering a deployment in consumer electronic devices.

REFERENCES

[1] B. Mendiburu, "3D Movie Making," *Focal Express*, 2009.

[2] A. Smolic et al., "Three-Dimensional Video Postproduction and Processing," *PIEEE*, vol. 99, no. 4, pp. 607–625, 2011.

[3] C. Zhu et al., *3d-tv System with Depth-image-based Rendering*. Springer, 2014.

[4] N. A. Dodgson, "Optical Devices: 3D without the Glasses," *Nature*, vol. 495, no. 7441, pp. 316–317, mar. 2013.

[5] J. Konrad and M. Halle, "3-D Displays and Signal Processing," *IEEE SPM*, vol. 24, no. 6, pp. 97–111, 2007.

[6] A. Boev et al., "Signal Processing for Stereoscopic and Multi-View 3D Displays," in *Handbook of Signal Processing Systems*. Springer New York, 2013, pp. 3–47.

[7] D. Tian et al., "View Synthesis Techniques for 3D Video," *Proc. SPIE*, vol. 7443, p. 74430T, 2009.

[8] A. Smolic et al., "Disparity-Aware Stereo 3D Production Tools," in *IEEE CVMP*, 2011, pp. 165–173.

[9] M. Tanimoto et al., "Free-Viewpoint TV," *IEEE SPM*, vol. 28, no. 1, pp. 67–76, jan. 2011.

[10] M. Tanimoto et al., "View Synthesis Algorithm in View Synthesis Reference Software 3.5 (VSRS3. 5) Document M16090, ISO/IEC JTC1/SC29/WG11 (MPEG)," 2009.

[11] N. Stefanoski et. al, "Automatic View Synthesis by Image-Domain-Warping," *IEEE TIP*, vol. 22, no. 9, pp. 3329–3341, 2013.

[12] G. Wolberg, "Digital Image Warping," *IEEE Computer Society press*, vol. 3, 1990.

[13] P. Ndjiki-Nya et al., "Depth Image Based Rendering with Advanced Texture Synthesis," in *IEEE ICME*, July 2010.

[14] M. Schaffner et al., "A complete real-time feature extraction and matching system based on semantic kernels binarized," in *VLSI-SoC: At the Crossroads of Emerging Trends*, ser. IFIP Advances in Information and Communication Technology. Springer International Publishing, 2015, vol. 461, pp. 144–167.

[15] P. Greisen et al., "Algorithm and VLSI Architecture for Real-time 1080P60 Video Retargeting," in *ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, ser. EGGH-HPG'12, 2012, pp. 57–66.

[16] P. Greisen et al., "Analysis and VLSI Implementation of EWA Rendering for Real-Time HD Video Applications," *IEEE TCSVT*, vol. 22, no. 11, pp. 1577–1589, nov. 2012.

[17] M. Schaffner et al., "MADmax: A 1080p Stereo-to-Multiview Rendering ASIC in 65 nm CMOS based on Image Domain Warping," in *ESSCIRC*, 2013, pp. 61–64.

[18] P. Greisen et al., "Evaluation and FPGA Implementation of Sparse Linear Solvers for Video Processing Applications," *IEEE TCSVT*, vol. 23, no. 8, pp. 1402–1407, 2013.

[19] M. Rubinstein et al., "A Comparative Study of Image Retargeting," in *ACM TOG*, vol. 29, no. 6, 2010, p. 160.

[20] S. Kopf et al., "Warping-Based Video Retargeting for Stereoscopic Video," *IEEE ICIP*, 2014.

[21] M. Lang et al., "Nonlinear Disparity Mapping for Stereoscopic 3D," *ACM TOG*, vol. 29, no. 4, p. 75, 2010.

[22] K. Müller et al., "Report of subjective test results from the call for proposals on 3d video coding," ISO/IEC JTC1/SC29/WG11, Geneva, Switzerland, Tech. Rep. MPEG2011/N12347, Nov. 2011.

[23] N. Stefanoski et al., "Image Quality vs. Rate Optimized Coding of Warps for View Synthesis in 3D Video Applications," in *IEEE ICIP*, Sept 2012, pp. 1289–1292.

[24] R. Hartley and A. Zisserman, "Multiple view geometry," *Cambridge university press Cambridge, UK*, 2000.

[25] M. Calonder et al., "BRIEF: Computing a Local Binary Descriptor Very Fast," in *IEEE TPAMI*, vol. 34, no. 7, 2012, pp. 1281–1298.

[26] F. Zilly et al., "Semantic Kernels Binarized - A Feature Descriptor for Fast and Robust Matching," in *CVMP*, nov. 2011, pp. 39–48.

[27] L. Itti et al., "A Model of Saliency-based Visual Attention for Rapid Scene Analysis," *IEEE TPAMI*, vol. 20, no. 11, 1998.

[28] H. Kim et al., "Saliency Prediction on Stereoscopic Videos," *IEEE TIP*, vol. 23, no. 4, pp. 1476–1490, April 2014.

[29] C. Guo et al., "Spatio-Temporal Saliency Detection Using Phase Spectrum of Quaternion Fourier Transform," in *IEEE CVPR*, June 2008, pp. 1–8.

[30] A. Björck, "Numerical Methods for Least Squares Problems," *SIAM*, 1996.

[31] Y. Saad, "Iterative Methods for Sparse Linear Systems Second Edition," *SIAM*, 2003.

[32] M. Schaffner et al., "DRAM or no-DRAM? Exploring Linear Solver Architectures for Image Domain Warping in 28 nm CMOS," in *DATE*, 2015.

[33] M. Zwicker et al., "Antialiasing for Automultiscopic 3D Displays," in *Eurographics conference on Rendering Techniques*, 2006, pp. 73–82.

[34] A. Chapiro et al., "Optimizing Stereo-to-Multiview Conversion for Autostereoscopic Displays," *Computer Graphics Forum*, vol. 33, no. 2, 2014.

[35] C. Riechert et al., "Fully Automatic Stereo-to-Multiview Conversion in Autostereoscopic Displays," *The best of IET and IBC*, vol. 4, no. 8, p. 14, 2012.

[36] C. Liao et al., "Stereo Matching and Viewpoint Synthesis FPGA Implementation," in *3D-TV System with Depth-Image-Based Rendering*. Springer New York, 2013.

[37] J.M. Perez et al., "High Memory Throughput FPGA Architecture for High-Definition Belief-Propagation Stereo Matching," in *SCS*, Nov 2009, pp. 1–6.

[38] N.Y.-C. Chang et al., "Algorithm and Architecture of Disparity Estimation With Mini-Census Adaptive Support Weight," in *IEEE TCSVT*, vol. 20, no. 6, June 2010, pp. 792–805.

[39] Lu Zhang et al., "Real-time High-definition Stereo Matching on FPGA," in *ACM/SIGDA FPGA*. ACM, 2011, pp. 55–64.

[40] A. Akın et al., "Dynamically Adaptive Real-Time Disparity Estimation Hardware using Iterative Refinement," *Integration, the VLSI Journal*, vol. 47, no. 3, pp. 365–376, 2014.

[41] M. Werner et al., "Hardware Implementation of a Full HD Real-Time Disparity Estimation Algorithm," *IEEE TCE*, vol. 60, no. 1, pp. 66–73, February 2014.

[42] Hsin-Jung Chen et al., "Real-time Multi-View Rendering Architecture for Autostereoscopic Displays," in *IEEE ISCAS*, May 2010, pp. 1165–1168.

[43] Yu-Cheng Fan et al., "DIBR Based Multi-View Generator Circuit and Chip Design," in *ICICS*, Dec 2013.

[44] Y. Horng et al., "Vlsi architecture for real-time hd1080p view synthesis engine," *IEEE TCSVT*, vol. 21, no. 9, 2011.

[45] Pei-Kuei Tsung et al., "A 216fps 4096x2160p 3DTV Set-Top Box SoC for Free-Viewpoint 3DTV Applications," in *IEEE ISSCC*, 2011.

[46] Fu-Jen Chang et al., "A 94fps View Synthesis Engine for HD1080p Video," in *IEEE VCIP*, 2011.

[47] H. Hirschmüller and D. Scharstein, "Evaluation of Cost Functions for Stereo Matching," in *IEEE CVPR*, 2007, pp. 1–8.

[48] T. Akenine-Moller et al., "Real-Time Rendering," *AK Peters*, 2008.

[49] H. Kaeslin, "Top-Down Digital VLSI Design, from VLSI Architectures to Gate-Level Circuits and FPGAs," *Morgan Kaufmann*, 2014.

[50] Y. Depeng et. al., "Compressed Sensing and Cholesky Decomposition on FPGAs and GPUs ," *Parallel Computing*, vol. 38, no. 8, 2012.

[51] F. De Dinechin et. al., "An FPGA-Specific Approach to Floating-point Accumulation and Sum-of-products," in *ICECE Technology*, 2008.

[52] S. Demirsoy and M. Langhammer, "Cholesky Decomposition Using Fused Datapath Synthesis," in *ACM/SIGDA FPGA*, 2009.

[53] M. Schaffner et al., "Efficient Image Resampling for Multiview Displays," in *IEEE ICASSP*, 2013.

Michael Schaffner received his BSc. and M.Sc. degrees from the Swiss Federal Institute of Technology Zurich, Switzerland, in 2009 and 2012. He is currently pursuing the Ph.D. degree with ETH Zurich. He has been a Research Assistant with the Integrated Systems Laboratory and with Disney Research Zurich, since 2012. His research interests include digital signal processing, video processing, and the design of very large scale integration circuits and systems. Michael Schaffner received the ETH Medal for his Diploma thesis in 2013.

Frank K. Gürkaynak obtained his BSc. and M.Sc. degrees from Electrical and Electronical Engineering Department of the Istanbul Technical University and his Ph.D. degree from ETH Zurich. He is employed by the Microelectronics Design Center of ETH Zurich and his research interests include design of VLSI systems, cryptography, and energy efficient processing systems.

Pierre Greisen received the M.Sc. degree from ETH Zurich in 2007, a Master's degree from Ecole Centrale Paris (ECP) in 2007, and the Dr.sc. degree from ETH Zurich in 2013. From 2009 to 2013, he has worked as research assistant at the Integrated Systems Laboratory (IIS) and at Disney Research Zurich in the field of digital signal and video processing. Since 2013, Dr. Greisen is working at the Goodyear Innovation Center Luxembourg in the area of data science and video processing. In 2013, he was awarded the ETH medal for his PhD thesis.

Hubert Kaeslin received both the M.Sc. and the Ph.D. degree in electrical engineering from ETH Zurich, Switzerland, in 1978 and 1985 respectively. Since 1989 he has been heading the Microelectronics Design Center of ETH Zurich which taped out roughly 400 circuit designs under his supervision over the past 25 years, both for research and educational purposes. These activities have led to the publication of two textbooks on digital VLSI design. His professional interests include dedicated VLSI architectures, energy-efficient circuits, hardware description languages, synchronous and self-timed (GALS) clocking methodologies, electronic design automation, semiconductor technology, digital signal processing, IT security, graph theory, and visual formalisms.

Dr. Kaeslin has authored or co-authored more than 75 papers in reviewed journals and conference proceedings. He is Senior Member IEEE and has been awarded the title of professor by ETH in 2010.

Luca Benini is the chair of digital Circuits and systems at ETHZ and a Full Professor at the University of Bologna. He has served as Chief Architect for the Platform2012/STHORM project in STmicroelectronics, Grenoble. He has held visiting and consulting researcher positions at EPFL, IMEC, Hewlett-Packard Laboratories, Stanford University. Dr. Benini's research interests are in energy-efficient system design and Multi-Core SoC design. He is also active in the area of energy-efficient smart sensors and sensor networks for biomedical and ambient intelligence applications. He has published more than 700 papers in peerreviewed international journals and conferences, four books and several book chapters. He is a Fellow of the IEEE and a member of the Academia Europaea.

Dr. Aljosa Smolic joined Disney Research Zurich in 2009, as Senior Research Scientist and Head of the Advanced Video Technology group. Before he was Scientific Project Manager at the Fraunhofer Heinrich-Hertz-Institut (HHI), Berlin, also heading a research group. He has been involved in many national and international research projects, where he conducted research in various fields of video processing and visual computing, and published more than 100 referred papers in these fields. He received the Dipl.-Ing. Degree in Electrical Engineering from the Technical University of Berlin, Germany in 1996, and the Dr.-Ing. Degree in Electrical Engineering and Information Technology from Aachen University of Technology (RWTH), Germany, in 2001. He is Associate Editor of IEEE Trans. on Image Processing, Area Editor for Signal Processing: Image Communication and served as Guest Editor for the Proceedings of the IEEE, and other scientific journals. He has been involved in MPEG standardization for 3D video as group leader and one of the Editors of the Multi-view Video Coding (MVC) standard. Further, he serves as Associate Lecturer at ETH Zurich teaching full lecture courses on Multimedia Communications.