Correspondence

Accelerating Image-Domain-Warping Virtual View Synthesis on GPGPU

Ronggang Wang, Jiajia Luo, Xiubao Jiang, Zhenyu Wang, Wenmin Wang, Ge Li, and Wen Gao

Abstract—**The image-domain-warping (IDW) method can effectively create high-quality virtual views. However, the IDW algorithm is very complex, and the software implementation for this method is far from real-time. In this paper, we propose an IDW-based view synthesis acceleration method on general-purpose computing on a graphics processing unit (GPGPU). Our method makes two main contributions. First, at the algorithm level, we employ FAST for sparse disparity estimation and adopt the successive over-relaxation iterative method to calculate warps. Second, at the platform level, two computation-intensive modules (data extraction and view synthesis) in IDW are offloaded to GPU using efficient data-level parallelism strategies. Experimental results demonstrate that our proposed acceleration method can speed up the original IDW algorithm by more than 110x, and HD stereo three-dimensional video can be converted to 8-view 4 K video (each view has an approximate 720P resolution) in real-time on a hybrid CPU + GPU (NVIDIA GTX980) platform.**

Index Terms—**General-purpose computing on a graphics processing unit (GPGPU), image-domain-warping (IDW), real-time, stereo 3D (S3D) to N-view, virtual view synthesis.**

I. Introduction

Multi-view autostereoscopic display (MAD) is becoming increasingly available nowadays. With multiple views as input, MAD could free viewers from wearing 3D glasses. However, content production for MADs is still a challenging task [5]. The simplest way involves direct capture, compression, and transmission of the required views. Even with highly efficient multi-view coding methods [6], [7], however, the cost of data transmission remains very high. A more practical approach is to capture $M < N$ views and then synthesize the remaining views at the receiver side [1]. Over time, stereo 3D (S3D) has become a dominant 3D content format. Consequently, real-time conversion from S3D to N-view would be a promising technology for MADs. The conversion depends on a key technology called virtual view synthesis.

Depth-image-based rendering (DIBR), the most popular view synthesis method, has been studied for many years [11]. The quality of a virtual view synthesized by DIBR depends heavily on the accuracy of depth data. However, a high-quality depth map is difficult to estimate and is usually generated through an interactive process. DIBR tends to cause annoying holes in synthesized views due to occlusion. Although many hole-filling algorithms [e.g., 2, 3, and 4] have been proposed, observable artifacts still remain around the edges of foreground objects. Even worse, hole-filling adds a heavy computation burden to DIBR for real-time applications.

Image-domain-warping (IDW) based view synthesis can address these issues of DIBR. IDW relies on sparse disparities rather than dense depth maps [8]. With the extracted image features, sparse disparities can be estimated and then be used to calculate an image domain deformation function, or a warp. Finally, pixels of input views are warped to new positions in virtual views. Unlike DIBR, this entire process can work automatically and produce visually plausible results. Additionally, warp continuity eliminates the occurrence of holes. However, IDW remains computation-intensive and is still far from being used in real-time applications [9]. Fortunately, this method has a high degree of parallelism and is suitable for acceleration with many-core platforms, such as graphics processing units (GPUs).

General-purpose computing on a graphics processing unit (GPGPU) has recently emerged as a new programming mode to accelerate massive data processing [20]. Benefiting from thousands of processing elements, GPU could process parallel workloads efficiently via single instruction multiple data (SIMD) [21]. Compute unified device architecture (CUDA) and open computing language (OpenCL) are two prevalent architectures supporting parallel computing. The former only allows programming on NVIDIA GPU devices, whereas the latter can work on heterogeneous platforms [12], [19].

In this paper, we propose a real-time IDW-based S3D to 8-view conversion system on a hybrid GPGPU platform. IDW is accelerated significantly at both the algorithm and platform levels.

At the algorithm level, firstly, we substitute FAST [15] for SIFT [13] to extract sparse disparity data. Although SIFT exhibits superior matching precision, it suffers from expensive computations [14]. In the S3D to N-view scenario, FAST is a better option for extracting image features efficiently with a slight sacrifice of visual quality. Secondly, when calculating warps, we utilize the successive over-relaxing (SOR) iterative method and OpenMP parallel framework to rapidly solve linear systems.

At the platform level, two computation-intensive modules of IDW, data extraction and view synthesis, are ported to GPU. For data extraction, feature matching in different image patches is independent of each other. Furthermore, view synthesis consists of block deformations performed independently on many triangles. These two modules exhibit a high degree of parallelism. We use OpenCL to implement parallelized computations for these two modules on a GPU device.

Manuscript received July 26, 2016; revised December 3, 2016; accepted January 9, 2017. Date of publication January 16, 2017; date of current version May 13, 2017. This work was supported in part by the National Natural Science Foundation of China under Grant 61672063 and Grant 61370115, in part by the China 863 project of 2015AA015905, in part by the Shenzhen Peacock Plan, Shenzhen Research Projects of JCYJ20150331100658943 and JCYJ20160506172227337, and in part by the Guangdong Province Projects of 2014B010117007 and 2014B090910001. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Leonel Sousa. *(Ronggang Wang and Jiajia Luo contributed equally to this work.)*

The authors are with the School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School, Shenzhen 518055, China (e-mail: rgwang@pkusz.edu.cn; abbyluo@sz.pku.edu.cn; xbjiang@pku.edu.cn; wangzhenyu@pkusz.edu.cn; wangwm@pkusz.edu.cn; gli@pkusz.edu.cn; wgao@pku.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2017.2654120

Experimental results show that with proposed acceleration scheme, the original IDW algorithm is accelerated by nearly 5x at the algorithm level and by more than 110x at the platform level plus algorithm level. The 8-view 4 K video (each view equals to 720P resolution) for MADs can be converted from S3D in real-time using a hybrid platform of CPU + NVIDIA GTX980 GPU.

The remainder of this paper is organized as follows. Section II reviews related works. Section III presents the proposed algorithm-level acceleration strategy, including a subsection to introduce the IDW algorithm. Section IV describes the proposed platform-level acceleration scheme on GPU. Section V presents the experimental results, and Section VI concludes the paper.

II. Related Work

The IDW algorithm used in 3D videos dates back to 2010, when Lang *et al.* [7] proposed non-linear disparity mapping for S3D content processing. With deep insights into stereo vision perception, they formalized a set of disparity mapping operators to nonlinearly retarget the depth of a stereoscopic scene. In 2011, Farre *et al.* [8] presented an IDW-based system to automatically create content for MADs, which yielded high-quality synthesized views. These works established the foundation for IDW as an effective alternative to DIBR for virtual view synthesis.

Later, Stefanoski *et al.* [9] proposed an IDW-based view synthesis method and devised a dedicated hardware architecture to achieve real-time performance. However, the hardware architecture for a specific MAD lacks flexibility.

In 2015, Yao *et al.* [10] proposed a real-time stereo to multi-view conversion system based on adaptive meshing. To reduce the computational complexity of IDW, they introduced adaptive meshing and relied on a block matching method to estimate sparse disparity. Some modules were parallelized on GPU using CUDA. In adaptive meshing, block merging was performed using block saliency similarity measurements. This merging process was too complex to be implemented on GPU completely. Although block merging could reduce the solution space of the energy equation, it would complicate the construction of the energy function. Given these limitations, only a few modules were ported to GPU, whereas a large amount of work was still undertaken by CPU. The authors claimed to have achieved real-time performance (i.e., 20.4 fps for 720P video, and 12.8 fps for 1080P video) on an NVIDIA Tesla K20 GPU housed in a Xeon workstation. However, sparse disparity estimation based on a block matching method has the disadvantage of serious matching errors, which would eventually cause artifacts in the synthesized views.

In this paper, we propose a new fast software implementation of IDW-based virtual view synthesis on a GPGPU platform. Here, we reserve the fixed meshing of the original IDW algorithm. The feature matching approach is also retained when estimating sparse disparity. Differently, we rely on FAST and BRIEF [16] to find feature correspondences very efficiently on GPU. And an iterative method is used to solve large linear equation systems in warp calculation. Task-level parallelism is also exploited to compute multiple warps simultaneously. Finally, image-domain warping (i.e., view synthesis module) is parallelized on a GPU device. These efforts allow real-time performance for a S3D to 8-view (each view is of 720P resolution) 4 K video conversion on a hybrid CPU + NVIDIA GTX980 GPU platform.

III. Algorithm-Level Acceleration

A. *Technical Background for the IDW Algorithm*

IDW employs sparse disparities to synthesize novel views. First, sparse disparities are automatically estimated. Then, the disparity data

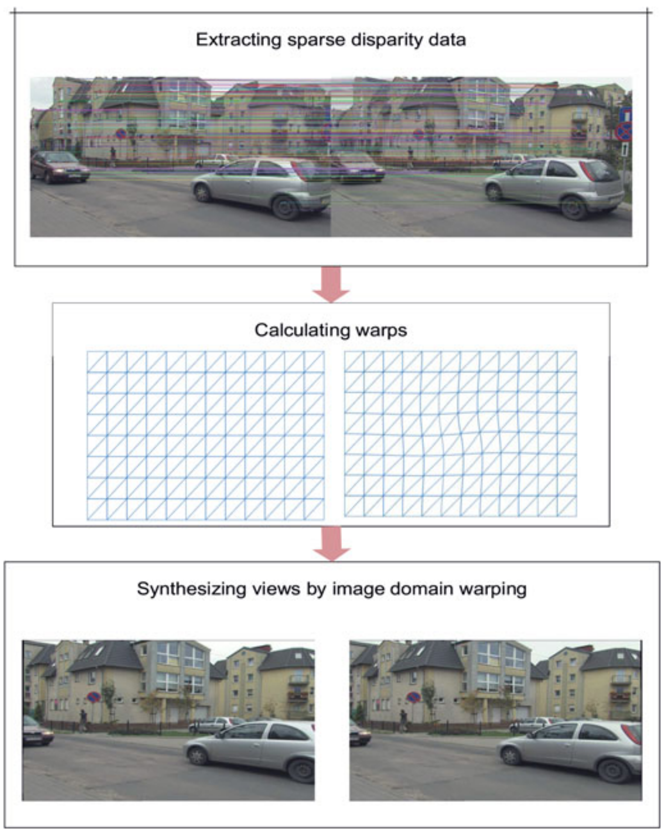
Extracting sparse disparity data
Calculating warps
Synthesizing views by image domain warping

Fig. 1. Overview of the IDW algorithm pipeline.

are used to compute image warps. Finally, with the guidance of warps, novel views are generated by image domain warping. An overview of the IDW processing pipeline is shown in Fig. 1.

1) Extracting Data: In this step, a sparse set of disparity features is extracted in the form of locl feature correspondences between the input stereo pair. Usually, SIFT as a robust feature extraction algorithm is adopted.

2) Calculating Warp: A warp is defined as a pixel mapping function ($\omega : [0,\ W] \times [0, H] \to \mathbb{R}^2$) and is used to distort the image and thus implement a desired change to the disparity. As Fig. 1 shows, the input view is discretized by a triangle mesh. To reduce the complexity of calculating the mapping relationships for all pixels, the warping function is only computed at the vertices of the triangle mesh, whereas the warping functions of non-vertex pixels are decided by the affine transformation of the triangles in which they are contained [17], [18].

The warp, ω, is solved by minimizing the quadratic energy function as

$$E(\omega) \coloneqq \lambda_d E_d(\omega) + \lambda_s E_s(\omega) + \lambda_t E_t(\omega) \quad (1)$$

where E_d is a disparity constraint term, E_s is a spatial smoothness constraint term, and E_t is a temporal smoothness constraint term. λ_d, λ_s, and λ_t are the corresponding weights.

The process used to construct the energy function for ω_L, which warps the left view (i.e., I_L) to the center position of the input stereo pair (i.e., I_L and I_R), is detailed as follows. The construction of ω_R is performed similarly.

a) Disparity term: First, the triangle S, containing point $\mathbf{p}_L$, is located for each feature correspondence $(\mathbf{p}_L, \mathbf{p}_R)$ obtained in the previous step. The relationship between S and $\mathbf{p}_L$ is formulated as

$$\mathbf{p}_L = \alpha \mathbf{v}_1 + \beta \mathbf{v}_2 + \gamma \mathbf{v}_3 \quad (2)$$

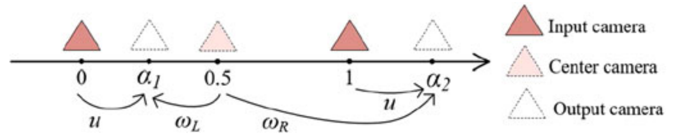

Fig. 2. Warp interpolation and extrapolation. Note that the axis is normalized by the baseline distance between the two input cameras. The warp at α_1 is computed by interpolation on u and ω_L, and the warp at α_2 is computed by extrapolation on u and ω_R.

where $[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ are the vertices of S and (α, β, γ) are the barycentric coordinates of $\mathbf{p}_L$ with regard to S.

The disparity term enforces a weak constraint on the distance between the warped position of $\mathbf{p}_L$ and its re-projected position $\mathbf{p}_M$ as

$$E_1(\mathbf{p}_L) = \|\alpha\omega_L(\mathbf{v}_1) + \beta\omega_L(\mathbf{v}_2) + \gamma\omega_L(\mathbf{v}_3) - \mathbf{p}_M\|^2 \tag{3}$$

where $\mathbf{p}_M = (\mathbf{p}_L + \mathbf{p}_R)/2$. By traversing all feature points, the disparity term is obtained as

$$E_d(\omega_L) = \sum_{\mathbf{p}_L \epsilon \mathbb{F}} E_1(\mathbf{p}_L) \tag{4}$$

where $\mathbb{F}$ is the set of feature correspondences.

b) Spatial smoothness term: Let (x, y) denote the index of a vertex in triangle mesh and $\mathbf{p}(x, y)$ denote its position in the image coordinate. The distortions of the vertical and horizontal edge of an upper triangle are measured by two functions as

$$\begin{aligned}\text{hor\_dist}(x,y) = \ &\|\omega_L(\mathbf{p}(x+1,y)) - \omega_L(\mathbf{p}(x,y)) \\ &- (\mathbf{p}(x+1,y) - \mathbf{p}(x,y))\|^2\end{aligned} \tag{5}$$

and

$$\begin{aligned}\text{ver\_dist}(x,y) = \ &\|\omega_L(\mathbf{p}(x,y+1)) - \omega_L(\mathbf{p}(x,y)) \\ &- (\mathbf{p}(x,y+1) - \mathbf{p}(x,y))\|^2 .\end{aligned} \tag{6}$$

The distortions of the vertical and horizontal edge of a lower triangle are measured similarly.

The spatial smoothness term $E_s(\omega_L)$ is obtained by summing the vertical and horizontal edge distortions of all upper and all lower right-angled triangles. It penalizes the geometrical distortion of triangles.

c) Temporal smoothness term: Temporal constraints are applied to ensure temporal stabilization of texture. Let ω_L^j be the warp of the j-th frame, and then the temporal smoothness term takes the form

$$E_t(\omega_L^j) = \sum_{\mathbf{p} \in \boldsymbol{M}} \left\| \omega_L^j(\mathbf{p}) - \omega_L^{j-1}(\mathbf{p})^2 \right\| \tag{7}$$

where $\mathbf{p}$ is the vertex in the triangle mesh M.

The energy function $E(\omega)$ is formulated as a quadratic expression of the coordinates of warped vertices in the triangle mesh. The resulting warp is determined by setting the partial derivative of $E(\omega)$ to 0, which is equivalent to solving a linear system (i.e., $Ax = B$).

3) Interpolating and Extrapolating Warps: In this step, all warps needed to synthesize as many output views as required for a particular MAD are computed. Interpolation and extrapolation on two calculated warps (i.e., ω_L and ω_R) could greatly reduce the complexity of calculating all the necessary warps. The process for interpolating and extrapolating warps is shown in Fig. 2.

u is a uniform warp (i.e., warp at the input camera position, which maps the input view to itself). ω_L and ω_R are two calculated warps at the center position of the two input cameras. The warps at output camera positions are computed by

$$\omega = \begin{cases} 2\alpha(\omega_L - u) + u, & \alpha \le 0.5 \\ 2(1-\alpha)(\omega_R - u) + u, & \alpha > 0.5 \end{cases}. \tag{8}$$

4) Synthesizing View: In this final step, virtual views are synthesized by warping the input stereo pair to targeted viewing positions under the guidance of the warp. The warped positions of vertices in the triangle mesh are solved by minimizing (1) in the previous step. The warped positions of non-vertex pixels inside a triangle are decided by the affine transformation of that triangle, which must be solved in this step at first. Using the warped positions of all pixels, a novel view could be rendered by sampling the source view.

TABLE I
RUN-TIMES REQUIRED FOR THE SYNTHESIS OF "ONE NEW VIEW" FROM A STEREOSCOPIC IMAGE PAIR [9]

Module Name	Seconds
Data extraction	2.5
Warp calculation	1.8
Warp interpolation/extrapolation	0.0000008
View synthesis	0.5

B. Algorithm-Level Acceleration for IDW

The run-time of each module in the original IDW is presented in Table I [9].

As Table I shows, data extraction, warp calculation, and view synthesis are the top three time-consuming modules. We propose the following algorithm-level acceleration strategies for data extraction and warp calculation.

Data extraction involves keypoint detecting and matching between the input stereo pair. IDW adopts the traditional SIFT algorithm, which provides robust feature matching. However, SIFT is very computationally complex. Therefore, we try to seek alternative feature extraction algorithms with lower computational complexities. SURF is a computationally efficient alternative to SIFT, but it remains unsuitable for real-time applications [14]. FAST is a high-speed feature detector [15]. Among SIFT, SURF, and FAST, the former two feature algorithms are invariant to scale and rotation, but the last one is not. Another difference between FAST and SIFT is that a large number of key points can be extracted with SIFT [13], whereas only a limited number of corner points can be detected by FAST. Given that for S3D to multiple-view conversion scenarios, the S3D input has been rectified to the same resolution and angle, the scale- and rotation-invariant characteristics of SIFT and SURF are redundant, but incur a heavy computation burden. Furthermore, the corner points of an image are usually clustered at edges and textured areas, which are vital to human vision perception. Thus, corner-points are sufficient to guarantee invariance in subjective quality. Therefore, in this paper, we adopt FAST to detect feature points, BRIEF to compute descriptors, and hamming distances to match features. The experimental results in Section V show that although some PSNR losses are caused by replacing SIFT with FAST, almost no observable degradation in subjective quality occurs. In addition to the algorithm level, data extraction can be further accelerated at the platform level. Since keypoint detecting is independent of pixels and feature matching is independent of key points, these processes are suitable for parallelization with GPU. The details of our proposed parallel implementation of data extraction are presented in Section IV.

For warp calculation, with regard to the energy function construction in it, the spatial smoothness term only need to be computed once and

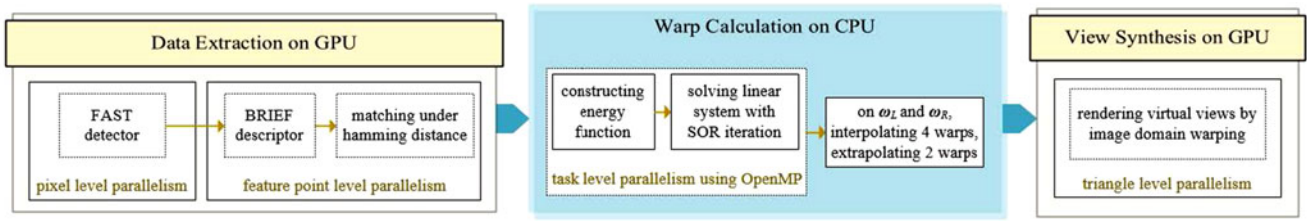
Data Extraction on GPU
FAST detector
BRIEF descriptor
matching under hamming distance
pixel level parallelism
feature point level parallelism
Warp Calculation on CPU
constructing energy function
solving linear system with SOR iteration
on ω_L and ω_R, interpolating 4 warps, extrapolating 2 warps
task level parallelism using OpenMP
View Synthesis on GPU
rendering virtual views by image domain warping
triangle level parallelism

Fig. 3. Complete parallel procedure of the proposed parallelization approach. Note that our S3D to 8-view conversion scenario requires 8 viewing positions: −0.2, 0.0, 0.2, 0.4, 0.6, 0.8, 1.0, and 1.2. The warps of the output views in positions −0.2 and 1.2 are produced through extrapolation on ω_L and ω_R, whereas those in position 0.2, 0.4, 0.6, and 0.8 are generated through interpolation. The definitions of ω_L and ω_R are shown in Section III-A.

can be reused for all frames; the temporal term can be acquired from the previous frame; and the disparity term can be easily calculated if the number of detected features is reasonably controlled. Minimizing the energy function consumes the largest amount of time because it involves solving a high-order linear system (i.e., $Ax = B$). The coefficient matrix (i.e., A) is large, sparse, and banded; accordingly, it is very time-consuming to solve it when using direct methods such as LU composition. The proposed algorithm aims to seek other efficient solvers for linear systems. Iterative methods, which approximate real solutions as closely as possible, are good replacements for directive methods. Jacobi iteration, Gauss-Seidel iteration, and SOR iteration are three representative iterative methods. Among them, the SOR method has the fastest convergence rate. Here, we adopt the SOR iterative method to solve the linear equation of a warp. More warps can be computed simultaneously by exploiting the task-level parallelism. Specific implementation details are presented in Section IV.

The view synthesis module requires inverse mapping to render a novel view. For each pixel in the novel view (i.e., warped position), its corresponding position in the source view is first determined, and its pixel value is then determined by sampling the source view via bilinear interpolation. Obviously, it is difficult to accelerate the view synthesis module at the algorithm level. Fortunately, as each pixel is mapped independently, this module exhibits great potential for acceleration on GPU. More details of the implementation related to data-level parallelism are provided in Section IV.

IV. Platform-Level Acceleration on GPGPU

As analyzed in Section III, the proposed method aims to accelerate the three most time-consuming modules of IDW algorithm: data extraction, warp calculation, and view synthesis. Based on the proposed algorithm-level acceleration strategies, in this section, we further propose a platform-level acceleration scheme on a GPGPU platform. The whole parallel procedure for our proposed parallelization approach is shown in Fig. 3. For data extraction, we use a simplified feature extraction algorithm ported to GPU with an effective parallel strategy. For warp calculation, a rapid linear system solver is implemented efficiently on CPU, and task-level parallelism is exploited when calculating ω_L and ω_R. Finally, the view synthesis module is ported to GPU by exploiting data-level parallelism.

A. Data Extraction on GPU

Here, we use FAST to detect feature points and BRIEF to compute descriptors. Our implementation adopts FAST-12 [15] and BRIEF-32 [16]. For FAST, the pixel intensity comparison threshold value is set to 30. For BRIEF, the sampling locations stay fixed relative to the detected point, and the smoothing kernel size is 9. Feature matching is realized by calculating the hamming distance between two binary-value descriptors.

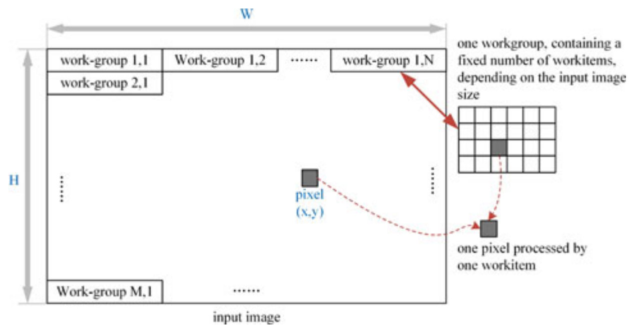
W
H
work-group 1,1
Work-group 1,2
......
work-group 1,N
work-group 2,1
Work-group M,1
pixel (x,y)
input image
one workgroup, containing a fixed number of workitems, depending on the input image size
one pixel processed by one workitem

Fig. 4. Thread allocation strategy for the FAST detector and BRIEF descriptor.

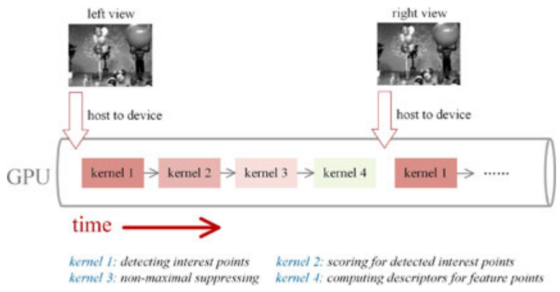
left view
right view
host to device
host to device
GPU
kernel 1
kernel 2
kernel 3
kernel 4
kernel 1
......
time
kernel 1: detecting interest points
kernel 2: scoring for detected interest points
kernel 3: non-maximal suppressing
kernel 4: computing descriptors for feature points

Fig. 5. Data transfers and kernel executions of the FAST detector and BRIEF descriptor for the left and right views.

The whole data extraction module is ported to a GPU device using OpenCL. Assuming the resolution of input views is $W \times H$, the thread allocation of FAST and BRIEF is shown in Fig. 4.

The whole data extraction module is ported to a GPU device using OpenCL. Assuming the resolution of input views is $W \times H$, the thread allocation of FAST and BRIEF is shown in Fig. 4 as follows.

As Fig. 4 shows, we allocate $M \times N$ (two-dimensional) work-groups comprising a fixed number of work-items, which yield a sum of $W \times H$ (two-dimensional) work-items. Each work-group corresponds to one image patch, and each work-item corresponds to a pixel. Using work-groups, the access to global memory is replaced by the access to more efficient local memory.

For FAST, three sequence-dependent kernels are designed corresponding to the three substeps (i.e., detecting corners, scoring for detected corners, and suppressing non-maximal corners). Only one kernel is designed for BRIEF. The FAST detector and BRIEF descriptor need to work on both the left and right view, to find feature correspondences between the input stereo pair. The sequence of the associated data transfers and kernel computations is shown in Fig. 5.

As Fig. 5 shows, one command queue is created, given one GPU device. After transferring the gray image data from the host (i.e., CPU) to the device (i.e., GPU), the four kernels are launched sequentially, with the output of the previous kernel becoming the input of current

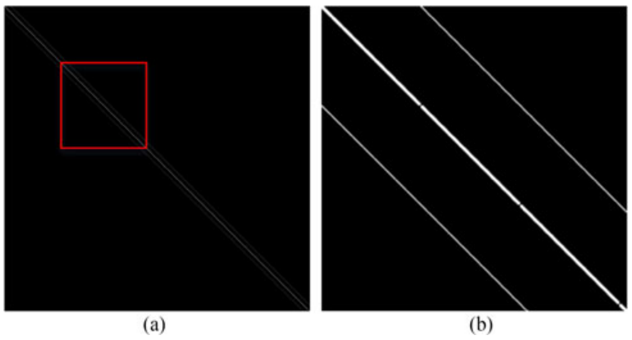

Fig. 6. (a) The coefficient matrix obtained when solving the linear system of the energy function's partial derivatives. (b) Close-up of the highlighted areas in (a). Black and white pixels specify zero and non-zero values in the coefficient matrix, respectively.

kernel. After processing the left view, the right view is processed in the same manner. Furthermore, the calculations for the left and right views can be overlapped when multiple GPU devices are available.

To address feature matching, hamming distances are computed between the feature points of the left and right views. Note that as the rectified image pair has zero vertical disparity, the feature correspondences should be located on the same horizontal line. Such epipolar constraint reduces feature matching from a 2-D to a 1-D search, which improves matching precision, as well as speed. Finally, the coordinates of matched correspondences are transferred from the device (GPU) to the host (CPU) as the input for the warp calculation module.

B. Warp Calculation on CPU

In this module, we compute two warps, ω_L and ω_R, which describe the nonlinear transformation of the input stereo pair, I_L and I_R, to their center viewing position. For rapid warp computation, we adopt the SOR iterative method to solve the linear system (i.e., $Ax = B$) formed during energy function minimization, as shown in (1).

The input view is discretized using $\mathrm{Grid}_x \times \mathrm{Grid}_y$ rectangles, and each rectangle is further divided into two triangles (shown in Fig. 1). The coefficient matrix (i.e., A) is a sparse, banded square matrix of order $(\mathrm{Grid}_x + 1) \times (\mathrm{Grid}_y + 1)$ which equals to the number of vertices in triangle mesh. The matrix A can be shown by Fig. 6.

In the SOR iteration process, the i-th component of the solution vector (i.e., x) in the k-th iteration is calculated as

$$x_i^{(k)} = (1-w)x_i^{(k-1)} + \frac{w}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right] \quad (9)$$

where w is the relaxation factor, a_{ij} is an entry in the coefficient matrix, and b_i is an entry in the constant matrix. We implement SOR iteration on a CPU rather than GPU because, as given by (9), the i-th component in the k-th iteration, $x_i^{(k)}$, relies on components $x_1^{(k)}$ to $x_{i-1}^{(k)}$ (both are in the k-th iteration). In other words, the calculation for any component in the k-th iteration should be postponed until the calculations for all previous components have been completed. Under such circumstances, parallel computation degenerates into serial computation, and the advantageous multiple-core architecture of GPU makes no sense.

To avoid moving massive unnecessary coefficient matrix entries (those entries equal to 0 are meaningless for SOR iteration) from memory to the processor, we designed a lightweight two-dimensional array containing only non-zero entries of the coefficient matrix. This array contains $(\mathrm{Grid}_x + 1) \times (\mathrm{Grid}_y + 1)$ rows and 5 columns, and the

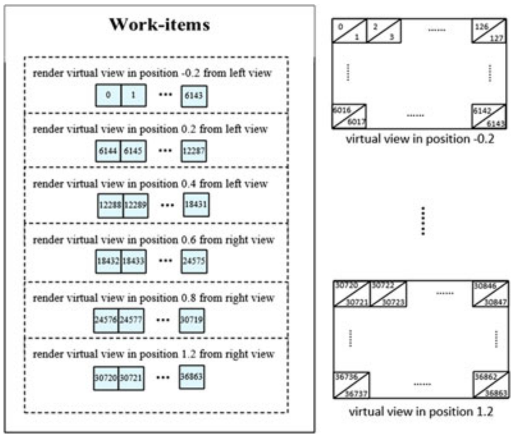
Work-items
render virtual view in position -0.2 from left view
0
1
6143
render virtual view in position 0.2 from left view
6144
6145
12287
render virtual view in position 0.4 from left view
12288
12289
18431
render virtual view in position 0.6 from right view
18432
18433
24575
render virtual view in position 0.8 from right view
24576
24577
30719
render virtual view in position 1.2 from right view
30720
30721
36863
0
1
2
3
126
127
6016
6017
6142
6143
virtual view in position -0.2
30720
30721
30722
30723
30846
30847
36736
36737
36862
36863
virtual view in position 1.2

Fig. 7. View synthesis thread allocation strategy. The virtual view positions are shown in Fig. 2.

entries in each row are given as

$$A_{\mathrm{sparse}}(i,0) = A(i, i - \mathrm{Grid}_x - 1) \quad (10.1)$$

$$A_{\mathrm{sparse}}(i,1) = A(i, i-1) \quad (10.2)$$

$$A_{\mathrm{sparse}}(i,2) = A(i, i) \quad (10.3)$$

$$A_{\mathrm{sparse}}(i,3) = A(i, i+1) \quad (10.4)$$

$$A_{\mathrm{sparse}}(i,4) = A(i, i + \mathrm{Grid}_x + 1). \quad (10.5)$$

Note that an entry should be set to 0 if its indexed entry $A(i,j)$ is out of bounds; for example, when $i = 0$, both $A(i,\ i - \mathrm{Grid}_x - 1)$ and $A(i, i-1)$ are out of bounds and $A_{\mathrm{sparse}}(i,0)$ and $A_{\mathrm{sparse}}(i,1)$ should be set to 0. Such a sparse representation of the coefficient matrix is cache-friendly and could help to speed up the iterating process.

In addition, we exploit task-level parallelism in this module. When minimizing the energy function to compute a warp, two linear systems that correspond to the x- and y-coordinates are obtained. So, a total of four linear systems must be solved to compute ω_L and ω_R. We employ a four-threaded OpenMP to solve these systems concurrently on a multi-core CPU.

Finally, through interpolation and extrapolation on two calculated warps (i.e., ω_L and ω_R), all the necessary warps in output camera positions are obtained according to (8).

C. View Synthesis on GPU

In this module, virtual view is rendered based on the input view (i.e., left or right view of the stereo pair) closer to the desired output position with the guidance of the corresponding warp.

Supposing that 8 viewing positions (−0.2, 0.0, 0.2, 0.4, 0.6, 0.8, 1.0, and 1.2) are required, 6 virtual views need to be synthesized (since output views in position 0.0 and 1.0 are the same with the input stereo pair). As mentioned earlier, the input view is discretized to $\mathrm{Grid}_x \times \mathrm{Grid}_y \times 2$ triangles. In our implementation, we set Grid_x to 64 and Grid_y to 48 so that each view consists of 6144 triangles.

The definition of warp indicates that although the warping function varies among triangles, all pixels in the same triangle are mapped with the same affine transformation. Here, we design a parallelized implementation between triangles on a GPU device. One kernel is designed to perform image domain warping and render each triangular area in the input views. The thread allocation strategy is shown in Fig. 7.

As Fig. 7 shows, a total of 36864 (one-dimensional) work-items are allocated, with 1024 (one-dimensional) work-items constituting a

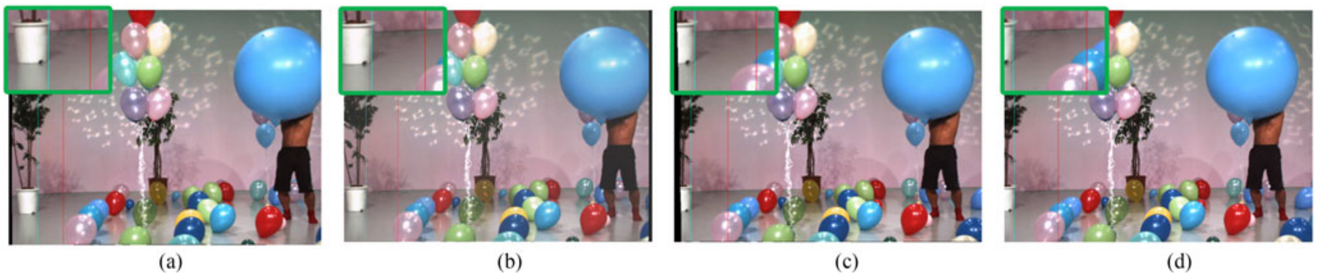

Fig. 8. Output views in positions −0.2, 0.2, 0.6, 1.0 (left to right), corresponding to (a), (b), (c), and (d), respectively. The two vertical lines represent auxiliary use. The red line is located exactly 180 pixels from the left boundary of the image, and the cyan line is located at the right edge of the vase. The distance between the two lines increases from left to right, as seen in close-ups, and indicates the change in parallax.

TABLE II
TEST VIDEO SEQUENCES FROM 3D-HEVC/HTM AND IMAGE PAIRS FROM MIDDLEBURY STEREO DATASETS

ID	NAME	RESOLUTION	INPUT VIEWS
S1	Balloons	1024 × 768	1,5
S2	Kendo	1024 × 768	1,5
S3	Newspaper	1024 × 768	2,6
I1	Aloe	641 × 555	1,5
I2	Baby1	620 × 555	1,5
I3	Bowling2	665 × 555	1,5
I4	Cloth3	625 × 555	1,5
I5	Rocks1	638 × 555	1,5
I6	Wood2	653 × 555	1,5

work-group. Using work-groups, access to global memory is replaced by access to more efficient local memory. In the kernel function, each work-item first computes the mapping relationship of the triangle, after which the corresponding positions are obtained in the source view; finally, the source view is sampled via bilinear interpolation to render a triangular area in the virtual view. This thread allocation strategy is simple but extensible for triangle mesh with a higher resolution. For instance, when a MAD requires more views (>8), the proposed thread allocation strategy could adapt well to this situation.

The warps (coordinates of mapped pixels in the novel view) and source views (source image data) must be transmitted to GPU memory prior to kernel execution. Warps are stored as buffer objects, and source views are stored as 2D image objects in OpenCL. The 2D image object is used to bind the image data to textures. By caching the textures on the GPU memory and accessing the data through an OpenCL sampler (the sampler should include a CLK_FILTER_LINEAR attribute to allow fast bilinear interpolation), efficient access to discrete image data is provided [19], [22]. View synthesis on GPU could be further parallelized when multiple GPU devices are available; for example, one GPU device could render views at positions <0.5, while another render views at positions >0.5.

V. Experimental Results

This section describes the extensive tests conducted on the MPEG 3D-HEVC test sequences and Middlebury stereo datasets[1] listed in Table II to validate the performance of our proposed method.

Using the proposed method, eight views are generated based on the input stereo pair. Parameters λ_d, λ_s, and λ_t from the energy function (1) are set to 1.0, 0.05, and 1.0, respectively. Fig. 8 shows an extracted frame of the synthesized views belonging to sequence S1 and demonstrates

TABLE III
OBJECTIVE QUALITY (PSNR) COMPARISON AMONG THE ORIGINAL IDW ALGORITHM, PROPOSED ALGORITHM WITH FAST SIMPLIFICATION, PROPOSED ALGORITHM WITH SOR SIMPLIFICATION, AND PROPOSED ALGORITHM WITH BOTH FAST AND SOR SIMPLIFICATION

ID	*Original Method*	*Simplification with FAST*	*Simplification with SOR*	*Simplification with both*
S1	28.96	27.94	28.23	27.78
S2	29.37	28.75	29.14	28.62
S3	26.03	24.81	25.79	24.75
I1	22.47	21.48	21.83	21.01
I2	29.61	28.56	28.99	28.09
I3	29.45	28.07	28.64	27.72
I4	28.89	27.93	28.24	27.51
I5	31.03	29.78	30.18	29.34
I6	33.72	32.37	33.18	32.08

that compared with DIBR, our method yields no holes and scarcely visible artifacts (e.g., boundary effects, halos).

A. Acceleration at Algorithm Level

In this subsection, we focus on reporting the quality losses due to the use of a simplified algorithm, including FAST during the data extraction module and SOR iteration during the warp calculation module.

On one hand, we measured the error introduced by FAST. Meanwhile, the directive methods adopted by the original IDW algorithm for solving the linear systems remained unchanged. We relied on the implementation in the LAPACK library.[2] On the other hand, we evaluated the error introduced by SOR. Likewise, we reserved the SIFT algorithm adopted by the original IDW algorithm for data extraction. Finally, we measured the error when both FAST and SOR were applied. The corresponding PSNRs are presented in Table III.

As Table III shows, when only FAST simplification was considered, the PSNR losses ranged from 0.6 db to 1.4 db, whereas when only SOR simplification was considered, the PSNR losses ranged from 0.2 db to 0.9 db. When both the FAST and SOR simplifications were eventually applied, the PSNR losses range from 0.7 db to 1.8 db. The objective quality losses caused by FAST are likely attributed to the fact that only corner points could be detected by FAST, the number of which is less than the feature points detected by SIFT. The objective quality losses resulting from SOR are attributable to the approximate solution obtained by iterative methods. Despite some losses in object quality, hardly any degradation in subjective quality was observed. More details about subjective quality evaluations are given in Table IV. Algorithm-

[1][Online]. Available: http://vision.middlebury.edu/stereo/data/

[2][Online]. Available: http://www.netlib.org/lapack/

TABLE IV
QUALITY COMPARISON BETWEEN ORIGINAL IDW ALGORITHM AND THE PROPOSED METHOD

ID	Evaluation Index	Original Method	Proposed Method	Trend
S1	PSNR	28.96	27.78	−1.18
	SSIM	**0.90**	**0.88**	**−0.02**
	MOS	3.50	3.38	−0.12
S2	PSNR	29.37	28.62	−0.75
	SSIM	**0.94**	**0.92**	**−0.02**
	MOS	3.50	3.13	−0.37
S3	PSNR	26.03	24.75	−1.28
	SSIM	**0.88**	**0.85**	**−0.03**
	MOS	2.63	2.50	−0.13
I1	PSNR	22.47	21.01	−1.46
	SSIM	**0.82**	**0.80**	**−0.02**
	MOS	2.25	2.12	−0.13
I2	PSNR	29.61	28.09	−1.52
	SSIM	**0.95**	**0.91**	**−0.04**
	MOS	3.54	3.19	−0.35
I3	PSNR	29.45	27.72	−1.73
	SSIM	**0.94**	**0.91**	**−0.03**
	MOS	3.52	3.28	−0.24
I4	PSNR	28.89	27.51	−1.38
	SSIM	**0.92**	**0.91**	**−0.01**
	MOS	3.49	3.38	−0.11
I5	PSNR	31.03	29.34	−1.69
	SSIM	**0.97**	**0.95**	**−0.02**
	MOS	3.66	3.56	−0.10
I6	PSNR	33.72	32.08	−1.64
	SSIM	**0.98**	**0.97**	**−0.01**
	MOS	3.85	3.82	−0.03

level simplifications can yield remarkable acceleration. Performance improvement details are demonstrated in the next subsection.

Additionally, we compared the synthesized views generated using the original IDW algorithm and the proposed method, respectively. Fig. 9 presents the synthesized views at position 0.5 for sequences S1, S2, and S3. Furthermore, we evaluated the corresponding objective quality (with the PSNR and perception-based SSIM indexes [23]) and subjective quality (with the MOS index), as listed in Table IV. As Table IV shows, despite losses of 0.8 to 1.8 db losses in PSNR, the degrees of decline regarding both SSIM and MOS are slight. In other words, although the objective quality of the views synthesized by the proposed method is somewhat lower than those synthesized by the original method, the losses in visual qualities were negligible. The simplification provided by FAST reduced the number of key points detected, compared with the original method, which would likely cause degradations in objective quality that manifest as PSNR losses. However, the slight declines in both the SSIM and MOS indexes reveal that the corner points detected by FAST are sufficient to preserve subjective quality in the salient areas of an image (i.e., where humans pay more attention).

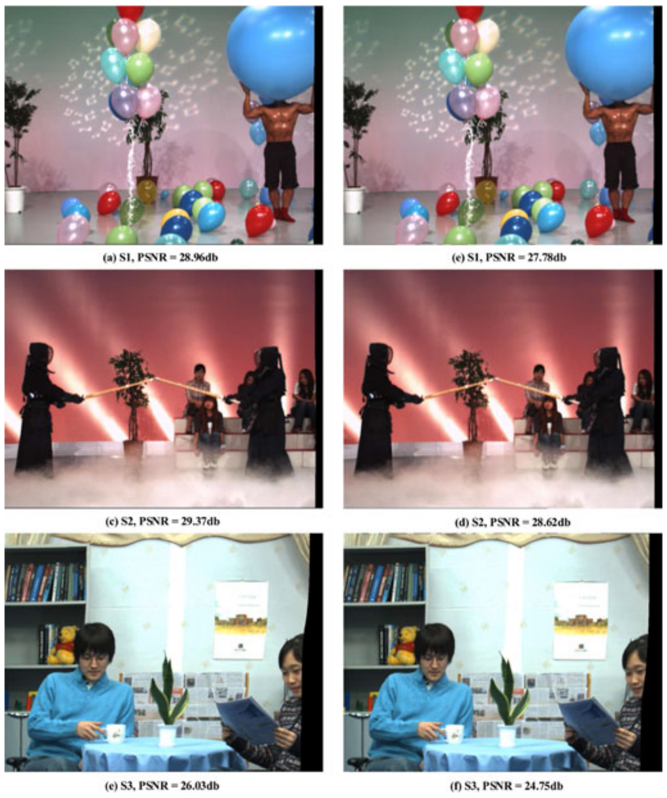

Fig. 9. Synthesized views. (a), (c), and (e) were obtained using the original IDW method, and (b), (d), and (f) were obtained using our proposed method.

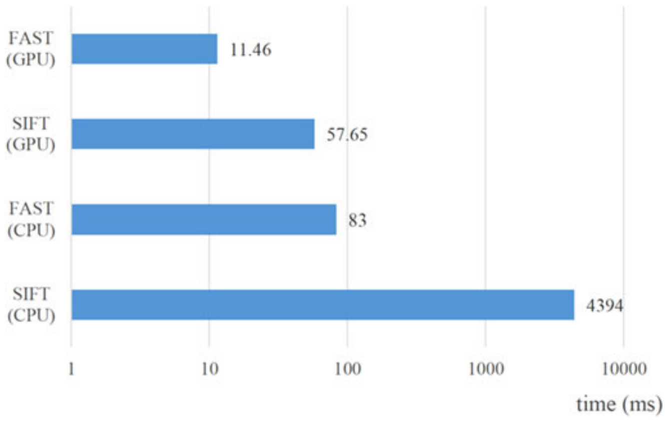

Fig. 10. Comparison of the execution time (in milliseconds) in the data extraction module, using SIFT (CPU), FAST and BRIEF (CPU), SIFT (GPU), and our implementation of FAST and BRIEF (GPU), respectively. The time axis is plotted on a logarithmic scale.

B. Acceleration at Platform-Level

This subsection describes the performance improvement in each module. This test is simulated in a C environment on a PC equipped with a 4.00-GHz Intel Core i7-4790K CPU and 16.0 GB of RAM, and an OpenCL environment with a NVIDIA GTX980 GPU. MSVC++ 11.0 is used as a C compiler. This environment also adopts a NVIDIA GPU Computing SDK (version 2.3b), OpenCL lib (version 1.1), and OpenCV lib (version 3.0).

First of all, the run-time of each module in the original IDW algorithm is provided in Table I [9], shown in Section III.

First, we adopted a simplified algorithm for the data extraction module, which was ported to the GPU using an effective parallel strategy. Specifically, we have an OpenCL implementation for FAST detector and BRIEF descriptor. We allocated 64×48 work-items for each work-group (i.e., the total number of work-groups was determined by the image resolution, $W \times H$). Fig. 10 depicts the run-time comparisons between SIFT and FAST (along with BRIEF). We used the CPU versions of SIFT and FAST implemented in OpenCV, and the GPU version of SIFT implemented in VisualSFM.[3] As Fig. 10 shows, on CPU, FAST was much faster than SIFT. Besides, our OpenCL implementation of FAST on GPU was approximately 5 times faster than SIFT on

[3][Online]. Available: http://cs.unc.edu/∼ccwu/siftgpu

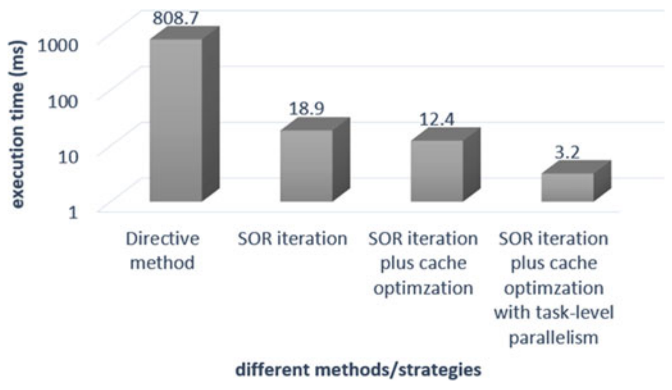

Fig. 11. Comparison of the execution times (in milliseconds) in the warp calculation module, using the directive method (implementation in the LAPACK library), SOR iterative method, SOR iterative method plus cache optimization, and SOR iterative method plus cache optimization with task-level parallel implementation, respectively. The time axis is plotted on a logarithmic scale.

TABLE V
COMPARISON OF THE EXECUTION TIMES (IN MILLISECONDS) REQUIRED TO SYNTHESIZE A SINGLE VIEW BETWEEN SEQUENTIAL CPU IMPLEMENTATION (IN C) AND OUR PARALLEL GPU IMPLEMENTATION (IN OPENCL)

		OpenCL	
Image resolution	C	data transfer	kernel execution
1024×768	94	0.51	1.03
1920×1088	254	2.02	2.58

the same platform. Overall, with the accelerations at both the algorithm (using FAST to substitute SIFT) and platform (resulting from the parallelized implementation on GPU) levels, the proposed algorithm led to much less execution time for the data extraction module.

Second, the SOR iterative method was applied with respect to warp calculation, and a cache-friendly sparse representation of the coefficient matrix was designed to accelerate the iterating process. Finally, task-level parallelism was exploited to compute several warps concurrently using OpenMP. Specifically, the maximum number of iterations, residual tolerance, and relaxation factor in SOR iteration were set to 2000, 0.1, and 1.9, respectively. Besides, the initial value of the SOR iteration of the first frame was set to 0, whereas those of the other frames were set to the solutions to the previous frames. The OpenMP environment variable OMP_NUM_THREADS was set to 4. Comparisons of the run-times needed to compute ω_L and ω_R by different strategies are given in Fig. 11. As shown, the acceleration strategy in the proposed algorithm could greatly relieve the computational burden of the warp calculation module. Besides, we also considered using the sparse solver routines provided by the Intel MKL library[4] to compute ω_L and ω_R. We found that more than 8 milliseconds were required to solve the four required sparse linear systems (as mentioned in Section IV-B), although <3 milliseconds were required to solve only one system.

Third, for the view synthesis module, we designed a parallelized implementation on a GPU device by exploiting data-level parallelism. Table V illustrates the time cost of synthesizing a single view of different sizes, using C (serial implementation on CPU) and OpenCL (concurrent implementation on GPU), respectively. As Table V shows, we achieved a speed increase of about 50x on GPU. We also found that the bottleneck constraining the execution speed lies in data transfer.

[4][Online]. Available: https://software.intel.com/en-us/intel-mkl

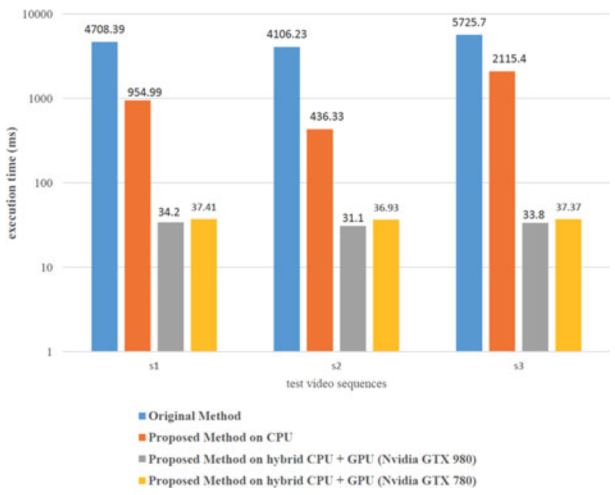

Fig. 12. Comparisons of the execution times of the entire view synthesis pipeline. The vertical axis reflects the time required to synthesize eight views per frame. The time axis is plotted on a logarithmic scale.

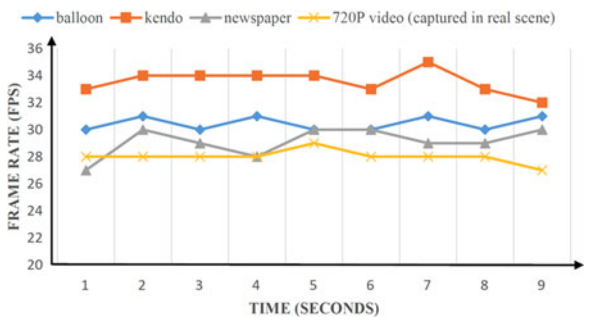

Fig. 13. Synthesized frames per second, with each frame containing 8 views.

Even so, the proposed parallelized implementation on GPU allowed the synthesis of eight (or more) views in real time.

Overall, we compared the total execution time of the proposed S3D to 8-view conversion approach with the original IDW algorithm [9]. To better demonstrate acceleration at both the algorithm and platform levels, we demonstrated the execution time of the proposed approach on CPU (acceleration in algorithm level) and a hybrid platform of CPU + NVIDIA GTX980 GPU (acceleration at both the algorithm and platform levels). Additionally, to verify scalability, we migrated our proposed approach to another hybrid platform (CPU + NVIDIA GTX780 GPU) and measured the execution time of the S3D to 8-view conversion process. The above-mentioned CPUs had the same configuration. More details are illustrated in Fig. 12.

As Fig. 12 shows, with algorithm-level acceleration, the proposed method can achieve an average increase in speed of approximately 5x, whereas acceleration at both the algorithm and platform levels yielded a speed increase of more than 110x. The proposed method could convert S3D to 8-view at a speed of 31 to 35 milliseconds per frame (about 30 fps) for video sequences S1, S2, and S3 (with a resolution of 1024×768) on the hybrid CPU + NVIDIA GTX980 GPU platform.

Finally, we deployed a system to convert S3D to 8-view in real time using the proposed approach on the hybrid CPU + NVIDIA GTX980 GPU platform. The input 720P stereo pair was captured by a binocular camera (Panasonic, Kadoma, Japan), and the synthesized views were simultaneously displayed by an 8-view 4 K MAD of TCL. We measured the frame rate data for the deployed system to convert the captured 720P video, as given in Fig. 13. For comparison, we also measured the frame rates for test sequence (S1, S2, and S3) conversion, as shown in Fig. 13.

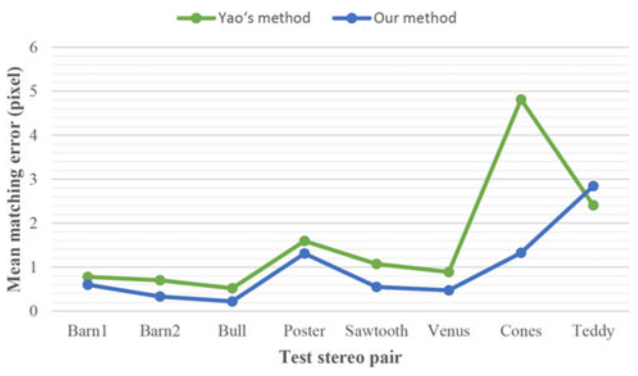

Fig. 14. Comparison of the mean matching errors (pixels) between the proposed method and Yao's method.

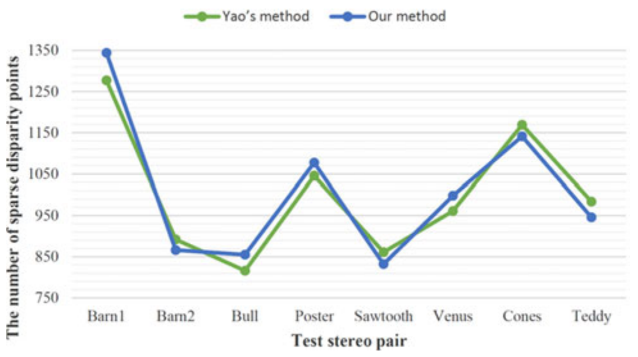

Fig. 15. Comparison of the number of disparate points between the proposed method and Yao's method.

As Fig. 13 shows, the frame rate fluctuated steadily around 30 fps. In other words, the proposed approach could convert 720P video from S3D to 8-view 4 K video in near real-time. In conclusion, the proposed approach has been proved feasible for real-time applications. Besides the performance advantages, the proposed approach exhibits good scalability and portability since it works well when migrated to other GPGPU platforms.

Last but not least, we also compared the proposed method with related works. In contrast to hardware solutions dedicated for a specific MAD [9], [24], the proposed method could generate different numbers of virtual views required by different MADs and therefore is more generalizable. Compared with the software acceleration method proposed by Yao [10], our method could generate the same number of disparity points (i.e., feature points obtained in the data extraction module) with reduced matching errors. This experiment was conducted on the Middlebury stereo database with ground truth, and the results are shown in Figs. 14 and 15. Obviously, mismatched feature points would cause artifacts in the synthesized virtual views.

VI. Conclusion

In this paper, we propose a new rapid software implementation for IDW-based view synthesis on a GPGPU platform. Our work makes two main contributions. First, at the algorithm level, we use FAST and BRIEF to extract sparse disparity data and adopt a successive over-relaxation (SOR) iteration to calculate warps. Second, at the platform level, we port the data extraction and view synthesis modules to a GPU device using OpenCL with an efficient parallelism strategy, and also use an OpenMP parallel framework to calculate several warps concurrently on CPU. Our efforts yielded a speed increase of more than 110x. Real-time performance for S3D to 8-view 4 K (each view with 720P resolution) video conversion was achieved on the hybrid CPU + NVIDIA GTX980 GPU platform.

Our approach is limited by the unutilized parallel opportunities between CPU and GPU. In our future work, we will design pipeline processing for tasks conducted on GPU (first and fourth modules) and on CPU (second and third modules) in the proposed method to achieve a better performance.

References

[1] A. Smolic *et al.*, "Intermediate view interpolation based on multiview video plus depth for advanced 3D video systems," in *Proc. IEEE Int. Conf. Image Process.*, Oct. 2008, pp. 2448–2451.
[2] A. Criminisi, P. Perez, and K. Toyama, "Region filling and object removal by exemplar-based image inpainting," *IEEE Trans. Image Process.*, vol. 13, no. 9, pp. 1200–1212, Sep. 2004.
[3] I. Ahn and C. Kim, "Depth-based disocclusion filling for virtual view synthesis," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jul. 2012, pp. 109–114.
[4] P. C. Kuo *et al.*, "High efficiency depth image-based rendering with simplified inpainting-based hole filling," *Multidimensional Syst. Signal Process.*, vol. 27, no. 3, pp. 1–23, 2016.
[5] A. Smolic *et al.*, "Three-dimensional video postproduction and processing," *Proc. IEEE*, vol. 99, no. 4, pp. 607–625, Apr. 2011.
[6] A. Gelman, P. L. Dragotti, and V. Velisavljevic, "Multiview image coding using depth layers and an optimized bit allocation," *IEEE Trans. Image Process.*, vol. 21, no. 9, pp. 4092–4105, Sep. 2012.
[7] M. Lang *et al.*, "Nonlinear disparity mapping for stereoscopic 3D," *ACM Trans. Graph.*, vol. 29, no. 4, 2010, Art. no. 75.
[8] M. Farre *et al.*, "Automatic content creation for multiview autostereoscopic displays using image domain warping," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jul. 2011, pp. 1–6.
[9] N. Stefanoski *et al.*, "Automatic view synthesis by image-domain-warping," *IEEE Trans. Image Process.*, vol. 22, no. 9, pp. 3329–3341, Sep. 2013.
[10] S. J. Yao *et al.*, "Real-time stereo to multi-view conversion system based on adaptive meshing," *J. Real-Time Image Process.*, pp. 1–19, 2015.
[11] C. Fehn, "Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV," in *Proc. SPIE Stereoscopic Displays Virtual Reality Syst. XI*, 2004, pp. 93–104.
[12] J. Fang, A. L. Varbanescu, and H. Sips, "A comprehensive performance comparison of CUDA and OpenCL," in *Proc. IEEE Int. Conf. Parallel Process.*, Sep. 2011, pp. 216–225.
[13] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.
[14] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. IEEE Int. Conf. Comput. Vis.*, Nov. 2011, pp. 2564–2571.
[15] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proc. Eur. Conf. Comput. Vis.*, 2006, pp. 430–443.
[16] M. Calonder *et al.*, "BRIEF: Computing a local binary descriptor very fast," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, pp. 1281–1298, Jul. 2012.
[17] G. Chaurasia, O. Sorkine, and G. Drettakis, "Silhouette - ware warping for image-based rendering," *Comput. Graph. Forum*, vol. 30, no. 4, pp. 1223–1232, 2011.
[18] F. Liu *et al.*, "Content-preserving warps for 3D video stabilization," *ACM Trans. Graph.*, vol. 28, no. 3, 2009, Art. no. 44.
[19] M. Scarpino, *OpenCL in Action: How to Accelerate Graphics and Computation*. NewYork, NY, USA: Manning, 2012.
[20] S. Momcilovic, A. Ilic, N. Roma, and L. Sousa, "Dynamic load balancing for real-time video encoding on heterogeneous CPU+GPU systems," *IEEE Trans. Multimedia*, vol. 16, no. 1, pp. 108–121, Jan. 2014.
[21] K. E. A. van de Sande, T. Gevers, and C. G. M. Snoek, "Empowering visual categorization with the GPU," *IEEE Trans. Multimedia*, vol. 13, no. 1, pp. 60–70, Feb. 2011.
[22] M. Schwalb, R. Ewerth, and B. Freisleben, "Fast motion estimation on graphics hardware for H.264 video encoding," *IEEE Trans. Multimedia*, vol. 11, no. 1, pp. 1–10, Jan. 2009.
[23] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.
[24] M. Schaffner *et al.*, "MADmax: A 1080p stereo-to-multiview rendering ASIC in 65 nm CMOS based on image domain warping," in *Proc. IEEE ESSCIRC*, Sep. 2013, pp. 61–64.