2012 INTERNATIONAL CONFERENCE ON
SIGNAL PROCESSING, COMMUNICATIONS AND COMPUTING
AUGUST 12 - 15, 2012, HONG KONG
www.icspcc2012.org
IEEE

2012 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC 2012)

The Polytechnic University of Hong Kong, China

12-15 August 2012

IEEE Hong Kong Section 香港分會 40th Anniversary (1971-2011)

IEEE XI'AN 西安

IEEE Catalog Number: CFP1251-ART

ISBN: 978-1-4673-2193-8

FOR SEMI-AUTOMATIC 2D TO 3D IMAGE CONVERSION

Author : *Xuyuan Xu; Lai-Man Po; Ka-Ho Ng; Ka-Man Wong; Chi-Wang Ting - City University of Hong Kong, China*
Kwok-Wai Cheung - Chu Hai College of Higher Education, China

Abstract : Depth map estimation from a single image is the key problem for the 2D to 3D image conversion. Many 2D to 3D converting processes, either automatic or semi-automatic, are proposed before. Quality of the depth map from automatic methods is low and there exists wrong depth values due to errors estimation in depth cue extraction. The semi-automatic approaches can generate a better quality of depth map based on the user-defined labels, which indicate a rough estimation of depth values in the scene, to generate the rest of depth value and reconstruct the stereoscopic image. However, they require complexity system and are very computational intensive. A simplified approach is to combine the depth maps from Graph Cuts and Random Walks to persevering the sharp boundary and fine detail inside the objects. The drawback is the time consuming of the energy minimization in the Graph Cuts. In this paper, a fast Watershed segmentation based on the priority queue, which indicates the neighbor distance relationship, is used to replace the Graph Cuts to generate the hard constraints depth map. It is appended to the neighbor cost in the Random Walks to generate the final depth map with hard constraints in the objects boundaries regions and fine detail inside objects. The Watershed and Random Walks are low computational intensive and can achieve approximate real time estimation which results in a fast stereoscopic conversion process. Experimental results demonstrate that it can produce good quality stereoscopic image in very short time.

Session Code : **SP2L.4 - Paper ID 0091**

Paper Title : **PARALLEL VIEW SYNTHESIS PROGRAMMING FOR FREE VIEWPOINT TELEVISION**

Author : *Jung, Jae-Il; Ho, Yo-Sung - Gwangju Institute of Science And Technology, Korea*

Abstract : View synthesis is one of the important techniques for free-viewpoint 3D image services. Unfortunately, computational complexity of depth image-based view synthesis is high, since it includes numerous matrix calculations and complex filters. In this paper, we implement the depth image-based rendering algorithm on a graphics processing unit (GPU) using the compute unified device architecture (CUDA). We perform memory uploads to the global memory of the GPU, and compute matrix calculations of all pixels in parallel. We also simplify the filters of rendering to reduce complexity and analyze performance according to block and image sizes. Experimental results show that our implementation is faster than conventional methods while preserving visual quality.

Session Code : **SP2L.5 - Paper ID 0072**

Paper Title : **QUALITY-EFFICIENT DE-INTERLACING FOR H.264-CODED VIDEOS**

Author : *Wei-Jen Yang; Kuo-Liang Chung; Le-Chung Lin - National Taiwan University of Science and Technology, Taiwan;*
Yong-Huai Huang - Jinwen University of Science And Technology, Taiwan

Abstract : In this paper, we propose an efficient de-interlacing method for H.264-coded video sequences with different resolutions. In our proposed method, using the syntax elements (SEs) in the H.264 bitstreams, two new strategies are delivered to improve the deinterlacing quality. The first strategy is based on the intra mode to improve the quality of the regions with skewed edges. The second strategy is based on the inter mode to refine the quality of de-interlaced videos as well as alleviate the error propagation side effect. Experimental results on popular test video sequences with the resolutions of common international format (CIF), quarter CIF (QCIF), standard- definition (SD), and full highdefinition (HD)

PARALLEL VIEW SYNTHESIS PROGRAMMING FOR FREE VIEWPOINT TELEVISION

Jae-Il Jung and Yo-Sung Ho
School of Information and Communications
Gwangju Institute of Science and Technology (GIST)
Gwangju, Republic of Korea
{jijung, hoyo}@gist.ac.kr

***Abstract*—View synthesis is one of the important techniques for free-viewpoint 3D image services. Unfortunately, computational complexity of depth image-based view synthesis is high, since it includes numerous matrix calculations and complex filters. In this paper, we implement the depth image-based rendering algorithm on a graphics processing unit (GPU) using the compute unified device architecture (CUDA). We perform memory uploads to the global memory of the GPU, and compute matrix calculations of all pixels in parallel. We also simplify the filters of rendering to reduce complexity and analyze performance according to block and image sizes. Experimental results show that our implementation is faster than conventional methods while preserving visual quality.**

Keywords-Depth image-based rendering; GPU; parallel programming; CUDA

I. INTRODUCTION

A 3D image provides immersive and realistic feelings to viewers and leads the next generation multimedia services. With the speedy growths of image processing, sensing and displaying, a lot of 3D products are releasing in the market such as TVs, games, PCs, and mobile products. This tendency will be continued for a long time [1, 2].

People cannot directly perceive the world in 3D, since the 3D world projects onto the curved surface, the retina, at the back of the eye. We estimate depth information based on some clues, and such clues are called depth cues. They are categorized into two types: binocular and monocular depth cues [3-5]. The binocular depth cues are activated when viewing a scene with both eyes and include stereopsis and convergence.

Unlike the binocular depth cues, the monocular depth cues help us to perceive depth when viewing a scene with one eye. Such a cue includes linear perspective, aerial perspective, focus effects, and so on. Since we can perceive relative depth feelings using the monocular depth cues from 2D pictures, such cues are often called pictorial cues.

On the basis of those two cues, 3D broadcasting services and we can enjoy 3D contents at home as shown in Fig. 1. The main factor of 3D image services is viewing comfort, and various researches focus on it for various applications. Among them, the free viewpoint television providing free-view navigation has been actively developed.

Figure 1. 3D video serivce and special display devices

The best way to get the images at various view positions is capturing with a lot of cameras, but it is practically impossible. Therefore, the images are generally synthesized from a few captured images. For view synthesis, the color images and their corresponding depth information are required, and the technology synthesizing intermediate views is called the depth image-based rendering (DIBR) [6].

Because the depth information represents the distance between camera and objects in a scene, it is possible to synthesize images at required view positions. Therefore we can reduce the number of viewpoints to be sent for 3D broadcasting. For instance, instead of transmitting ten-view videos, we are able to reconstruct them with only two color and depth videos. Exploiting this concept, the moving picture expert group (MPEG) researched 3D video systems using DIBR [7, 8].

In DIBR, it is very important to get an accurate depth image, since the quality of depth image directly affects visual quality of synthesized images. Although, a lot of stereo matching algorithms have been researched by huge number of works, it is still a hard problem for texture-less or periodic regions in images [9]. Therefore some hardware approaches take center stage such as structured light patterns and depth cameras [10]. After getting the depth information, we synthesize the virtual view images using DIBR. DIBR is a time consuming process,

978-1-4673-2193-8/12/$31.00 ©2012 IEEE

since it is has a lot of matrix calculations and pixel-wise image processing.

In this paper, we accelerate the DIBR process by using graphics processor unit (GPU), which has evolved into an absolute computing workhorse. The GPUs offer incredible resources for both graphics and non-graphics processing, because they have multiple cores driven by very high memory bandwidth.

II. Parallel Programming on GPU

Recently, the programmable GPU has evolved into an absolute computing workhorse. Since GPUs have multiple cores with high memory bandwidth, they can be proper resources for various processes [11].

Since GPU has many transistors for data processing, it can intensively compute with parallel structures. Therefore, the GPU is especially suited to address problems which are expressed as parallel computations with high arithmetic intensity. Data-parallel processing sends data elements to parallel threads. For accelerating, many applications having process large data sets adopt such a data-parallel programming model.

It is very effective especially for 3D image processing; because it contains many pixels and vertices to be calculated. In addition, this structure shows powerful results for conventional image and media processing applications such as video encoding and decoding, image scaling, stereo vision, and pattern recognition [12, 13]. Even if some applications have no pixel or vertex structures, they can be accelerated by data-parallel processing.

A lot of programming languages and libraries, such as OpenMP, CUDA, and MPI, for data-parallel processing have been developed. Among them, CUDA developed by NVIDIA is a powerful library and has a simple structure. CUDA intensively decreases computational times via various threads in GPUs. We can easily access to the virtual instruction set and memory of GPUs via CUDA. In other words, we can freely use GPUs as CPUs for complex applications.

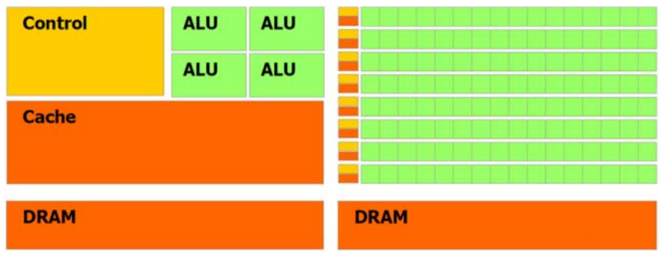


Figure 2. Structures of CPU and GPU

In general, GPUs have parallel structures that emphasize executing many concurrent threads slowly, rather than executing a single thread very quickly. It is the main difference between CPU and GPU. Figure 2 shows the different structures between CPU and GPU. While CPU has a limited number of memories and logic units for a fast single thread, GPU has a lot of memories and units which are suitable for parallel programming.

III. Parallel DIBR on GPU

In this paper, we accelerate the DIBR process by using GPU explained in Sec. 2. In this section, we explain our implementation with the general procedure of DIBR. Figure 3 shows the overall procedure of DIBR and parallel types. The red and blue boxes represent pixel- and column-wise structures.

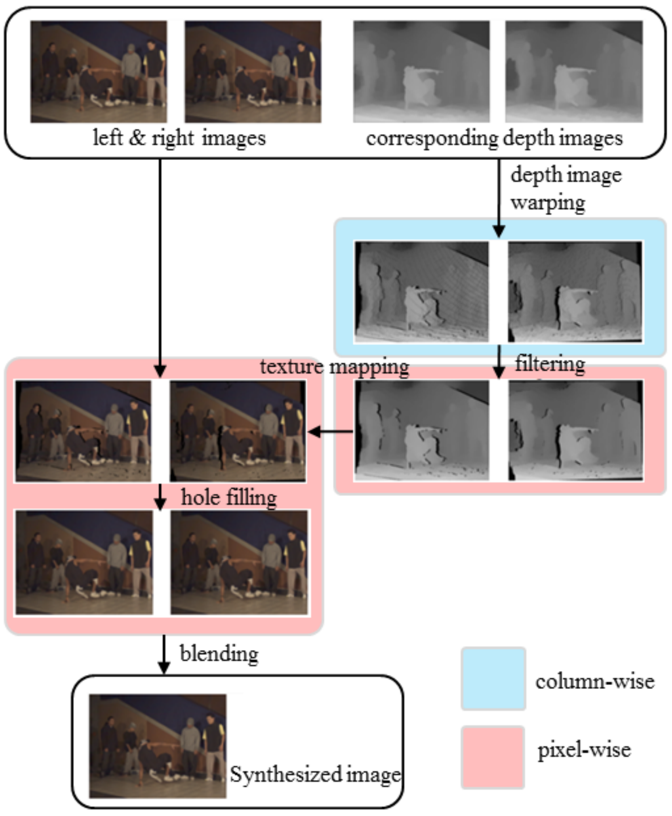


Figure 3. Procedure of DIBR with parallel types

A. *Depth image warping*

At first, we read input color and depth images of left and right views, and upload them to a global memory of GPU. We design our implementation so that the CPU and GPU do not exchange unnecessary data, because the speed of data exchange between Host and Device units is very slow.

Then, in order to fill the holes caused by round off error, we generate a depth image for the target viewpoint instead of direct color mapping. If all camera parameters are given, we can find pixel correspondences between two cameras [14]. When a point **M** in world coordinate is projected to a camera, a pixel **m** in the image can be found by (1). The representations of a single point $\mathbf{M} = [X\ Y\ Z\ 1]^T$ and a projected point $\mathbf{m} = [x\ y\ 1]^T$ are the homogeneous form of a position.

$$\mathbf{m} = \mathbf{A}[\mathbf{R} \mid \mathbf{t}]\mathbf{M} \qquad (1)$$

where **A** is the intrinsic camera parameter, and **R** and **t** are the extrinsic camera parameters. The detail elements of the intrinsic parameter are shown in (2), and *f*, contained in *a* and *b,* relates to the focal length.

$$\mathbf{A} = \begin{bmatrix} a & -a\cot\theta & u_0 \\ 0 & b/\sin\theta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (2)$$
$$where,\ a = kf, b = lf$$

In (2), u_0 and v_0 define the position of the principal point in the retinal coordinate system, *k* and *l* represent a pixel dimensions, and θ is the angle between the two image axes.

When we find the corresponding pixel between a reference and target images, we project a pixel $\mathbf{m_r}$ in the reference image to the world coordinate using Eq. (3).

$$\mathbf{M_r} = \mathbf{R_r}^{-1}\cdot\mathbf{A_r}^{-1}\cdot\mathbf{m_r}\cdot d(\mathbf{m_r})\text{-}\mathbf{R_r}^{-1}\cdot\mathbf{t_r} \qquad (3)$$

where the representations of $\mathbf{A_r}$, $\mathbf{R_r}$, and $\mathbf{t_r}$ stand for camera parameters of the reference view. d($\mathbf{m_r}$) is a depth value of the pixel of $\mathbf{m_r}$. After projection of $\mathbf{m_r}$, we project $\mathbf{M_r}$ onto the required view position using (4).

$$\mathbf{m_t} = \mathbf{A_t}[\mathbf{R_t}|\mathbf{t_t}]\mathbf{M_r} \qquad (4)$$

If we conduct this process serially on CPU, it takes a very long time. Especially 3D warping and texture mapping processes are respectively conducted two times for left and right images. Therefore we parallelize this process on GPU and calculate the four sets of homography matrices for every depth value to reduce the total amount of matrix calculations. Then, we upload the matrices to a global memory. To prevent pixel overlapping caused by depth values, we use a column-wise structure and conduct this process in the proper directions.

B. Depth filtering

Since the warped depth image has holes from round off errors and inaccurate boundaries, the textures of foreground objects can remain in background regions after texture calling. This phenomenon degrades visual quality. Therefore we fill the holes and dilate depth image with consideration the direction of desired view position as (5).

$$D_{Dilated} = \begin{cases} Depth & \dfrac{D_{cur} + \sum_{N} \begin{cases} D_{nb} & if(|D_{cur} - D_{nb}| < Th) \\ 0 & else \end{cases}}{N_{vnb} + 1} \end{cases} \qquad (5)$$

where D_{cur} and D_{nb} are the depth values of current and neighboring pixels, respectively. *N* is the set of neighboring pixels, and N_{vnd} means the number of valid neighbors whose absolute difference is smaller than threshold *Th*. In this paper, we set *Th* with 10.

C. Texture mapping and hole filling

After depth filtering, the next step is texture mapping for the target viewpoint image. In the same manner as depth warping, we can call the color information for every pixel at the required view position. Unlike depth warping, this process does not need a column-wise structure, because it shows the same result regardless of a processing order. Therefore we use a pixel-wise structure for parallelizing.

The hole regions in the warped depth image, which are newly exposed regions cause by viewpoint change, still exist in the mapped color image. The reference image has no information on the appeared areas, but we can find the corresponding information of the hole area in the other input image. Copying those information is our strategy to fill in the hole areas. After hole filling, we blend two synthesized view into one according to view distances. Such processes are perfectly independent of neighboring pixels, thus we parallelize them with pixel-wise structures. Finally, we call the blended image in the global memory of GPU to CPU.

IV. Experimental Results

In order to evaluate the performance of our method, we synthesized an intermediate view of MPEG sequences: café and bookarrival. For the experiment, we used the desktop computer which is equipped with an Intel Core 2 Quad CPU running at 2.4 GHz and NVIDIA Geforce GTX 580 GPU. The GPU has 512 CUDA cores and 1.5 GB standard memory. It works on 772 MHz core clock and on 1,544 shader clock.

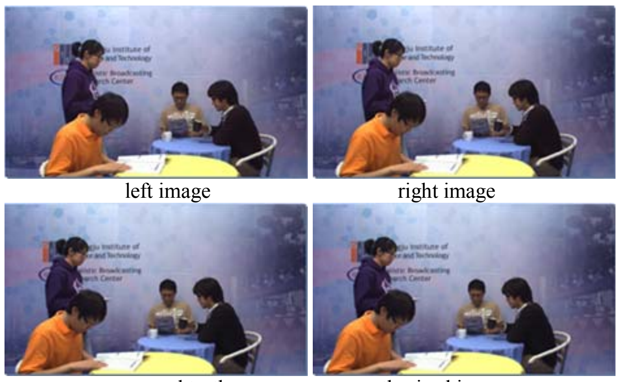
Before checking processing times, we found the optimal block size for DIBR. We had tests on five block sizes: 2, 4, 6, 8, and 10. On our system, the block size of eight shows the best performance, thus we set the block size with eight for further experiments.

Table I shows the comparison result of the processing times between CPU and GPU implementations, according to image sizes. For every image size, our implementation shows faster results than CPU implementation.

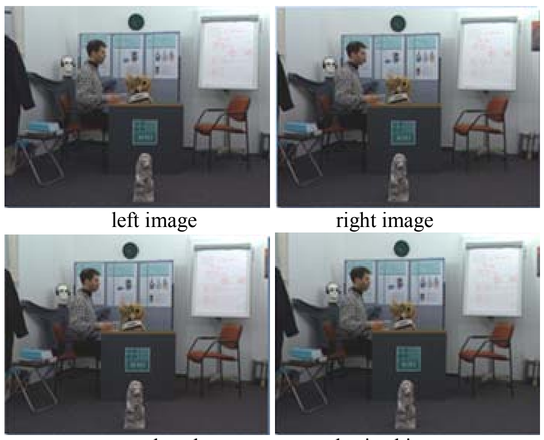
TABLE I. COMPARISON OF PROCESSING TIMES

Image Size (10 frames)	CPU (s)	GPU (s)
640 x 480	16.48	0.21
1024 x 768	22.44	0.33
1280 x 960	28.01	0.52
1920 x 1080	33.18	0.82

Our method is about 40 times faster than the conventional DIBR on CPU, and it can synthesize more than 10 frames for full-HD images. Figure 4 demonstrates the visual quality of synthesized images. Our method provides similar results with ground truth images, and their PSNR values are greater than 32dB. The results of objective quality measure depend on the depth quality.



(a) *Café* sequence (32.03dB)



(b) *Bookarrival* sequence (34.14dB)

Figure 4. visual quality of the synthesized images

V. CONCLUSION

In order to synthesize images at various view positions, we utilize the DIBR techniques, but it is a time-consuming process. In this paper, we accelerate the DIBR process by using GPU and modifying the DIBR part. To reduce processing time, we minimize unnecessary date exchanges between Host and Device units, and design the optimal parallel structure for the filters and matrix calculations. We analyzed the processing time according to grid and block sizes. Experimental results show that our implementation is about 40 times faster than the conventional DIBR on CPU.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2011-0030822).

REFERENCES

[1] A. Kubota, A. Smolic, M. Magnor, M. Tanimoto, T. Chen, and C. Zhang, "Multi-View Imaging and 3DTV (Special Issue Overview and Introduction)," *IEEE Signal Processing Magazine,* vol. 24, no. 6, pp. 10-21 2007.

[2] M. Tanimoto, "Overview of free viewpoint television," *Signal Processing: Image Communication,* vol. 21, no. 6, pp. 454-461, 2006.

[3] V. De Silva, A. Fernando, S. Worrall, H. Kodikara Arachchi, and A. Kondoz, "Sensitivity Analysis of the Human Visual System for Depth Cues in Stereoscopic 3-D Displays," *IEEE Transactions on Multimedia,* vol. 13, no. 3, pp. 498-506, 2011.

[4] S. Reichelt, R. Häussler, G. Fütterer, and N. Leister, "Depth cues in human visual perception and their realization in 3D displays," p. 76900B, 2010.

[5] A. Saxena, M. Sun, and A. Y. Ng, "Make3d: Learning 3d scene structure from a single still image," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 31, no. 5, pp. 824-840, 2009.

[6] C. Fehn, "Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV," p. 93, 2004.

[7] A. Smolic and D. McCutchen, "3DAV exploration of video-based rendering technology in MPEG," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 14, no. 3, pp. 348-356, 2004.

[8] A. Smolic, K. Mueller, P. Merkle, C. Fehn, P. Kauff, P. Eisert, and T. Wiegand, "3d video and free viewpoint video-technologies, applications and mpeg standards," in *IEEE International Conference on Multimedia and Expo* pp. 2161-2164, 2006.

[9] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision,* vol. 47, no. 1, pp. 7-42, 2002.

[10] E. K. Lee and Y. S. Ho, "Generation of high-quality depth maps using hybrid camera system for 3-D video," *Journal of Visual Communication and Image Representation,* vol. 22, no. 1, pp. 73-84, 2011.

[11] D. Kirk, "NVIDIA CUDA software and GPU parallel computing architecture," in *the 6th international symposium on Memory management,* pp. 103-104, 2007.

[12] J. Woetzel and R. Koch, "Real-time multi-stereo depth estimation on GPU with approximative discontinuity handling," in *the 1st European Conference on Visual Media Production*, pp. 245-254, 2004.

[13] A. Brunton, C. Shu, and G. Roth, "Belief propagation on the GPU for stereo vision," in *the 3rd Canadian Conference on Computer and Robot Vision*, pp. 76-76, 2006.

[14] R. Hartley, A. Zisserman, and I. ebrary, *Multiple view geometry in computer vision*. 2: Cambridge Univ Press, 2003.