Real-Time CPU-Based Virtual View Synthesis

Adrian Dziembowski, Jakub Stankowski
Chair of Multimedia Telecommunications and Microelectronics
Poznan University of Technology
Poznań, Poland
{adziembowski, jstankowski}@multimedia.edu.pl

***Abstract*—This paper presents the fast virtual view synthesis method for free navigation purposes. Presented method was implemented on CPU in order to allow free navigation for even cheap personal comput-ers with integrated graphic cards. The described synthesis technique is based on the hybrid view synthesis, combining advantages of forward-type and backward-type synthesis. It allows to synthesize high resolution sequences in the real time, allowing the viewer to freely navigate around the scene, e.g. volleyball playfield. Three implementations were developed and tested: single-threaded, vectorized and multi-threaded implementation.**

Keywords—virtual view synthesis; free-viewpoint television; free navigation

I. INTRODUCTION

The main purpose of the Free-Viewpoint Television (FTV) [1], [2] is to allow a viewer to navigate freely around a scene captured by multiple cameras [3], [4], [5], [6].

When the viewer chooses to observe the scene from a viewpoint in between the real views, it is necessary to synthesize the virtual view (Fig. 1). In order to allow virtual navigation, the virtual view synthesis should be performed in the real-time.

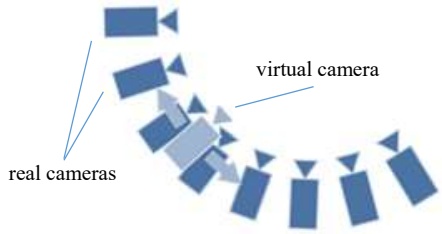

Fig. 1. View synthesis idea

The goal of the paper is to study open questions in the real-time view synthesis using low-cost implementations. The authors' considerations will be related to the experience with practical implementations in an experimental free-viewpoint television system.

II. ALGORITHM DESCRIPTION

In order to obtain the highest possible quality of the synthesized virtual views, the hybrid-type synthesis was proposed. This approach combines advantages of forward-type [7], [8] and backward-type [9], [10] synthesis. In contrary to the forward-type synthesis, which is faster and easier to implement in the real-time, the backward synthesis allows to filter the depth map of the virtual view before texture projection thus reducing erroneous influence of improperly computed, inconsistent depth maps.

The proposed synthesis algorithm can be divided into three main steps, presented in Fig. 2. In the first one, the depth map for the virtual view is calculated (orange block). In this step all the pixels from both the real views are projected to the virtual view but only their depth information is used. Color of each pixel is projected in the second step (blue block), where the actual virtual view is synthesized. In this step, combined virtual depth obtained from two real views is used for projecting color components from left or right real view. The last step of the virtual view synthesis is postprocessing of the synthesized view (purple block in Fig. 2). This step consists of texture filtering and inpainting, where first operation removes small artifacts and the second one fills remaining holes in the virtual view.

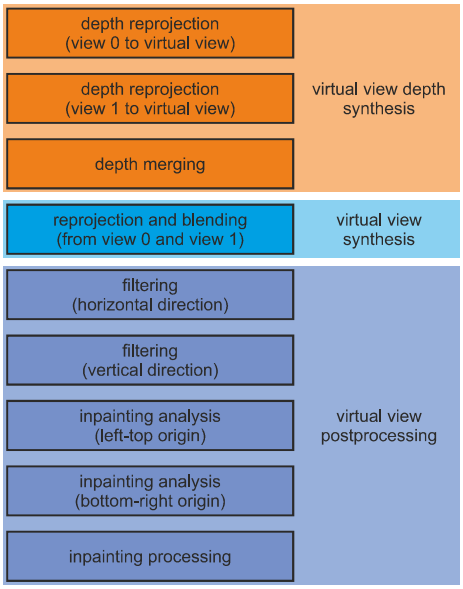

Fig. 2. View synthesis algorithm

As it was mentioned, the proposed hybrid approach allows to filter the virtual depth map. However, in the presented approach the virtual depth map and view are filtered simultaneously, so in the scheme presented in Fig. 2 we merged these two operations into common block called "filtering". The only filtering executed in the virtual depth synthesis block is the ghost edges removal [11], where pixels close to the edges in the depth maps are omitted during projection in order to avoid false edges in the virtual view.

A. Depth projection and merging

All the points of each real view can be represented as a vector $\mathbf{m} = [x\ y\ z\ 1]$, where x and y represent its position in the real view and z is its depth. In the step of depth projection, each pixel is projected from a real to the virtual view:

$$\mathbf{m_V} = \mathbf{m} \cdot \mathbf{P_V} \cdot \mathbf{P_R^{-1}}\,, \tag{1}$$

where $\mathbf{m_V} = [x_V\ y_V\ z_V\ 1]$ is the point position in the virtual view (x_V, y_V) and its depth (z_V). $\mathbf{P_R}$ and $\mathbf{P_V}$ are projection matrices for the real and the virtual view, respectively. Each projection matrix combines both the intrinsic and extrinsic parameters of the camera: $\mathbf{P} = \mathbf{K} \cdot \mathbf{E}$, where the intrinsic matrix $\mathbf{K}$ includes the camera focal length and the location of the principal point of the camera and extrinsic matrix $\mathbf{E}$ consists of the translation vector concatenated to the rotation matrix [12].

The result of depth projection are two depth maps – first for the points projected from the left real view and the second containing depth projected from the right real view. In the next step, both of these depth maps are merged in order to produce one consistent virtual depth map. In this step, for each point the lower (closer to the virtual camera) depth value is chosen.

B. View projection and blending

In the typical, backward-type synthesis, projection of color of all the points of the virtual view is based on previously estimated depth map. In this operation all the points are projected as in (1) (but in the opposite direction) in order to estimate the position of corresponding points in the real views [10].

Projection requires a number of operations, thus in the real-time algorithm the hybrid method was proposed. In this approach, projection is performed by the simple reading an array created during depth projection.

Any point of the virtual view could be visible in one or two real views or not visible in any real view. When the point was visible in one view, its color is just copied from the real view. If it was captured by both the real cameras, it is checked whether point's depth projected from both real views is similar. If this condition is true, the color of the point in the virtual view is obtained by averaging colors from both real views; if not – it is copied from the point with lower depth value (color of a point representing closer object). This approach allows to avoid blending background and foreground areas.

C. Synthesized view filtering

The purpose of the virtual view filtering is to eliminate two effects: view discontinuities entailed by finite resolution of the views and depth maps and small (but subjectively disturbing) artifacts caused by erroneous depth maps.

In order to remove discontinuities in the virtual view, for each point within a view it is checked whether the difference between colors of that point and its left neighbor is higher, than the difference between colors of that point's left and right neighbor. If it is higher, it is assumed that color of that point was wrong and it is replaced by the color of its right neighbor. In order to remove both horizontal and vertical discontinuities, analogous operation is done for vertical direction.

Errors in the depth maps may cause appearance of small non-existing objects, both in the background and in the foreground. While navigating within the scene, the foreground objects move faster, than other objects in the scene. Therefore, wrongly synthesized areas cause significant subjective quality deterioration.

In order to remove these artifacts, a simple method was proposed. For each point of the virtual view it is checked whether the point is closer to the virtual camera than its left and right neighbor. If this condition is true, color of this point is replaced by the color of its right neighbor. Of course, similar to discontinuities removal, analogous operation is performed for vertical direction.

D. Synthesized view inpainting

At this stage, synthesized and filtered virtual view still contains areas with no projected information (disoccluded areas). These areas have to be inpainted based on their neighborhood [13], [14], [15], [16], [17], [18].

In described algorithm, we proposed fast, depth-based interpolating of these regions, where color of each inpainted is copied from one of four neighboring projected points: nearest point to the top, bottom, left and right. In order to choose copying direction, depth of each neighbor and distance between each neighbor and point being inpainted are compared. The smaller the distance and the lower depth value, the higher priority has particular neighbor.

III. Implementation

In order to make free navigation possible even for personal computers without powerful graphic cards, the proposed technique was developed and implemented for CPU. It significantly differs our method from existing methods, implemented for GPU [7], [19], [20] or FPGA [21], [22].

A. Single-threaded implementation

The algorithm presented in the previous section has some simplifications, e.g. in filtering and inpainting operations. However, the core of the algorithm could not be performed in the real-time for high resolution sequences.

Let us consider depth projection presented in (1). Even if $\mathbf{P_V} \cdot \mathbf{P_R^{-1}}$ multiplication would be calculated only once for the entire real view, $\mathbf{m} \cdot \mathbf{P_V} \cdot \mathbf{P_R^{-1}}$ would require 16 multiplications and 12 additions for each pixel of each real view. For FullHD sequence it gives over 33M multiplications and almost 25M additions for each real view. Texture projection is performed analogously, so total number of multiplications and additions needed for projection only would reach 133M and 100M, respectively for multiplications and additions. Even assuming only 1 clock cycle for one addition and 1 for multiplication, and 4 GHz CPU, one frame of multiview sequence will be processed in 0.06s, ignoring the entire postprocessing block. It is far away from real-time processing, where (for 25 fps) one frame should be processed in 0.04 s.

In order to reduce number of operations, multiplication (1) was divided into independent parts, e.g. multiplication of the

second column of $\mathbf{P_V} \cdot \mathbf{P_R^{-1}}$ matrix and vertical position of the point (y) is performed only once for entire row of the real view. Also multiplication for entire image columns is performed once and the results are saved in look-up table.

Another time-saving operation is divisions elimination. They are the most time-consuming of all simple mathematical operations, so in the presented algorithm, number of divisions was significantly reduced and most of them were replaced by multiplications or performed only once for the entire view – in the whole view synthesis process only one division operation is performed in the loop for all the points.

As it was presented in the subsection 2B, projecting texture from the real views to the virtual one requires the same amount of operations as depth projection. In the proposed algorithm, this process is simplified – in the step of depth projection for all the points of the virtual view position of corresponding points in both real views is saved in an array. Therefore, instead of many multiplications and additions – texture projection is performed by the simple array reading.

Another small improvement is the color blending for points with similar depth projected from both the real views. This operation was optimized and is performed by the bit shifts.

After texture projection the two postprocessing operations are performed – filtering and inpainting. Both of them are performed in only two additional loops. For the image filtering, in the first loop virtual view is filtered horizontally (in order to remove thin vertical holes), in the second one – vertically for horizontal holes removing. The inpainting operation requires three loops. First two for view analysis, where positions of the nearest projected points in every direction are calculated and the third loop for the actual inpainting processing. However, view analysis for searching left and top nearest projected points is performed simultaneously to the view projection so the entire postprocessing requires only 4 loops on the virtual view.

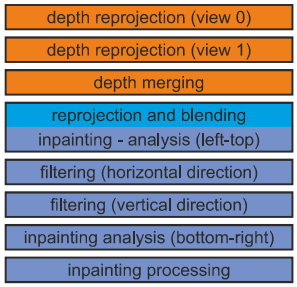

Fig. 3. Single-threaded implementation

B. Vectorized implementation

As a result of profiling of the optimized implementation described in the subsection 3A, the depth map projection has been identified as the most complex operation in virtual view synthesis. Therefore, an effort have been made in order to reduce the computation time related to the depth projection. Since the depth projection consist of significant number of calculations and few memory access operations, we decided to exploit the low-level data parallelism and prepare the vectorized implementation.

The vectorized implementation has been developed by using x86 specific SIMD instructions from AVX, AVX2 [23] and FMA [24] instruction sets. All of the abovementioned instruction sets allow the programmer to use wide 256-bit registers containing 8 single precision floating point numbers. Moreover, the FMA instruction set allows to combine one multiplication and one addition into the single fused multiple-add operation [25]. The combination of multiplication followed by addition repeats very often during matrix multiplication.

The usage of the SIMD instructions results in ability to process 8 values of the source depth at once. As a result the vectorized implementation is significantly faster than the scalar one.

C. Multi-threaded implementation

The last possibility to speed up the synthesis process is to divide the work into multiple processor cores and execute synthesis related operations in parallel. Since the virtual view synthesis seems similar to other image processing workloads, the obvious way of parallelization is to divide the image into segments (tiles, slices, rows, etc.) and perform processing in each row separately. Unfortunately, some of virtual view synthesis related operations cannot be parallelized in such a simple way due to data dependencies. Examples of such an operations are: depth projection and inpainting analysis.

Due to the fact that some operations cannot be easily divided into independent tasks, we designed a hybrid approach to parallelize the virtual view synthesis. Despite the fact that projection of the depth cannot be divided into the separate tasks, the projection is performed separately for every input view, therefore calculations for view 0 and view 1 could be performed in the separate threads. The same approach has been applied for the inpainting analysis, where one thread executes analysis from the left-top origin and second executes analysis from the bottom-right origin.

Remaining operations like texture projection, depth blending, filtering and inpainting could be parallelized by dividing the picture area into the arbitrary number of slices and processing each slice in the separate thread. The multithreaded approach has been illustrated in Fig. 4.

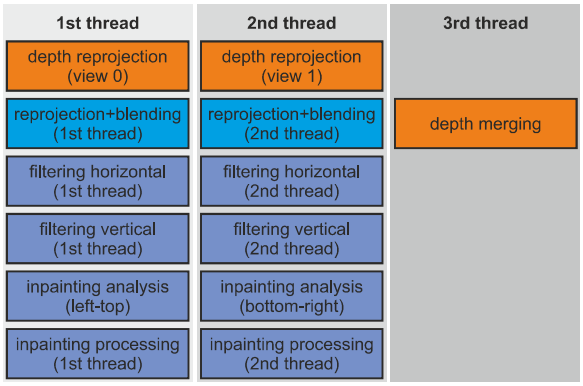

Fig. 4. Multi-threaded implementation

IV. EXPERIMENTAL EVALUATION

A. Processing time estimation

All proposed algorithms and implementations have been evaluated experimentally in order to determine performance and computational complexity.

In order to investigate the influence of the proposed methods into the virtual view synthesis processing time, the four different implementation have been prepared:

- the reference implementation with depth projection implemented without any improvements (as in the subsection 2A),
- the optimized implementation as described in the subsection 3A,
- the vectorized implementation including previously mentioned optimizations and with depth projection implemented using vector operations,
- the multi-threaded implementation based on vectorized one with addition of multi-threaded processing.

The experiments have been performed with the use of Poznan_Basketball and Poznan_Volleyball [26] test sequences (1080p25, complex, difficult to synthesize). We prepared two test scenarios: first with FullHD input and FullHD output and second with FullHD input and output reduced to ¼ input resolution (qHD). The calculations were performed on the desktop computer equipped with 4.0 GHz CPU based on the "Skylake" microarchitecture.

The experimental results have been summarized in TABLE I. For every test scenario and every tested implementation, the average calculation time for each synthesis step is estimated (DP – depth projection, DM – depth merging, VP – view projection, FH – filtering horizontal, FV – filtering vertical, IA – inpainting analysis, IP – inpainting processing).

TABLE I. PROCESSING TIME
(AVERAGED FOR 1500 FRAMES FOR 2 TEST SEQUENCES)

Implementation	Processing time (FullHD→FullHD) [ms]							
	DP	***DM***	***VP***	***FH***	***FV***	***IA***	***IP***	***Total***
Reference	60.44	1.04	8.25	5.95	6.85	1.83	2.33	**86.70**
Optimized	29.77	1.04	8.19	6.05	6.80	1.85	2.35	**56.05**
Vectorized	19.74	1.05	8.18	5.97	6.44	1.87	2.33	**45.58**
Multi-threaded	8.23	4.65		3.34	3.32	2.15	1.43	**23.12**

Implementation	Processing time (FullHD→qHD) [ms]							
	DP	***DM***	***VP***	***FH***	***FV***	***IA***	***IP***	***Total***
Reference	59.91	0.25	2.12	1.18	1.57	0.42	0.41	**65.87**
Optimized	24.79	0.23	2.19	1.19	1.58	0.43	0.41	**30.82**
Vectorized	13.47	0.22	2.22	1.18	1.52	0.42	0.40	**19.42**
Multi-threaded	7.38	1.33		0.67	0.85	0.53	0.25	**11.01**

DP – depth projection, DM – depth merging, VP – view projection,
FH – filtering horizontal, FV – filtering vertical,
IA – inpainting analysis, IP – inpainting processing

All the described improvements allowed to significantly reduce the computational time. With all proposed modifications (algorithmic optimization (subsection 3A), vectorization (subsection 3B) and multi-threading (subsection 3C) we managed to reduce the synthesis time below value required for the real-time processing (40 ms for 25 fps). Theroposed synthesis algorithm is able to compute virtual view in ~23 ms for FullHD output and in ~11 ms for the reduced qHD output.

B. Quality evaluation

The quality of the virtual views synthesized using proposed view synthesis algorithm was compared to the state-of-the-art synthesis method – VSRS [27] (View Synthesis Reference Software).

The results are presented in Table II. For two FullHD test sequences [26] the PSNR value for one reference view was calculated and averaged for 10 s of the sequence (250 frames).

TABLE II. PROPOSED VS. STATE-OF-THE ART METHOD

Test sequence	PSNR [dB]		Processing time for 1 frame [ms]	
	VSRS	***Proposed***	***VSRS***	***Proposed***
Poznan_Basketball	26.76	26.82	2163	23
Poznan_Volleyball	28.33	28.37	2134	23

The quality was also evaluated subjectively during experts' viewing. Performed subjective tests indicated that the quality of views synthesized using proposed approach is noticeably higher, than for reference software.

V. SUMMARY

In this paper, the fast real-time CPU based virtual view synthesis has been presented. The experimental evaluation proves that proposed algorithmic and implementation improvements allows to significantly reduce the FullHD synthesis time and fulfill the real time synthesis requirements, preserving the good quality of the synthesized virtual views.

ACKNOWLEDGEMENT

The presented work has been funded by the Polish Ministry of Science and Higher Education for the status activity consisting of research and development and associated tasks supporting development of young scientists and doctoral students in 2018 in Chair of Multimedia Telecommunications and Microelectronics.

REFERENCES

[1] M. Tanimoto, et. al., "FTV for 3-D spatial communication", *Proc. IEEE*, vol. 100, no. 4, pp. 905-917, 2012.

[2] C.C. Lee, A. Tabatabai, K. Tashiro, "Free viewpoint video (FVV) survey and future research direction", *APSIPA Tr. on Signal and Information Processing*, vol. 4, 10.2015.

[3] C.L. Zitnick, S.B. Kang, M. Uyttendaele, S. Winder, R. Szeliski, "High-quality video view interpolation using a layered representation", *ACM TOG*, vol. 3, s. 600-608, 08.2004.

[4] P. Goorts, M. Dumont, S. Rogmans, P. Bekaert, "An end-to-end system for free viewpoint video for smooth camera transitions", *IC3D 2012*, Liege, Belgium, 3-5.12.2012.

[5] O. Stankiewicz, et. al., "A free-viewpoint television system for horizontal virtual navigation", *IEEE Tr. on Mult*, 2018.
[6] M. Tanimoto, "Overview of FTV (free-viewpoint television)", *ICME 2009*, New York, USA, 28.06-3.07.2009.
[7] L. Do, G. Bravo, S. Zinger, P. de With, "Real-time free-viewpoint DIBR on GPUs for large base-line multi-view 3DTV videos", *VCIP 2011*, Tainan, Taiwan, 6-9.11.2011.
[8] S. Zinger, et. al., "Free-viewpoint depth image based rendering", *J. Vis Comm. & Im. Repr.,* vol. 21, pp. 533-541, 2010.
[9] M. Domański, M. Gotfryd, K. Wegner, "View synthesis for multiview video transmission", *IPCV 2009*, USA, 2009.
[10] L. Du-Hsiu, H. Hsueh-Ming, L. Yu-Lun, "Virtual view synthesis using backward depth warping algorithm", *Picture Coding Symposium, PCS 2013*, San Jose, USA, 2013.
[11] A. Dziembowski, et. al., "Multiview Synthesis – improved view synthesis for virtual navigation", *32nd Picture Coding Symposium, PCS 2016*, Nuremberg, Germany, 4-7.12.2016.
[12] A. Heyden, M. Pollefeys, "Multiple view geometry", in: *Emerging Topics in Comp. Vis*., Prentice Hall, 63-75, 2008.
[13] C. Barnes, et. al., "Patch-Match: a randomized correspondence algorithm for structural image editing", *ACM Transactions on Graphics - TOG*, vol. 28, no. 3, 2009.
[14] M. Bertalmio, G. Sapiro, V. Caselles, "Image inpainting", *SIGGRAPH 2000*, New Orlean, USA, 2000.
[15] N. Komodakis, G. Tziritas, "Image completion using efficient belief propagation via priority scheduling and dynamic pruning", *IEEE Tr. Im. Proc*., vol. 16, pp. 2649-2661, 2007.
[16] K.J. Oh, S. Yea, Y.S. Ho, "Hole filling method using depth based inpainting for view synthesis in free viewpoint television and 3-D video", *PCS 2009*, Chicago, USA, 2009.
[17] A. Criminisi, P. Prez, K. Toyama, "Region filling and object removal by exemplar-based image inpainting", *IEEE Tr. on Image Processing*, vol. 13, no. 9, pp. 1200-1212, 2004.
[18] I. Daribo, B. Pesquet-Popescu, "Depth-aided image inpainting for novel view synthesis", *MMSP 2010*, France, 10.2010.
[19] S. Rogmans, et. al., "Real-time stereo-based view synthesis algorithms: a unified framework and evaluation on commodity GPUs", *Sig. Proc.: Im. Comm.*, vol. 24, pp. 49-64, 2009.
[20] L. Yao, Y. Liu, W. Xu, "Real-time virtual view synthesis using light field", *EURASIP Journal on Image and Video Processing*, vol. 2016, pp. 1-10, 2016.
[21] A. Akin, et. al., "Real-time free viewpoint synthesis using three-camera disparity estimation hardware", *ISCAS 2015*, Lisbon, Portugal, 24-27.05.2015.
[22] J. Wang, L.A. Roeningen, "Real time believable stereo and virtual view synthesis engine for autostereoscopic display", *IC3D 2012*, Liege, Belgium, 3-5.12.2012.
[23] C. Lomont, "Introduction to Intel® Advanced Vector Extensions", *Intel White Paper*, 2011.
[24] E. Quinnell, E.E. Swartzlander, C. Lemonds, "Floating-Point Fused Multiply-Add Architectures", *41 Conf. on Sign., Syst. and Comp*., Pacific Grove, pp. 331-337, 2007.
[25] "IEEE Standard for Floating-Point Arithmetic", *IEEE Std 754-2008*, 1-70, 2008.
[26] M. Domański, et al., "Free-viewpoint television demonstration for sports events", *ISO/IEC JTC1/SC29/ WG11 MPEG2018, M41994*, Gwangju, Korea, 2018.
[27] T. Senoh, et.al, "View Synthesis Reference Software (VSRS) 4.2 with improved inpainting and hole filling", *ISO/IEC JTC1/SC29/WG11 MPEG2013, M40657*, Hobart, Australia, 3-7.04.2017.