MPEG-I DEPTH ESTIMATION REFERENCE SOFTWARE

Ségolène Rogge [*], *Daniele Bonatto* † [*], *Jaime Sancho* ‡,
Rubén Salvador ‡, Eduardo Juarez ‡, Adrian Munteanu [*]and Gauthier Lafruit †

[*] Vrije Universiteit Brussel (VUB), Brussels, Belgium
† Université Libre de Bruxelles (ULB), Brussels, Belgium
‡ Universidad Politécnica de Madrid (UPM), Madrid, Spain
gauthier.lafruit@ulb.ac.be

ABSTRACT

For enabling virtual reality on natural content, Depth Image-Based Rendering (DIBR) techniques have been steadily developed over the past decade, but their quality highly depends on that of the depth estimation. This paper is an attempt to deliver good-quality Depth Estimation Reference Software (DERS) that is well-structured for further use in the worldwide MPEG standardization committee.

The existing DERS has been refactored, debugged and extended to any number of input views for generating accurate depth maps. Their quality has been validated by synthesizing DIBR virtual views with the Reference View Synthesizer (RVS) and the Versatile View Synthesizer (VVS), using the available MPEG test sequences. Resulting images and runtimes are reported.

Index Terms— MPEG, Depth Estimation, RDE, View Synthesis, RVS

1. INTRODUCTION

The Moving Picture Experts Group (MPEG) is a consortium which has the task of standardizing codecs and tools for the video community. The MPEG-I (MPEG-Immersive) subgroup was set up to develop virtual reality techniques that synthesize virtual views from a sparse set of fixed input camera views. Software packages have recently been released, in particular the Reference View Synthesizer (RVS) and the Versatile View Synthesizer (VVS). The MPEG-I group also generated test material for testing and validation.

Estimating high-quality depth remains a delicate process, but it is also crucial for creating synthesized views, where any depth imperfection may have a detrimental impact on the output quality [7, 6]. So far, the Depth Estimation Reference Software (DERS) has been used (and reused from previous activities) for over 7 years and a lot of work has been invested, currently ending up in a quite powerful software: DERS8.0. However, hundreds of man hours of poorly-coordinated work resulted in a hardly extensible software package, with a lot of dead code. We therefore decided to refactor and improve this software - the subject of the present paper - keeping the same overall performance in output quality while obtaining a simpler code.

Finally, aiming at developing the MPEG-I Depth Estimation Reference Software (DERS), we modified the configuration files and the images reading/writing code to be the one of RVS, one of the two formerly cited view synthesis tools. This ensures seamless integration into a full reference software package that can be eventually provided to the MPEG-I community.

ALGORITHM 1
General Overview

Require: R = reference image, I = search images, N = Number of labels
for all $l \in [0, N]$ **do**
  for all $i \in I$ **do**
    for all $p \in R$ **do**
      compute error[p][l][i] based on motion_R[p]
    end for
    E[p][l] = $\min_i$(error[p][l][i])
  end for
end for
graphCut(E)

2. METHOD

The algorithms explained in this section are based on DERS 8.0 [12, 11], which core functionalities are succinctly described. The original DERS8.0 code was refactored, debugged and generalized to work with any number of input views (originally, up to 4 input views were allowed).

The depth estimation algorithm is made of three main parts: the matching cost, the temporal enhancement and the graph cut. As shown in Algorithm 1, we compute an error cost for each pixel of the reference image, using all possible labels (or depth). This error is computed for each search image, and is based on a motion map of the reference image. The final error per pixel and label is set as the minimum error

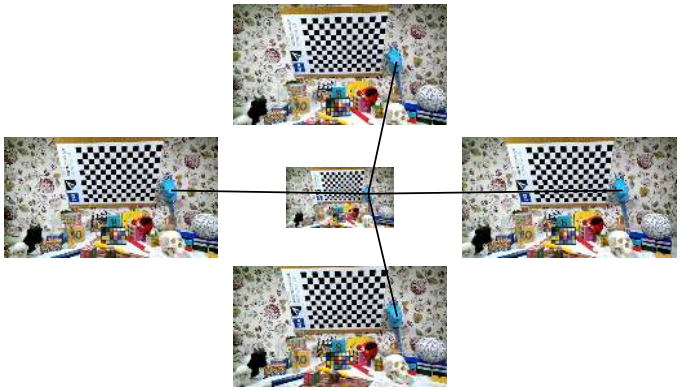


Fig. 1: Cost matching illustration between the reference view with 4 upsampled neighboring search views, on the PlaneA dataset[10].

among the image pairs. This is then given to the graph cut optimization that will find the best depth estimation per pixel.

2.1. Sub-pixel accuracy and image resizing

Baselines can be very different from one dataset to the other, ranging from less than a millimeter to meters. When the baseline is very small, pixels displacement is likely to be smaller than one pixel. Sub-pixel accuracy is therefore needed in order to obtain a good depth estimation. We use a sub-pixel precision by upsampling the input images. Senoh et al. [11] used a bi-cubic interpolation between the pixels with a precision up to $1/8$ pixel.

2.2. Matching cost

This part of the algorithm computes the cost of assigning a label l (representing the depth) to a given pixel (i, j) in the reference image r, leading to a correspondent pixel (u, v) in the search image s. This is done using a modified version of the Sum of Absolute Difference (SAD) between the two images within a 3×3 window w centered on the pixels. Any configuration of input views can be used such as in Fig. 1 or Fig. 2. This cost is composed of three costs, each one based on a different channel of the YUV image :

$$C_{ij}^{sr}(l) = C_Y + C_U + C_V \tag{1}$$

where

$$C_Y = \sum K \odot |Y_w^s(u, v) - Y_w^r(i, j)| \tag{2}$$

$$C_U = 0.15 \times |U^s(u, v) - U^r(i, j)| \tag{3}$$

$$C_V = 0.15 \times |V^s(u, v) - V^r(i, j)| \tag{4}$$

In equation (2), $\odot$ is an element-wise matrix multiplication, and K is a normally distributed kernel used to give more

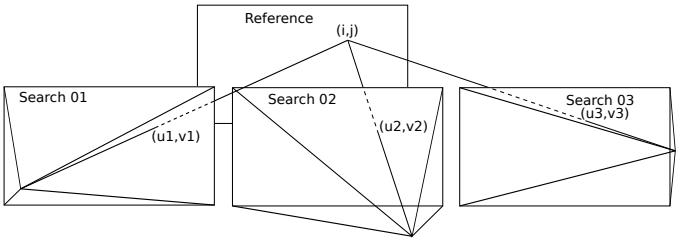


Fig. 2: Cost matching illustration between the reference view with 3 upsampled neighboring search views.

importance to the pixels closer to the center of the window with the following values :

$$K = \begin{pmatrix} 0.05 & 0.09 & 0.05 \\ 0.09 & 0.14 & 0.09 \\ 0.05 & 0.09 & 0.05 \end{pmatrix} \tag{5}$$

This is done for each pixel of the reference image, e.g. the one we want to compute the depth for, with each one of the n search images as shown in Fig. 1. Then, the final cost for the pixel (i, j) and the depth z is the minimal cost among all the search images :

$$C_{ij}(l) = \min_n (C_{ij}^{nr}(l)) \tag{6}$$

Doing this insures that we have found the best corresponding pixel, and not to take into account wrongly matched pixels due to occlusions.

2.3. Temporal coherence

In order to deal with video content, the depth estimation of two consecutive frames has to be coherent. If it was not the case, when using them to synthesize views, the user would see the same pixel in two consecutive frames with variating depth, giving the impression that all pixels' distance is changing over time in the scene leading to a non realistic 3D experience.

This temporal coherence between frames is based on a motion map. If it is not the first frame of the video, we compare its reference image with the one of the previous frame. We take a patch centered on each pixel in both images and compute the average difference in the luma component. If that difference is bigger than a threshold, we set the motion map of all the pixels within that patch to true.

Once we know which pixels have changed during the transition between the two frames, we can enforce non moving pixels to have similar depth values. Indeed, in the case of moving pixel the cost is re-computed as explained in 2.2, while if there was no motion for a given pixel, its cost is computed based on the depth of the previous frame as follows :

$$C_{ij}(l_t) = \begin{cases} 0, & \text{if } l_t = l_{t-1} \\ C_{ij}(l_{t-1}) + |l_t - l_{t-1}|, & \text{otherwise} \end{cases} \tag{7}$$

2.4. Graph cut

A first depth map approximation could be computed in a winner-takes-all approach, by keeping the label with the smallest cost for each pixel. However, this would result in a local per pixel estimation without any depth coherence between neighbors. To obtain a better depth map, we initialize a Markov Random Field (MRF) graph with the costs of each pixels and labels, and use Graph cut with α-expansion as an energy minimization technique to find the best cut in the graph leading to a good global depth estimation [2, 4].

The main point of the graph cut algorithm is to find a satisfying energy function for the problem at hand. We can find a review of useful energy functions for the depth estimation problem in [4] where for a two input views they use:

$$E(d) = \sum_{p \in \mathcal{P}} D(p, d_p) + \sum_{q \in \mathcal{N}} K_{(p,q)} \cdot T(d_p \neq d_q) \qquad (8)$$

Where d_p is the disparity label of a pixel p in the set of pixels $\mathcal{P}$ in the left image and D(p,d) the cost for assigning it the label d. q i

However, this simple approach doesn't take into account the occlusions nor the n-view camera problem. Kolmogorov et al. [9] solved the two problems at the same time using:

$$E(f) = E_{\text{data}}(f) + E_{\text{smoothness}}(f) + E_{\text{visibility}}(f) \qquad (9)$$

where $E_{\text{data}}(f)$ is the photo-consistency:

$$E_{\text{data}}(f) = \sum_{<p_1, f(p)>, <q, f(q)> \in I} D(p, q) \qquad (10)$$

$E_{\text{smoothness}}(f)$ a smoothness parameter to improve the coherence between neighboring pixels and $E_{\text{visibility}}(f)$ increase the error if the pixel is occluded.

Instead of this visibility parameter, our function uses a reliability map and a smoothing map that will be explained in Section 2.5.

Our energy can be expressed as:

$$E^*(l) = D^* \times M_R \times C_{ij}(l) \qquad (11)$$

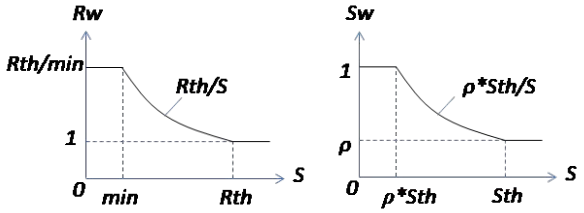
Where $E^*(l)$ is the smallest error among the different search images for label l. It is computed using D^* as the baseline between the reference image and the search image from which we selected $E^*(l)$, M_R as the reliability map and $C_{ij}(l)$ as the matching costs found in (6).

The smoothing is inserted as a new node in the MRF representation between the source and the E_{data} node. It is computed using:

$$E_{\text{smoothing}}(l) = \lambda_s \times E_d \qquad (12)$$

Where E_d is a factor giving more importance to close labels. The λ_s is a factor which represents the importance of the smoothing error term against the data term in the optimization process.

Further parallelizations will be made to faster the code in a future iteration.



(a) Reliability weight (b) Smoothing weight

Fig. 3: Reliability and Smoothing weights[11]

2.5. Reliability and smoothing map

When an image is untextured, finding the best match is a challenge as multiple depth level will have the same error cost. To avoid this, [11] used a reliability map which weights the matching error based on the color slopes between adjacent pixels of the reference image. It can be seen in Fig. 3a that the reliability weight decreases when the slope increases. In a very textured part of the image, the color slope will be higher and the estimation is likely to be correct; therefore, we put a small weight on the matching cost of that pixel. Algorithm 2 describes the procedure to compute the reliability map where gapY is a weighted averages of the pixels in a 3×3 window and gapU, gapV are left/right differences for an horizontal pair and up/bottom for a vertical pair and E_p is the final matching error for pixel p. The same approach is used to compute the smoothing map.

ALGORITHM 2
Reliability map computation

Require: T = threshold, min = arbitrary minimum (0.3),
  for all $p \in$ ReferenceView **do**
    gap = gapY + gapU + gapV
    if gap $<$ min **then**
      $W = \frac{T}{\min}$
    end if
    if min $\leq$ gap $< T$ **then**
      $W = \frac{T}{gap}$
    end if
    if $T \leq$ gap **then**
      $W = 1$
    end if
    if gap $== 0$ and $E == 0$ **then**
      $E_p = \frac{W}{\min}$
    end if
  end for

3. DATASETS AND CONFIGURATION FILES

Our algorithm was tested on several well known MPEG datasets, whose properties are shown in Table 1. In addition, all the configuration files and camera parameter files are included in an online repository[1].

4. RESULTS

The quality of RDE depth maps was tested synthesizing virtual views using both RVS and VVS for the frames of interest. Then, these views were compared to their corresponding original views using the Weighted-to-Spherically-Uniform Peak-Signal-to-Noise-Ratio (WS-PSNR) metric [1]. With these values, an average PSNR was calculated for each dataset generating the final results included in Table 2. The same process was followed to obtain the DERS PSNR results, which were used to obtain the delta PSNR value in percentage (with DERS 8.0 as reference).

These results show that the two algorithms obtain a PSNR difference below 5%, both for VVS and RVS, which have similar results for all the datasets. These slight differences between synthesis programs are not considered important given they are the result of an averaging process. This means that, for the same dataset, some views obtained better PSNR results using VVS whilst others obtained better results using RVS.

As an example, Fig. 4 provides one depth map for each one of the datasets tested. It shows the subjective degree of quality and the smoothness achieved with RDE.

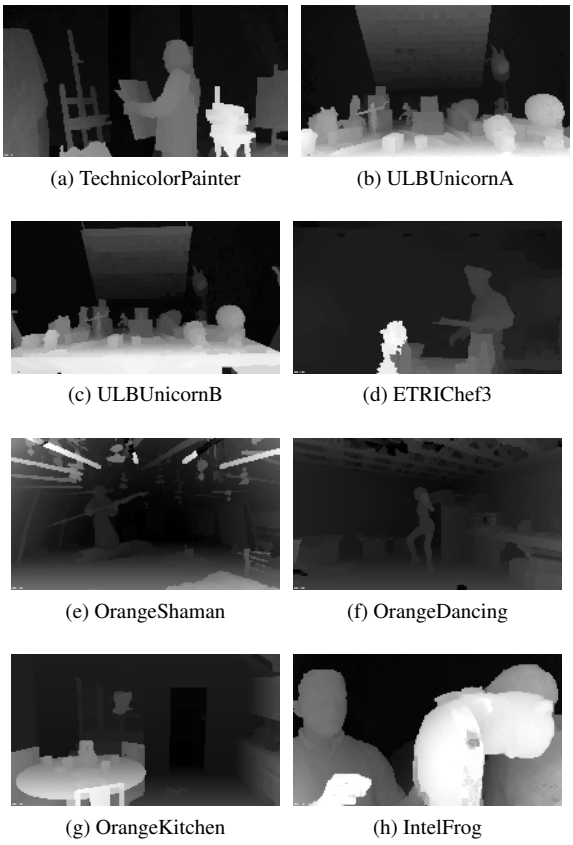
In addition, Fig. 5 depicts the synthesis detail for one view within the TechnicolorPainter dataset. In this example, the differences between VVS and RVS can be noticed in the background green strip and also in the edges of the man's face. Although these differences are subtle, they show how the different synthesizers treat tough areas of the image and explain the different PSNR values obtained in Table 2.

Figure 5 also displays the subjective level of quality for VVS and RVS synthesis provided that the detail chosen is considered to be the most noticeable error in the view.

Finally, this work includes the time consumed by RDE, DERS 8.0, VVS and RVS in order to provide insights on the temporal magnitude order of the problem. All the tests were made using a Linux 18.04 computer with an Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz and 64 GB GDDR4 RAM @ 2133 MHz.

The results obtained are gathered in Table 3. On the one hand, the time results for RDE and DERS 8.0 are divided into (i) the time needed per view, (ii) the time needed for the first frame and (iii) the time needed for one of the following frames. On the other hand, the time results for VVS and RVS only include (i) the time needed per view and (ii) per frame. This difference is given by the temporal enhancement

[1] https://cutt.ly/bjUxSR



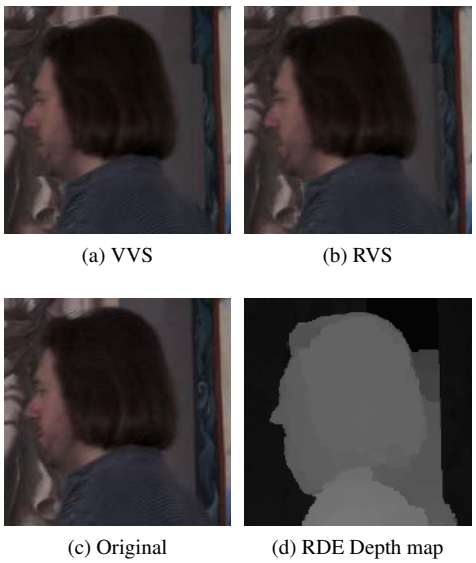
(a) TechnicolorPainter (b) ULBUnicornA

(c) ULBUnicornB (d) ETRIChef3

(e) OrangeShaman (f) OrangeDancing

(g) OrangeKitchen (h) IntelFrog

Fig. 4: Depth maps for one frame for the tested datasets.



(a) VVS (b) RVS

(c) Original (d) RDE Depth map

Fig. 5: Detail of TechnicolorPainter for the images generated by VVS and RVS along with the original view and the depth map generated by RDE.

Dataset	Scene type	Configuration	Frames	Frames of interest	Resolution
TechnicolorPainter	natural	array 4×4	371 (30 fps)	60-67 (8)	2048×1088
ULBUnicornA	natural	array 5×5	1 (1)	1 (1)	1920×1080
ULBUnicornB	natural	array 5×3	1 (1)	1 (1)	1920×1080
ETRIChef3	natural	array 5×5	300 (30 fps)	60-67 (8)	1920×1080
OrangeShaman	synthetic	array 5×5	300 (30 fps)	20-27 (8)	1920×1080
OrangeDancing	synthetic	arc 14×3	300 (30 fps)	50-57 (8)	1920×1080
OrangeKitchen	synthetic	array 5×5	90 (30 fps)	50-57 (8)	1920×1080
IntelFrog	natural	linear 14×1	300 (30 fps)	140-147 (8)	1920×1080

Table 1: Overview of the used datasets to synthesize novel depths[10, 8].

Dataset	RDE						DERS 8.0					
	VVS PSNR (dB)			RVS PSNR (dB)			VVS ΔPSNR (%)			RVS ΔPSNR (%)		
	Y	U	V	Y	U	V	Y	U	V	Y	U	V
TechnicolorPainter	37.138	34.679	37.424	36.771	34.719	37.699	0.54	2.09	1.26	0.91	3.90	3.67
ULBUnicornA	31.155	44.295	44.402	30.667	44.694	44.750	-0.52	-0.17	-0.05	-0.58	-0.21	-0.13
ULBUnicornB	31.364	44.102	44.395	31.425	44.631	44.869	-1.61	-0.02	-0.07	-0.68	0.14	0.05
ETRIChef3	33.940	40.748	41.466	33.571	40.968	41.564	-1.66	-2.07	-0.86	-3.37	-1.00	-0.38
OrangeShaman	39.304	49.326	47.230	39.576	49.940	47.963	3.39	3.00	3.68	3.17	3.28	4.26
OrangeDancing	31.783	48.910	50.403	32.327	49.472	50.911	1.75	2.08	1.95	1.28	1.70	1.70
OrangeKitchen	31.878	47.081	50.107	33.093	48.288	51.126	0.12	-0.42	-0.90	1.57	-0.13	-0.52
IntelFrog	26.861	41.182	40.140	26.821	41.117	39.837	-0.02	0.51	2.39	-0.42	0.15	2.02

Table 2: Average PSNR results for VVS and RVS for the used datasets.

Dataset	RDE time (s)			DERS 8.0 time (s)			VVS time (s)		RVS time (s)	
	View	Frame 0	Frame 1+	View	Frame 0	Frame 1+	View	Frame	View	Frame
TechnicolorPainter	3307	678	380	1726	546	165	111	14	135	17
ULBUnicornA	1075	1075	1075	718	718	718	8	8	15	15
ULBUnicornB	832	832	832	810	810	810	7	7	15	15
ETRIChef3	4239	1025	461	1706	753	136	170	21	152	19
OrangeShaman	2018	710	193	1997	1438	78	225	28	128	16
OrangeDancing	2644	912	256	4440	2585	255	307	39	136	17
OrangeKitchen	2746	1085	239	1942	1165	111	88	11	130	16
IntelFrog	4598	846	530	3840	866	436	141	18	160	20

Table 3: Results of time for the tested datasets. Synthesis times for VVS and RVS refers to the time needed by these synthesizers using RDE depthmaps. The time consumed by them when using DERS 8.0 depthmaps is similar, and it is not included here due to space limitations.

feature that implements RDE and DERS 8.0, speeding up the calculation of the consecutive frames. Despite this fact, the time consumed by RDE and DERS 8.0 is by far the largest, needing more than half an hour to generate a sequence of 8 frames. In the case of the synthesizers, the time per frame is around one order of magnitude less, with more variable times for VVS than for RVS.

We have shown that RDE and DERS 8.0 give similar quality results. However, the amount of code was divided by a factor 10, and no speed optimization was done for the moment. This will be done in a future iteration of this work.

5. CONCLUSIONS

This work presents a refactored, debugged and well structured depth estimation software called Reference Depth Estimation (RDE), which offers equivalent results to the MPEG Depth Estimation Reference Software (DERS 8.0). This is shown by applying two view synthesis algorithms to well-known datasets and both DERS and RDE computed depth maps. This improves the analysis and understanding of the code, allowing the interest of new researchers in the tool. This first step toward a robust and easy to use depth estimation algorithm shows promising results and improves the potential of future extensions.

COPYRIGHTS

ACKNOWLEDGEMENT

One of the authors is a FWO-SB PhD fellow funded by the Fund for Scientific Research-Flanders (FWO), project number 1S83118N. This work is also supported by the FNRS, National Fund of Scientific Research, Belgium, through project Colibrih under contract No.: CDR J.0096.19, by the Spanish Government through PLATINO project (TEC2017-86722-C4-4-R) and the by the Regional Government of Madrid through NEMESIS-3D-CM project (Y2018/BIO-4826).

6. REFERENCES

[1] WS-PSNR Software Manual [N18069]. *ISO/IEC JTC1/SC29/WG11*, October 2018.

[2] Andrew Blake, Pushmeet Kohli, and Carsten Rother. *Markov Random Fields for Vision and Image Processing*. The MIT Press, 2011.

[3] Daniele Bonatto, Arnaud Schenkel, Tim Lenertz, Yan Li, and Gauthier Lafruit. ULB High Density 2d/3d Camera Array data set, version 2 [M41083]. *ISO/IEC JTC1/SC29/WG11*, July 2017.

[4] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, September 2004.

[5] D. Doyen, T. Langlois, B. Vandame, F. Babon, G. Boisson, N. Sabater, R. Gendrot, and A. Schubert. Light field content from 16-camera rig [M40010]. *ISO/IEC JTC1/SC29/WG11*, January 2017.

[6] S. Fachada, D. Bonatto, A. Schenkel, and G. Lafruit. Depth image based view synthesis with multiple reference views for virtual reality. In *2018 - 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4, June 2018.

[7] S. Fachada, D. Bonatto, A. Schenkel, and G. Lafruit. Free navigation in natural scenery with DIBR: RVS and VSRS in MPEG-I standardization. In *2018 International Conference on 3D Immersion (IC3D)*, pages 1–6, Dec 2018.

[8] Joel Jung, Bart Kroon, Renaud Doré, Gauthier Lafruit, and Jill Boyce. CTC on 3dof+ and Windowed 6dof (v2) [N17726]. *ISO/IEC JTC1/SC29/WG11*, July 2018.

[9] Vladimir Kolmogorov and Ramin Zabih. Multi-camera scene reconstruction via graph cuts. In *Proceedings of the 7th European Conference on Computer Vision-Part III*, ECCV '02, pages 82–96, Berlin, Heidelberg, 2002. Springer-Verlag.

[10] A. Schenkel, D. Bonatto, S. Fachada, H. Guillaume, and G. Lafruit. Natural scenes datasets for exploration in 6dof navigation. In *2018 International Conference on 3D Immersion (IC3D)*, pages 1–8, Dec 2018.

[11] T. Senoh, N. Tetsutani, and H. Yasuda. Depth estimation and view synthesis for immersive media. In *2018 International Conference on 3D Immersion (IC3D)*, pages 1–8, Dec 2018.

[12] T. Senoh, K. Wakunami, H. Sasaki, R. Oi, and K. Yamamoto. Fast depth estimation using non-iterative local optimization for super multi-view images. In *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1042–1046, Dec 2015.

[13] Olgierd Stankiewicz, Marek Domanski, Adrian Dziembowski, Adam Grzelka, Dawid Mieloch, and Jaroslaw Samelak. A Free-Viewpoint Television System for Horizontal Virtual Navigation. *IEEE Transactions on Multimedia*, 20(8):2182–2195, August 2018.