3D Scene Modeling from Dense Video Light Fields

Xiaoran Jiang, Christian Galea, Laurent Guillo, Christine Guillemot

3D SCENE MODELING FROM DENSE VIDEO LIGHT FIELDS

Xiaoran Jiang, Christian Galea, Laurent Guillo, Christine Guillemot

INRIA, CNRS, IRISA
Campus de Beaulieu
Rennes, France

ABSTRACT

Light field imaging offers unprecedented opportunities for advanced scene analysis and modelling, with potential applications in various domains such as augmented reality, 3D robotics, and microscopy. This paper illustrates the potential of dense video light fields for 3D scene modeling. We first recall the principles of plenoptic cameras and present a downloadable test dataset captured with a Raytrix 2.0 plenoptic camera. Methods to estimate the scene depth and to construct a 3D point cloud representation of the scene from the captured light field are then described.

***Index Terms*—** Light field video, plenoptic cameras, depth maps, 3D point clouds.

1. INTRODUCTION

Light field imaging has recently gained in popularity, both at the research and industrial level. Targeted applications include depth estimation, change of viewpoint and view synthesis, augmented reality content capture or post production. Capturing devices range from cameras arrays [1] to camera gantries in which a single camera moves along a plane and takes captures at regular time intervals, and to plenoptic cameras [2, 3]. Two optical designs have been considered for plenoptic cameras, the so-called plenoptic 1.0 design, called unfocused plenoptic camera, in which the main lens focuses the subject on the lenslet array [2], and the plenoptic 2.0 design [3], also called focused plenoptic camera, in which the image plane of the main lens is the object plane of the lenslet array.

Considering the simplified 4D representation proposed in [4] and [5], a light field can be seen as capturing an array of viewpoints (called sub-aperture images in the case of microlens based capturing devices) of the imaged scene. Camera arrays are naturally designed to capture the set of views, thus offering a high spatial resolution for each view but a low angular resolution (limited set of views), and hence a large baseline. In contrast, plenoptic cameras use an array of microlenses placed in front of the photosensor to separate the light rays striking each microlens into a small image on the photosensor, and this way capture dense angular information with a small baseline.

In this paper, we used the Raytrix R8 focused video light field camera to capture test video light fields, and we illustrate some possibilities offered by such image modality in terms of scene depth estimation and 3D point clouds scene representation. The acquisition set-up and the captured dataset, that is available for download, are described in Section 2. A method for estimating a disparity (or equivalently depth) map for every viewpoint is presented in Section 3, while most state-of-the-art methods, often operating in the epipolar plane images, produce a depth map for the central view only [6–11]. For each viewpoint, one can then construct a 3D point cloud using the corresponding estimated depth maps, while the point clouds at each viewpoint may also be aligned with each other, as explained in Section 4.

2. SCENE CAPTURE WITH AN R8 RAYTRIX VIDEO LIGHT FIELD CAMERA

In order to provide input data to the algorithms to generate disparity maps and 3D point clouds, several scenes have been captured as dense video light fields. The sections below describe the video plenoptic camera we used, the characteristics of the scenes and how the 5×5 viewpoints are extracted from the raw data.

2.1. The R8 Raytrix plenoptic camera

The video plenoptic camera we used to capture the test scenes is a R8 Raytrix plenoptic video camera [12] with a C-mount Ricoh lens FL-BC3518-9M. Its focal length is 35mm and its maximum aperture ratio is 1:1.8. The resolution of the sensor is 3840×2160 pixels (Ultra HD). The micro lens array (MLA) contains three kinds of lenses with different focal lengths. The Raytrix API [13] provides more details about the characteristics. They are listed in the Table 1 below:

This work has been supported by the EU H2020 Research and Innovation Programme under grant agreement No 694122 (ERC advanced grant CLIM).

Sensor size (widthxheight)	21.12x11.88 mm
Sensor physical pixel size	0.0055 mm
Width of raw image	3840 pixels
Height of raw imahe	2160 pixels
MLA lens pitch	34.9767 pixels
MLA lens pitch	0.19 mm
Lens MLA sensor distance	0.222 mm
MLA lens type count	3
Lens #0 MLA focus range min.VD	1
Lens #0 MLA focus range max.VD	3.2
Lens #1 MLA focus range min.VD	2.6
Lens #1 MLA focus range max.VD	4.2
Lens #2 MLA focus range min.VD	3.5
Lens #2 MLA focus range max.VD	100

Table 1: Sensor and MLA characteristics of the R8 Raytrix video camera

2.2. Captured scenes

Three different scenes have been captured with the R8 Raytrix camera: ChessPieces, Boxer-IrishMan-Gladiator and Chess Pieces-MovingCamera. The R8 Raytrix camera can accurately capture a scene included in a working volume, the size of which depends on parameters such as the focal length of the main lens and the distance between the camera and the scene. Our experiments have shown that this volume has roughly a $10\times10\times10$ cm size when the R8 camera is equipped with a 35mm main lens. For the first two scenes, this volume is centered on a turning table with pieces or characters on it and with a calendar as background. The camera is located 35cm from the scene. For the third scene, the scene is still and is composed of chess pieces with a calendar as background. The camera moves alongside the scene such that the distance between the camera and the background remains constant, at 60cm. All these videos are composed of 300 frames at a frame rate of 30fps. The first frames for each video are depicted in Table 2.

2.3. Extracting views from raw data

The RxLive Raytrix software manages the R8 camera and records the raw data which can be processed to obtain 2D images. We only use it to save the raw data as a sequence file. We used the Raytrix API to access each Raytrix frame in the sequence file in order to extract 25 views as if we had twenty-five virtual cameras (5 rows $\times$ 5 columns). Each virtual camera is 4mm away from its neighbours as depicted in Figure 1.

To extract views, we take advantage of the Raytrix API and the function "RX::ApiLF::RxSetPar". Called with the parameter "Epar::VirtCamPinHoleStd_ViewOffsetMM_g", it allows the specification of an offset of a virtual camera in the plane perpendicular to the z-axis of main lens. The offset has a value of 4mm in both the horizontal and vertical directions, and is measured from the center of the main lens.

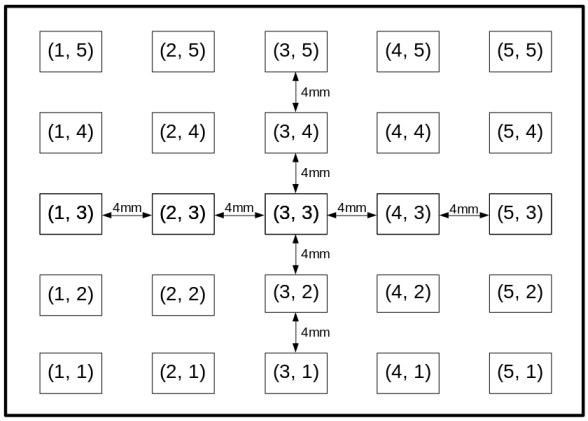

Fig. 1: 25 virtual cameras per frame, one for each extracted view

3. DISPARITY ESTIMATION

In order to construct the 3D model of the scene, the depth information must first be estimated, which can be derived from the disparity between the different views. The algorithm used for estimating a disparity map for each light field view proceeds as follows. We first estimate the disparity between the four corner views of the light filed using an optical flow estimator. These disparity maps are then propagated to the other viewpoints using forward warping and any holes arising due to occlusions are filled using a low rank matrix completion method. A Total Variation (TV) regularization is finally applied in the Epipolar Plane Image (EPI) domain of the resulting disparity maps to enforce view consistency.

3.1. Optical flow

Captured by a Raytrix lenslet camera, the light field videos in our dataset are of narrow baseline (less than 2 pixels between adjacent views). In this case, the views on the extreme corners arguably contain all color and geometric information, from which the whole light field can be reconstructed. Therefore, we choose to only exploit the corner views to infer disparity maps for every viewpoint.

An optical flow estimator is used to compute disparity between the corner views. In this work, the learning-based optical flow estimator FlowNet 2.0 [14] is selected for its accuracy and time efficiency. Taking the view $L_{1,1}$ as an example, FlowNet 2.0 is performed three times, between the aforementioned view and the view $L_{1,5}$, $L_{5,1}$ and $L_{5,5}$. We let $D_x^{(i,j)\rightarrow(i',j')}$ and $D_y^{(i,j)\rightarrow(i',j')}$ denote the optical flow components from view $L_{i,j}$ to view $L_{i',j'}$, on axis x and y respectively. A set of 4 estimates is therefore obtained at view $L_{1,1}$:

$\{D_x^{(1,1)\to(5,1)}, D_y^{(1,1)\to(1,5)}, D_x^{(1,1)\to(5,5)}, D_y^{(1,1)\to(5,5)}\}$ (since the light field views are rectified, a scene point moves only horizontally from position $(1,1)$ to $(5,1)$, and only vertically from position $(1,1)$ to $(1,5)$, thus the two optical flow components $D_x^{(1,1)\to(1,5)}$ and $D_y^{(1,1)\to(5,1)}$ should equal to zero). Moreover, since our light field is a square grid of 5×5 views, and we assume that the horizontal and vertical baseline between views is identical, these estimates are considered to reveal the same disparity information.

As proposed in our recent work [15], these disparity maps can be fused to a single map within an energy minimization framework, and the resulting maps can be furthermore enhanced by a superpixel-based edge-preserving filtering. However, for the generation of our dataset, which contains 3 light field videos of 300 frames each, each frame being a light field of 5×5 views of 1920×1080 pixels, this process becomes time consuming. Thus, a simple average of the aforementioned 4 estimates is performed to obtain a single disparity map $\breve{D}_{1,1}$. The same process is performed at the three other corner views. These maps are then normalized by dividing them by the number of angular intervals between corner views such that they represent disparity between adjacent views, assuming a constant baseline.

3.2. Propagation and inpainting

The obtained disparity maps $\breve{D}_{\mathbf{r}}$, $\mathbf{r} = (u_r, v_r)$ corresponding to the 4 input corner positions are projected to a novel position $\mathbf{s} = (u_s, v_s) \in [\![1 \,..\, 5]\!] \times [\![1 \,..\, 5]\!]$ by using the disparity information itself. Forward warping is applied:

$$\breve{D}_{\mathbf{s}}^{\mathbf{r}}(x', y') = \breve{D}_{\mathbf{r}}(x, y), \tag{1}$$

with

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} u_s - u_i \\ v_s - v_i \end{pmatrix} \cdot \breve{D}_{\mathbf{r}}(x, y). \tag{2}$$

$\breve{D}_{\mathbf{s}}^{\mathbf{r}}$ denotes the warped disparity map from position $\mathbf{r}$ to $\mathbf{s}$. The projected pixel (x', y') in this destination image may fall onto non integer coordinates, and interpolation should be performed to retrieve values on integer coordinates. It is well known that the resulting interpolated image will have cracks and holes. The former is caused since an integer coordinate may not be the nearest neighbor of any projected coordinates. The latter results from the fact that a part of the scene which is revealed in the projected view can not be seen in the reference view. Therefore, we also define a binary mask $M_{\mathbf{s}}^{\mathbf{r}}$, with value 0 at the missing value (cracks + holes) positions, and 1 anywhere else.

At each position $\mathbf{s}$ of the light field, we thus obtain 4 pairs of disparity maps, warped from the 4 corners, and their corresponding masks ($\breve{D}_{\mathbf{s}}^{\mathbf{r}}, \breve{M}_{\mathbf{s}}^{\mathbf{r}}$). Considering the strong correlation between neighboring viewpoints of a light field, the missing values in the warped maps may be found not only within the 4 maps in the same position, but also within other maps in some neighboring positions. We thus construct the matrix $\breve{H}$ of 4×25 columns (25 views in each light field video frame), each column being a vectorized warped disparity map:
$\breve{H} = \Big[\text{vec}(\breve{D}_{\mathbf{s_1}}^{\mathbf{r_1}}) \,|..|\, \text{vec}(\breve{D}_{\mathbf{s_1}}^{\mathbf{r_4}}) \,|..|\, \text{vec}(\breve{D}_{\mathbf{s_{25}}}^{\mathbf{r_1}}) \,|..|\, \text{vec}(\breve{D}_{\mathbf{s_{25}}}^{\mathbf{r_4}})\Big]$.
Correspondingly, the mask matrix indicating the position of missing values (due to occlusions) is also defined:
$\breve{M} = \Big[\text{vec}(\breve{M}_{\mathbf{s_1}}^{\mathbf{r_1}}) \,|..|\, \text{vec}(\breve{M}_{\mathbf{s_1}}^{\mathbf{r_4}}) \,|..|\, \text{vec}(\breve{M}_{\mathbf{s_{25}}}^{\mathbf{r_1}}) \,|..|\, \text{vec}(\breve{M}_{\mathbf{s_{25}}}^{\mathbf{r_4}})\Big]$.

Given the small disparity between views, the matrix $\breve{H}$ is assumed to be of low rank. The missing disparity values, due to occlusions, can therefore be recovered using a low rank matrix completion approach [17] which searches for a low rank matrix $\hat{H}$ such that

$$\begin{aligned} &\min_{\hat{H}} \text{rank}(\hat{H}) \\ &\text{s.t. } P_{\breve{M}}(\hat{H}) = P_{\breve{M}}(\breve{H}), \end{aligned} \tag{3}$$

where $P_{\breve{M}}$ is the sampling operator such that $P_{\breve{M}}(H)_{i,j}$ is equal to $H_{i,j}$ if $\breve{M}_{i,j} = 1$, and zero otherwise. This inpainting problem is solved efficiently using the Inexact ALM (IALM) method [17]. The disparity values are well recovered in practice because the occlusion zones at different viewpoints are unlikely to overlap.

To obtain one single disparity map per viewpoint, a simple average is computed: $\hat{D}_{\mathbf{s}} = mean(\hat{D}_{\mathbf{s}}^{\mathbf{r_1}}, ..., \hat{D}_{\mathbf{s}}^{\mathbf{r_4}})$. And finally, in order to enforce angular consistency of the final disparity maps of the whole light field scene, a step of total variational regularization (TV-L1) [18] is applied on epipolar plane images of the resulting disparity maps.

4. 3D POINT CLOUDS CONSTRUCTION

Each pixel with coordinates $\mathbf{k} = (x, y)$ in a disparity map $\hat{D}_{\mathbf{s}}$ as obtained with the method described in Section 3 may be projected into a point in 3-D space having coordinates $(X_{\mathbf{k}}, Y_{\mathbf{k}}, Z_{\mathbf{k}})$ using an approach similar to that described in [19]. Specifically, the 3-D point coordinates can be derived from the view pixel coordinates using the pinhole camera model [20] as follows:

$$X_{\mathbf{k}} = \frac{(\frac{x}{W-1} - \frac{1}{2}) \cdot ss \cdot Z_{\mathbf{k}}}{f} \tag{4}$$

$$Y_{\mathbf{k}} = \frac{(\frac{y}{H-1} - \frac{1}{2}) \cdot ss \cdot Z_{\mathbf{k}}}{f} \tag{5}$$

where ss is the sensor size of the larger of both dimensions (mm), f is the focal length (mm), $x \in [0, W-1]$ and $y \in [0, H-1]$ are the column and row indices, respectively, of the pixel in view $L_{i,j}$ being considered, and $W = 1920$ pixels and $H = 1080$ pixels are the width and height of $L_{i,j}$, respectively, such that ss/W and ss/H give the pixel size and $W/2$ places the origin at the centre of the image.

Fig. 2: Examples of the light fields in the new dataset, and the corresponding disparity maps and point clouds: the fist row corresponds to the first frame of the central view of each of the three video light fields, the second row to the associated disparity maps, and the third row to the corresponding 3D point clouds, viewed with the MeshLab software program [16].

$Z_{\mathbf{k}}$ can be similarly derived from the pinhole camera model, accounting for the translation between the cameras and the fact that the disparity is equal to zero at the focal plane (i.e. when $Z = f_d$, where fd is the focus distance (mm)). Thus, $Z_{\mathbf{k}}$ may be formally defined as follows:

$$Z_{\mathbf{k}} = \frac{b \cdot f \cdot fd \cdot max(H, W)}{d_{\mathbf{k}} \cdot fd \cdot ss + b \cdot f \cdot max(H, W)} \tag{6}$$

where b is the baseline (in millimetres, mm).

–

Each pixel with coordinates $\mathbf{k} = (x, y)$ in a disparity map $\hat{D}_{\mathbf{s}}$ as obtained with the method described in Section 3 may be projected into a point in 3-D space having coordinates $(X_{\mathbf{k}}, Y_{\mathbf{k}}, Z_{\mathbf{k}})$ using an approach similar to that described in [19]. Specifically, given a disparity value $d_{\mathbf{k}}$ in $\hat{D}_{\mathbf{s}}$ representing the shift of a pixel in one view to an adjacent view, the depth value $Z_{\mathbf{k}}$ may be computed as follows:

$$Z_{\mathbf{k}} = \frac{b \cdot f \cdot fd \cdot max(H, W)}{d_{\mathbf{k}} \cdot fd \cdot ss + b \cdot f \cdot max(H, W)} \tag{7}$$

where $x \in [0, W-1]$ and $y \in [0, H-1]$ are the column and row indices, respectively, of the pixel in view $L_{i,j}$ being considered, $W = 1920$ pixels and $H = 1080$ pixels are the width and height of $L_{i,j}$, respectively, b is the baseline (in millimetres, mm), f is the focal length (mm), fd is the focus distance (mm), and ss is the sensor size of the larger of both dimensions (mm).

The $X_{\mathbf{k}}$-coordinates may then be obtained as follows:

$$X_{\mathbf{k}} = \frac{(\frac{x}{W-1} - \frac{1}{2}) \cdot ss \cdot Z_{\mathbf{k}}}{f} \tag{8}$$

Similarly, the $Y_{\mathbf{k}}$-coordinates are given by:

$$Y_{\mathbf{k}} = \frac{(\frac{y}{H-1} - \frac{1}{2}) \cdot ss \cdot Z_{\mathbf{k}}}{f} \tag{9}$$

With the above procedure, the $(X_{\mathbf{k}}, Y_{\mathbf{k}}, Z_{\mathbf{k}})$ coordinates of a pixel $\mathbf{k} = (x, y)$ given disparity value $d_{\mathbf{k}}$ are now available. Repeating this process for all pixels within $L_{i,j}$ yields a set of points that is generally referred to as a *point cloud*. Point clouds may then be obtained for each of the $H_{LF} \times W_{LF}$ views.

At this juncture, points in different views are not aligned such that those corresponding to the same area of an object coincide. Hence, the $X_{\mathbf{k}}$- and $Y_{\mathbf{k}}$- coordinates need to be shifted such that the point clouds from different views are

aligned to a common coordinate system. Given a point cloud in view (u, v), where $u \in [1, W_{LF}]$ and $v \in [1, H_{LF}]$ are the column and row indices, respectively, of a view that needs to be aligned with the points in view (u_c, v_c), and $W_{LF} = H_{LF} = 5$ represent the number of views in the horizontal and vertical directions, respectively, then Equations (8) and (9) may be slightly modified to represent the shift required for the $X_{\mathbf{k}}$- and $Y_{\mathbf{k}}$-coordinates, respectively, as follows:

$$X_{\mathbf{k}} = X_{\mathbf{k}} + \frac{d_{\mathbf{k}} \cdot ss \cdot Z_{\mathbf{k}} \cdot (u_c - u)}{f \cdot (W - 1)} \tag{10}$$

$$Y_{\mathbf{k}} = Y_{\mathbf{k}} + \frac{d_{\mathbf{k}} \cdot ss \cdot Z_{\mathbf{k}} \cdot (v_c - v)}{f \cdot (H - 1)} \tag{11}$$

5. DISCUSSION AND CONCLUSION

While there exist datasets acquired with rigs of video cameras [21], to the best of our knowledge there do not exist publicly available dense light field videos capturing real scenes. The dataset that we created with the R8 Raytrix video camera allows us to extract 25 views per frame and to generate disparity maps and 3D point clouds as shown in Figure 2. The dataset (raw data, extracted views, corresponding disparity maps, and animations demonstrating the variation of the point clouds across the video frames) can be downloaded at `http://clim.inria.fr/DataSoftware.html`.

The minimal and maximal disparity values are very low as shown in Table 2. These low values are related to the density of the light field and to the distance between virtual cameras. When the distance is larger (e.g. 8mm), artefacts appear in the views of the border as shown in Figure 3. That is why we kept this distance as low as possible. However, the algorithms that we presented show that it is still possible to exploit views with low disparities in order to generate, for instance, disparity maps or 3D point clouds. Moreover, the presented techniques are efficient, such that the disparity maps and point clouds can be obtained quite rapidly. The distortion of the 3D point cloud Figure 2 bottom right is mainly due to the high distance of the camera to the scene which is not compatible with the working volume and the distance required for our R8 Raytrix camera equipped with a 35mm main lens.

In the current stage of works, disparity maps and 3D point clouds are computed frame by frame. The temporal consistency could be better exploited in the future work in order to obtain more accurate 3D scene reconstruction. An improvement in the depth map estimation, particularly in the background of the scene and along the contours of objects that cause the fusion of foreground and background in 3D, would also enable the generation of point clouds with less distortions and artefacts.

Fig. 3: View (5,5) of ChessPieces-MovingCamera when the distance between virtual cameras is set to 8mm. Artifacts appear in the top right corner.

Video Name	Min.	Max.
ChessPieces	-1.52	0.45
Boxer-IrishMan-Gladiator	-1.50	0.39
ChessPieces-MovingCamera	-1.67	0.11

Table 2: Min. and max. values (pixels) for disparity maps

6. REFERENCES

[1] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Mark Horowitz, and Marc Levoy. High performance imaging using large camera arrays. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 765–776. ACM, 2005.

[2] Ren Ng. *Light field photography*. PhD thesis, Stanford University, 2006.

[3] Todor Georgiev, Georgi Chunev, and Andrew Lumsdaine. Superresolution with the focused plenoptic camera. In *Computational Imaging*, 2011.

[4] Marc Levoy and Pat Hanrahan. Light field rendering. In *23rd Annual Conf. on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 31–42. ACM, 1996.

[5] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM, 1996.

[6] Shuo Zhang, Hao Sheng, Chao Li, Jun Zhang, and Zhang Xiong. Robust depth estimation for light field via spinning parallelogram operator. *Journal Computer Vision and Image Understanding*, 145:148–159, 2016.

[7] Ole Johannsen, Antonin Sulc, and Bastian Goldluecke. Occlusion-aware depth estimation using sparse light field coding. In *German Conference on Pattern Recognition (GCPR)*, pages 207–218, 2016.

[8] Sven Wanner and Bastian Goldluecke. Variational light field analysis for disparity estimation and super-resolution. *IEEE Transactions of Pattern analysis and machine intelligence (TPAMI)*, 36(3), 2013.

[9] Hae-Gon Jeon, Jaesik Park, Gyeongmin Choe, Jinsun Park, Yunsu Bok, Yu-Wing Tai, and In So Kweon. Accurate depth map estimation from a lenslet light field camera. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[10] Michael W. Tao, Sunil Hadap, Jitendra Malik, and Ravi Ramamoorthi. Depth from combining defocus and correspondence using light-field cameras. In *International Conference on Computer Vision (ICCV)*, 2013.

[11] Ting-Chun Wang, Alexei Efros, and Ravi Ramamoorthi. Occlusion-aware depth estimation using light-field cameras. In *International Conference on Computer Vision (ICCV)*, 2015.

[12] Raytrix Products. `http://raytrix.de/products/`.

[13] Raytrix Light Field SDK v3.1. `https://raytrix.de/Rx.ApiLF.3.1/namespace_e_par.html`.

[14] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[15] Xiaoran Jiang, Mikael Le Pendu, and Christine Guillemot. Depth estimation with occlusion handling from a sparse set of light field views. In *IEEE International Conference on Image Processing (ICIP)*, 2018.

[16] MeshLab. `http://www.meshlab.net`.

[17] Zhouchen Lin, Risheng Liu, and Zhixun Su. Linearized alternating direction method with adaptive penalty for low rank representation. In *Neural Information Processing Systems (NIPS)*, 2011.

[18] Antonin Chambolle, Vicent Caselles, Matteo Novaga, Daniel Cremers, and Thomas Pock. An introduction to Total Variation for Image Analysis. *hal-00437581*, 2009.

[19] Katrin Honauer, Ole Johannsen, Daniel Kondermann, and Bastian Goldluecke. A Dataset and Evaluation Methodology for Depth Estimation on 4D Light Fields. In Shang-Hong Lai, Vincent Lepetit, Ko Nishino, and Yoichi Sato, editors, *Computer Vision – ACCV 2016*, pages 19–34, Cham, 2017. Springer International Publishing.

[20] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2nd edition, 2003.

[21] Neus Sabater, Guillaume Boisson, Benoit Vandame, Paul Kerbiriou, Frederic Babon, Matthieu Hog, Tristan Langlois, Remy Gendrot, Olivier Bureller, Arno Schubert, and Valerie Allie. Dataset and pipeline for multi-view light-field video. In *CVPR Workshops*, 2017.

A. C++ SNIPPET TO EXTRACT VIEWS

The following is a snippet of the code to extract views. It assumes that the Raytrix API has been correctly initialized and that a rayHandle referring to an instant of the video (i.e a raySequence) is given to the following method named "extractViewsFromCurrentFrame". The namespace is Rx::ApiLF.

```
void extractViewsFromCurrentFrame(
                                unsigned frameIndex,
                                unsigned rayHandle)
{

 // Process image with no white image
 printf("Preprocessing normalized...\n");
 Rx::ApiLF::RxPreProcess();

 // compute depth

 // set the current target view
 // to the view camera space
 Rx::ApiLF::RxSetPar(
     Rx::ApiLF::EPar::Proj_CurrentTargetView,
     unsigned(Rx::Projection::ESpace::ViewCamera));

 // Set Pre Process Parameters
 RxSetPar(EPar::PreProc_DenoiseNLM_Enable, 1U);
 RxSetPar(EPar::PreProc_DenoiseNLM_FilterDia, 4U);
 RxSetPar(EPar::PreProc_DenoiseNLM_NoiseLevel,0.15);
 RxSetPar(EPar::PreProc_DenoiseNLM_BlendFactor,0.2);

 RxSetPar(EPar::PreProc_Sharp1_Enable, 1U);
 RxSetPar(EPar::PreProc_Sharp1_BlurStdDev, 2.5);
 RxSetPar(EPar::PreProc_Sharp1_Factor, 1.8);

 Rx::ApiLF::RxSetPar(Rx::ApiLF::EPar::Depth_Path_NearResLevel, 3U);

 // Set depth parameters for different lens types.
 // Make depth parameters for different lens types equal.
 for (unsigned uLensTypeIdx = 0; uLensTypeIdx < 3; ++uLensTypeIdx)
 {
  RxSetPar(EPar::Depth_Path_LT_PixelStep, uLensTypeIdx, 0.5);
  RxSetPar(EPar::Depth_Path_LT_MinStdDev, uLensTypeIdx, 0.00);
  RxSetPar(EPar::Depth_Path_LT_MinCor, uLensTypeIdx, 0.9);
  RxSetPar(EPar::Depth_Path_LT_MinPolyParAbs, uLensTypeIdx, 0.05);
  RxSetPar(EPar::Depth_Path_LT_MaxPolyCtrDelta, uLensTypeIdx, 0.5);
  RxSetPar(EPar::Depth_Path_LT_PatchDia, uLensTypeIdx, 4U);
  RxSetPar(EPar::Depth_Path_LT_PatchStride, uLensTypeIdx, 1U);
 }

 // Depth Fuse Parameters
 RxSetPar(EPar::Depth_Fuse_MedianRad, 0U);
 RxSetPar(EPar::Depth_Fuse_MeanRad, 0U);
 RxSetPar(EPar::Depth_Fuse_MeanMaxBlackPart, 0.2);

 // Depth Fill Parameters
 RxSetPar(EPar::Depth_Fill_Enabled, 1U);
 RxSetPar(EPar::Depth_Fill_IterCnt, 4U);
 RxSetPar(EPar::Depth_Fill_IterSize, 1U);
 RxSetPar(EPar::Depth_Fill_Complete, 1U);

 // Depth Fill Bilateral filter
 RxSetPar(EPar::Depth_Fill_Bilateral_Enabled, 1U);
 RxSetPar(EPar::Depth_Fill_Bilateral_FilterRad, 5U);
 RxSetPar(EPar::Depth_Fill_Bilateral_Edge, 0.1);
 RxSetPar(EPar::Depth_Fill_Bilateral_Range, 5.0);

 // The result of this operation is stored in the internal image

 // Calculate the raw depth and subsequently
 // the fused depth map in the view camera space
 Rx::ApiLF::RxDepthRay();

 Rx::CRxImage xImage;
 printf("Reading total focus image from CUDA device...\n");

 // save the depthmap image
 Rx::ApiLF::RxDepthMap();

 // Color code the calculated depth to visualize it properly
 Rx::ApiLF::RxDepthColorCode();
```

```
// compute the extracted views
//

 const double initValue = 0.0;
 Rx::CRxArrayDouble adValue(2, initValue);


 RxGetPar(EPar::VirtCamPinholeStd_ViewOffsetMM_g, myadValue);

 // get the horizontal distance in mm between virtual cameras
 int dx = this->m_configParameters.getDY();

 // get the vertical distance in mm between virtual cameras
 int dy = this->m_configParameters.getDY();

 int xinf = (m_configParameters.getNbCamerasPerLine()   1) / 2 * 1 * dx;
 int yinf = (m_configParameters.getNbCamerasPerColumn()   1) / 2 * 1 * dy;

 // set horizontal index to 0
 int v = 0;

 // extract images row by row
 for (int i = xinf; i <= xinf; i = i + dx)
 {
  v++;
  // set the horizontal coordinate of the virtual camera
  myadValue[0] = i;

  // set vertical index to 0
  int u = 0;

  // for a given row extract images for all columns
  for (int j = yinf; j <= yinf; j = j + dy)
  {
   u++;

   // set the vertical coordinate of the virtual camera
   adValue[1] = j;

   RxSetPar(EPar::VirtCamPinholeStd_ViewOffsetMM_g, adValue);

   RxGetPar(EPar::VirtCamPinholeStd_ViewOffsetMM_g, adValue);

   // for the position of the virtual camera get the total focus image
   RxTotalFocus();
   // Get  image from CUDA device
   RxGetImage(EImgID::TotalFocus_ViewCamera, xImage);

   // Save image
   char saveImageName[PATH_FILENAME_LENGTH];
   snprintf(saveImageName, sizeof(saveImageName),
            "%s/sa_%d_%d.png", frame.c_str(), v, u);
   Rx::CRxString sxFile2 = saveImageName;
   std::cout << "Saving_focused_image_as_" << sxFile2.ToCString() << "...\n";

   Rx::FileIO::CImage xImageFile;
   xImageFile.Write(&xImage, sxFile2);


  }
 }
}
```