ANALYSING FOREGROUND SEGMENTATION IN DEEP LEARNING BASED DEPTH ESTIMATION ON FREE-VIEWPOINT VIDEO SYSTEMS

Javier Usón, Julián Cabrera, Daniel Corregidor, Narciso García

Grupo de Tratamiento de Imágenes, Information Processing and Telecommunications Center,
ETSI Telecomunicación, Universidad Politécnica de Madrid, Madrid, Spain.
Email: {jup, julian.cabrera, dcl, narciso}@gti.ssr.upm.es

ABSTRACT

Volumetric video acquisition systems enable realistic virtual experiences such as Free-Viewpoint Video (FVV). Stereo matching is a well known way of obtaining this volumetric information as depth images, calculating the disparity between two stereo color images. On these applications, the background of the scene captured is static and does not change, so foreground information is much more valuable. We propose adding foreground segmentation to help learning based algorithms, such as deep learning models, improve results previously obtained. We utilized the framework Detectron2 to model foreground segmentation by detecting people. Additionally, we built a large stereo dataset focused on FVV systems. Finally, we modified a successful deep learning model from the state-of-the-art, CREStereo, to add foreground segmentation and performed supervised training on it to estimate disparity, obtaining promising results.

***Index Terms*—** Foreground segmentation, deep learning, stereo dataset, depth, multiview video, free viewpoint video.

Fig. 1. A stereo dataset was generated, containing 72000 stereo pairs and the depth image associated to the left frame computed using time of flight cameras. Foreground masks were computed over them using Detectron2.

I. INTRODUCTION

Nowadays, immersive video technologies are gaining importance and innovative experiences are being developed such as Free Viewpoint Video (FVV) technology. It allows the user to navigate freely around a scene as if she were controlling a virtual camera that could be positioned anywhere [1]. These systems rely on a volumetric video acquisition stage that captures the scene from multiple cameras and yields a 3D representation of the scene containing both texture and geometric information. In that sense, the performance of any FVV system depends highly on the quality of the generated geometric data.

This volumetric information can come in the shape of depth images, where each pixel represents the distance from the objects of the scene to the camera. They can be obtained using numerous different means, and these means are divided into two categories: passive and active. Passive methods require simple hardware but it is hard to obtain good quality depth images and often the algorithms to compute them are very complex and computationally heavy. On the other hand, active methods use more complex and expensive hardware to generate good quality depth images with light algorithms.

Being able to obtain good results out of passive methods would greatly cut the costs of the acquisition systems. Stereo matching is a classical computer vision problem aimed to archive this. Given two images, one left and other right, disparity maps can be computed based on matches between them, and they can be transformed into depth if the geometry of the cameras that captured them is known. As in many other difficult tasks, deep learning is being applied to try and solve this problem, obtaining very promising results.

Images captured by static multicamera setups for FVV applications have a large percentage of pixels corresponding to the background that barely change, meaning that the foreground information is much more rich and valuable. This inspired us to apply foreground segmentation to the learning algorithms, giving more importance to these more valuable pixels to obtain better results when estimating disparities.

To help develop these kind of learning algorithms, a large

stereo dataset focused on multiview setups has been developed. This implied calibrating the camera setup, rectifying the stereo image pairs, obtaining depth images using time of flight (active) hardware, and using Detectron2 [2] for foreground segmentation by detecting people.

Finally, the deep learning model CREStereo [3] was modified to make use of the foreground segmentation. Both original and modified versions were trained using the newly generated dataset and compared to validate the improvements obtained.

II. Previous Work

A. FVV Live

This project was inspired and belongs to the research effort for the FVV Live (free viewpoint video live) system [1] [4]. This system makes use of a multicamera setup to render the view from a virtual camera which can be located in any position close to the real cameras, everything while working in real time.

B. Stereo Matching and Segmentation

Depth perception and object detection are two classical computer vision problems that often appear combined. Works such as [5] try to solve them together, and others like [6] take advantage of the segmentation to perform a better stereo matching.

More recent studies apply deep learning techniques to this combined problem, such as [7] adding the results from an object detection net to the stereo matching model, or [8] performing both task in a unique model to speed up the process.

C. Detectron2

Detectron2 [2] is a computer vision framework developed by Meta for object detection and segmentation. For this project, its functionality of detecting people was used to obtain binary masks for foreground segmentation.

D. Stereo Datasets

Generating datasets is a key part of developing learning based algorithms. Stereo datasets are complex to obtain as there is no way of directly obtaining dense stereo matches. For this reason, the mainly used datasets are computer generated using 3D modeling, for example Scene Flow [9]. This is no easy task either, as realistic and varied geometries are difficult to obtain too.

Real life stereo datasets are normally captured using complex camera settings, they make use of active technology to obtain the depth of the scene and then transform it into stereo disparity. Some well known examples are KITTI [10], Middlebury [11] and ETH3D [12]. However, specific datasets for FVV scenarios are not yet available.

E. Capture Hardware

The hardware used to capture the dataset were Microsoft Azure Kinect cameras. They were built with a very high resolution color camera and a IR (infrared) transmitter and receiver to measure depth using time of flight. Modulated IR light is emitted into the scene and then recorded to obtain high quality depth images. The only downside is that problems can appear when capturing the light back, this yields invalid pixels in the depth image.

F. CREStereo

Using deep neural networks to solve the stereo problem is a task that has been greatly developed in the past few years. Well known datasets release benchmarks so people can submit their algorithms and results are ranked.

CREStereo (Cascaded REcurent Stereo matching network) [3] is a recently released deep learning model from the state-of-the-art. This model obtained very good results in well known benchmarks like KITTI and ETH3D, and was at the top of the Middlebury ranking at the time of the development of this project.

This model combines a coarse-to-fine hierarchical approach to better detect small details, with a stacked cascade architecture for high resolution interference, and an adaptive group correlation layer to avoid problems caused by errors in rectification as it is meant for practical situations.

The original model was developed using the MegEngine framework [13], but for this project we used a Pytorch version instead, available in [14].

III. FVV Dataset

We generated a dataset to help develop learning based algorithms for stereo matching making use of the foreground segmentation: the FVV Dataset. The process followed is detailed in this section.

A. Recording the Dataset

To record the dataset, four Azure Kinect cameras were used. They were placed in two rigid stereo rigs to simulate two stereo cameras recording the scene at the same time. These cameras allow hardware triggered capture, so all four are connected to ensure synchronized capture between them.

The dataset consists of recordings of 16 people acting on 6 scenes of different complexities. An example of each one of this sequences can be seen on Fig. 2

- **Reader**: the simplest sequence, The person acting stands in place while reading some magazines and books.
- **Ad**: The person acting stands behind a table and shows a metallic piece and uses some tools on it. While still moving little, the movements are more complex.
- **Talk**: This sequence has the person acting moving around the stage while talking on their phone. It presents the complexity of the movement of the person.

(a) Reader sequence (b) Ad sequence (c) Talk sequence (d) Boxes sequence (e) Lecture sequence (f) Tidy sequence

Fig. 2. Frames belonging to the sequences recorded for the dataset, ordered by complexity. In the first four, only one person was recorded, whereas the last two sequences were recorded in couples. Recorded people were told to act freely to ensure more differences appeared between sequences

- **Boxes**: Some boxes lie around the stage, the person acting moves them from one side to the other. This not only has the complexity of the movement, but also has the person interacting with different objects.
- **Lecture**: Two people are recorded talking about a metallic piece, they move it around and hand it between them while standing in place.
- **Tidy**: The most complex scene. The stage has some pieces of furniture and some small details, two people tidy the stage, moving everything around.

Taking into account the restrictions of our hardware, the final recordings for the dataset were 30 seconds long, recorded at 15 fps and the resolution was 1280x720.

B. Capture Software

Making use of the Azure Kinect SDK (Software Development Kit) [15] and the OpenCV library [16], we developed a software to capture and post-processing the sequences.

This software is able to record sequences using the four cameras (two stereo pairs) in a synchronized fashion. In addition, it uses OpenCV stereo camera calibration to obtain the extrinsic parameters that relate the global position and orientation of each pair of cameras. Their stereo calibration file is stored together with the two videos as individual frames as PNGs (Portable Network Graphics). The two stereo color images are stored as RGB (Red Green Blue) with 8 bit per channel. It also performs post-processing on the images before storing them.

Using the camera calibration, we can once again use OpenCV to rectify the stereo pair of images. This is the process where both images are transformed so every two stereo matching pixels are located on the same horizontal line, making the stereo matching problem much easier.

Depth is stored as a one channel image with 16 bits per pixel, where the pixels represent the absolute distance between the camera and the objects from the scene measured in millimeters. From the two Azure Kinect cameras we obtain two depth images from the point of view of the IR sensor. These do not match any of the color frames, but the Azure Kinect SDK lets us warp these images to their respective color sensors. For the depth image we stored the one corresponding to the left color frame.

For the depth image we stored the one corresponding to the left color frame. However, time of flight techniques are not perfect, and invalid values appear when one pixel is not capable of correctly measure the depth. Some of the invalid pixels that appear on the left depth image can be filled making use of the right depth image by warping the right image to the left using the stereo calibration. The resulting depth image matches the left color image and has some of its invalid pixels filled using the information from the right image.

C. Foreground Segmentation

To model this foreground segmentation we apply object detection and detect people on the color images. We use the framework Detectron2 for this task. It generates a binary mask as shown on the bottom left image from Fig. 1.

D. FVV Dataset Structure

FVV Dataset consists of a total of 72000 stereo images with their respective foreground masks and depth image. These images are organized by sequences, each one of them is a directory and inside we can find the following structure:

- **Original**: not rectified frames.
  - **Color_L**: left RGB frame.
  - **Color_L_mask**: binary mask for the left frame.
  - **Color_R**: right RGB frame.
  - **Color_R_mask**: binary mask for the right frame.
  - **Depth**: original left depth image.
  - **Complete_Depth**: left depth image filled with information from the right camera.
- **Rectified**: rectified frames.
  - **Color_L**: rectified left RGB frame.
  - **Color_L_mask**: binary mask for the rectified left frame.
  - **Color_R**: rectified right RGB frame.
  - **Color_R_mask**: binary mask for the rectified right frame.
  - **Depth**: rectified left depth image.

– **Complete_Depth**: rectified left depth image filled with information from the right camera.

IV. Experiments

A. Approach

We implemented a disparity estimation system modifying a deep learning model from the state-of-the-art, CREStereo [3], to make use of the foreground segmentation in the shape of binary masks. Then, we trained it using the newly generated dataset and compared the results to the ones obtained by the original model.

For these experiments, first we develop a data-loader module that allows us to load the dataset dividing it by sequences. It also takes care of transforming the stored depth images (Z) into disparity maps ($x_l - x_r$). To do it, we stored in the calibration file the baseline (B) and focal distance (f) obtained in the calibration and rectification process. We use them to compute the disparity map using (1).

$$(x_l - x_r) = \frac{Bf}{Z} \tag{1}$$

Using this data-loader we divide the train and test sets by the people acting on the sequences. We use 12 people for the train set and 4 for test, which means 75% for training (54000 stereo pairs) and 25% for testing (18000 stereo pairs).

To evaluate the trained models we used the L_1 loss (2), the L_2 loss (3) and the Bad2.0 metric (4). We calculate the same metrics but only inside of the foreground mask. Non-valid pixels in the ground truth are never taken into account when calculating these metrics.

$$L_1 = \frac{1}{N}\sum_N |d_g - d_p| \tag{2}$$

$$L_2 = \frac{1}{N}\sqrt{\sum_N (d_g - d_p)^2} \tag{3}$$

$$Bad2.0 = \frac{1}{N}\sum_N (|d_g - d_p| > 2) \cdot 100 \tag{4}$$

B. Training the Original Model

We follow a similar approach as the developers of the model to train it using our new dataset. We trained it on a NVIDIA GeForce RTX 3080 GPU, and its memory limitations forced us to use a batch size of 1 and to reduce the size of the images to 910x512, which we do by random cropping the original images. The process consists of 300 epochs, each one using a random group of 500 images.

We use Adam as the optimizer with a variable learning rate. It starts with a warm up phase going from $8 \cdot 10^{-5}$ to $4 \cdot 10^{-4}$ linearly in 6 epochs. Then it remains at that value until epoch 180. Finally, it falls to $2 \cdot 10^{-5}$ linearly.

TABLE I
Results of the tests on CREStereo

Experiment	L_1	L_2	Bad 2.0	L_1 D	L_2 D	Bad 2.0 D	Avg. time (s)
CRES Original	2.18	5.14	22.78	2.58	3.42	50.21	1.40
CRES Detectron	1.53	2.29	23.08	2.25	2.96	45.41	1.64

C. Adding Foreground Segmentation

To improve the results of the model, we propose adding the foreground segmentation masks to the training. We add the masks as a fourth channel on the input image and modify the training loss function to give a weight of 0.75 to the pixels inside of the mask and 0.25 to the ones outside of it.

We train the modified model using the same parameters as with the original one. Now the memory consumption is bigger, so we reduce the size of the images to 704x400.

V. Results

Once the models were trained, we test them using the mentioned metrics. The results are presented on Table I. There, CRES original is the unmodified model trained with our dataset, CRES stereo is the model that uses the foreground segmentation and the metrics marked with a D are the ones computed only on the foreground mask.

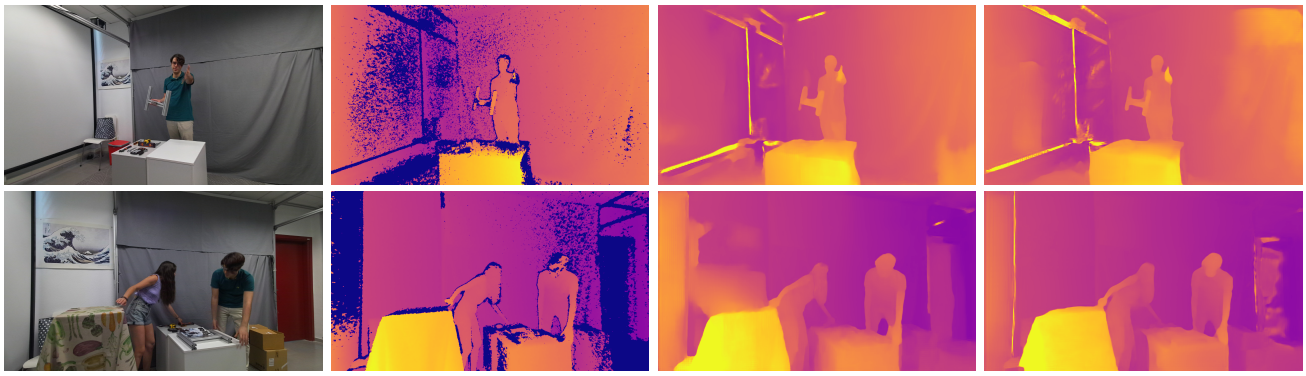
The results confirm that we were able to generate a very good disparity estimation system using our dataset. Adding the foreground segmentation to the training not only made the estimation better on the foreground, but helped getting better results on the background as well. Fig. 3 shows visual results of the estimation, the images present good quality and a nicely defined foreground. Adding Detectron2 helps getting better foreground definition, as we can see on the metallic piece on the hand of the actor on the top row, and the table on the bottom row.

VI. Conclusions

We have proposed the use of foreground segmentation when training deep learning algorithms for stereo disparity and depth estimation in a FVV scenario. We have tested it on a state-of-the-art model and archived a very promising improvement in the performance of the system. We have also generated a large dataset to help create disparity estimation systems focused on multicamera DIBR applications. The dataset can easily be extended and the original media can be processed in different ways to obtain different results.

Acknowledgment

This work has been supported by the European Union's H2020 research and innovation programme under project "5G-RECORDS: 5G key technology enableRs for Emerging media COntent pRoDuction Services" (grant agreement 957102) and by project "SARAOS" (PID2020-115132RB)



(a) Left Image (b) Ground truth (c) CREStereo (d) CRES using Detectron

Fig. 3. Disparity maps resulting from the estimation using the models trained with the newly generated stereo dataset. The images on the top row belong to the simplest sequence recorded, and the ones on the bottom row come from the most complex sequence.

funded by MCIN/AEI/10.13039/501100011033 of the Spanish Government.

We thank the volunteers who were willing to let us make their image public to help develop and publish this dataset.

VII. References

[1] Pablo Carballeira, Carlos Carmona, César Díaz, Daniel Berjón, Daniel Corregidor, Julián Cabrera, Francisco Morán, Carmen Doblado, Sergio Arnaldo, María del Mar Martín, and Narciso García, "Fvv live: A real-time free-viewpoint video system with consumer electronics hardware," *IEEE Transactions on Multimedia*, vol. 24, pp. 2378–2391, 2022.

[2] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.

[3] Jiankun Li, Peisen Wang, Pengfei Xiong, Tao Cai, Ziwei Yan, Lei Yang, Jiangyu Liu, Haoqiang Fan, and Shuaicheng Liu, "Practical stereo matching via cascaded recurrent network with adaptive correlation," 2022.

[4] Pablo Pérez, Daniel Corregidor, Emilio Garrido, Ignacio Benito, Ester González-Sosa, Julián Cabrera, Daniel Berjón, César Díaz, Francisco Morán, Narciso García, Life Member IEEE, Josué Igual, , and Jaime Ruiz, "Live free-viewpoint video in immersive media production over 5g networks," *IEEE Transactions on Broadcasting*, vol. 68, no. 2, pp. 439–450, 2022.

[5] Michael Bleyer, Carsten Rother, Pushmeet Kohli, Daniel Scharstein, and Sudipta Sinha, "Object stereo — joint stereo matching and object segmentation," in *CVPR 2011*, 2011, pp. 3081–3088.

[6] M. Gerrits and P. Bekaert, "Local stereo matching with segmentation-based outlier rejection," in *The 3rd Canadian Conference on Computer and Robot Vision (CRV'06)*, 2006, pp. 66–66.

[7] Shengyou Hua, Zhiyong Sun, Bo Song, Pengpeng Liang, and Erkang Cheng, "Pseudo segmentation for semantic information-aware stereo matching," *IEEE Signal Processing Letters*, vol. 29, pp. 837–841, 2022.

[8] Pier Luigi Dovesi, Matteo Poggi, Lorenzo Andraghetti, Miquel Martí, Hedvig Kjellström, Alessandro Pieropan, and Stefano Mattoccia, "Real-time semantic stereo matching," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 10780–10787.

[9] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, arXiv:1512.02134.

[10] Moritz Menze and Andreas Geiger, "Object scene flow for autonomous vehicles," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[11] Daniel Scharstein, Heiko Hirschmüller, York Kitajima, Greg Krathwohl, Nera Neši´ Neši´c, Xi Wang, and Porter Westling, "High-resolution stereo datasets with subpixel-accurate ground truth," *German Conference on Pattern Recognition (GCPR 2014)*, 2014.

[12] Thomas Schöps, Johannes L. Schönberger, Silvano Galliani, Torsten Sattler, Konrad Schindler, Marc Pollefeys, and Andreas Geiger, "A multi-view stereo benchmark with high-resolution images and multi-camera videos," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[13] "Megengine: A fast, scalable and easy-to-use deep learning framework," https://github.com/MegEngine/MegEngine, 2020.

[14] Ibai Gorordo, "Crestereo-pytorch," https://github.com/ibaiGorordo/CREStereo-Pytorch, 2022.

[15] "Azure kinect sensor sdk," https://microsoft.github.io/Azure\-Kinect\-Sensor-SDK/master/index.html, 2022.

[16] "Opencv calib3d," https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html, 2022.