**3GPP TSG SA WG4#129-e S4-241488**

**Online, 19th - 23rd August 2024**

Agenda item: 9.9

Source: China Mobile Com. Corporation

Title: [FS\_Beyond2D] Available Datasets, tools, softwares for Stereoscopic Video Test Sequences

Document for Discussion and Agreement

# Introduction

During the last Telco meeting, a scenario on “UE-to-UE Beyond 2D Video Streaming” had been discussed and agreed (S4aV240022 [1]) to be documented in Permanent Document of FS\_Beyond2D (S4-241321 [2]). This proposal provides some datasets, generation softwares, and capturing tools for stereoscopic video test sequences.

# Proposed

##### 2.1 Candidate Sources Sequences

This section offers public datasets, generation software, and capturing tools for producing stereoscopic source sequences.

Editor’s Note: The selection of test sequences prioritizes quality considerations.

###### 2.1.1 Public Datasets

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dataset** | **Generated**  **/Natural** | **Stereo** | **Depth** | **Resolution** | **Download Link** |
| InStereo2K [3]A | Natural | Y | Y | 1080\*860 | <https://github.com/YuhuaXu/StereoDataset> |
| Middlebury 2021 Mobile stereo datasets [5] | Natural | Y | Y | 1920\*1080 | <https://vision.middlebury.edu/stereo/data/scenes2021/> |
| Middlebury 2014 stereo datasets [5] | Natural | Y | Y | 2964\*1988 | <https://vision.middlebury.edu/stereo/data/scenes2014/> |

Copyrights:

**InStereo2K:** This dataset is made freely available to academic and non-academic entities for non-commercial purposes such as academic research, teaching, scientific publications, or personal experimentation.[The author had granted the permission to use this dataset]

**Middlebury:**We grant permission to use and publish all images and numerical results on this website. If you report performance results, we request that you cite our paper. Instructions on how to cite our datasets are listed on the datasets page. If you want to cite this website, please use the URL "vision.middlebury.edu/stereo/".

###### 2.1.2 AI Based 2D-to-Stereo3D Conversion

The AI-based conversion of existing 2D images to stereo3D is proving commercially viable and fulfills the growing need for high quality stereoscopic images. This approach is particularly effective when creating content for the new generation of autostereoscopic displays that require multiple stereo images.

1. **Commercial Services**

- Owl3D - AI-powered 2D to 3D video conversion software: <https://www.owl3d.com/>

- Leia Inc. Immersity AI | Convert Image and Video to 3D: [https://www.immersity.ai/]( https:/www.immersity.ai/)

-Meta Turning any 2D photo into 3D using convolutional neural nets <https://ai.meta.com/blog/powered-by-ai-turning-any-2d-photo-into-3d-using-convolutional-neural-nets/>

1. **Algorithm**

Various open-source algorithms and platforms use deep neural networks to perform real-time end-to-end conversion of 2D videos and images to stereoscopic 3D format. The performance of these conversions may vary based on the algorithm's efficiency and the hardware capabilities.

The following are capabilities and requirements of the Deep3D [6] algorithm [(https://github.com/HypoX64/Deep3D]((https:/github.com/HypoX64/Deep3D)):

### Prerequisites:

* Linux, Mac OS, Windows
* Python 3.7+
* [ffmpeg 3.4.6+](http://ffmpeg.org/)
* [Pytorch 1.7.1](https://pytorch.org/)
* CPU or NVIDIA GPU

Inference speed:

| **Plan** | **360p (FPS)** | **720p (FPS)** | **1080p (FPS)** | **4k (FPS)** |
| --- | --- | --- | --- | --- |
| GPU (2080ti) | 84 | 87 | 77 | 26 |
| CPU (Xeon Platinum 8260) | 27.7 | 14.1 | 7.2 | 2.0 |

1. **Related tools**

As 2D-to-Stereo3D conversion algorithms usually take RGB video format, the Python codes below can be used to convert between YUV and RGB formats.

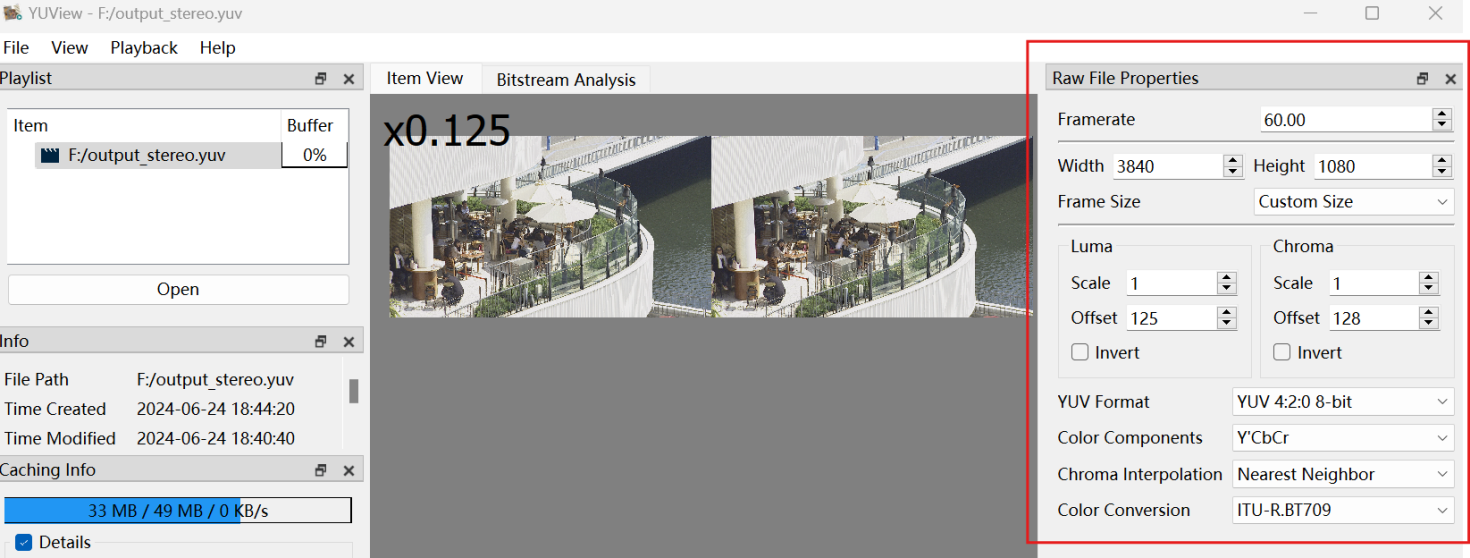
A Python code for converting YUV to RGB:

1. import cv2
2. import numpy as np
4. def read\_yuv\_frame(file, width, height):
5. # calculate size of YUV frame
6. frame\_size = width \* height \* 3 // 2
7. # Read YUV frames
8. yuv = np.fromfile(file, dtype=np.uint8, count=frame\_size)
9. **if** yuv.size != frame\_size:
10. **return** None
11. # reconstruct YUV data to YUV420 format
12. y = yuv[0:width\*height].reshape((height, width))
13. u = yuv[width\*height:width\*height + (width//2)\*(height//2)].reshape((height//2, width//2))
14. v = yuv[width\*height + (width//2)\*(height//2):].reshape((height//2, width//2))
15. **return** y, u, v
17. def yuv\_to\_rgb(y, u, v):
18. # convert YUV420 to YUV444
19. u = cv2.resize(u, (y.shape[1], y.shape[0]), interpolation=cv2.INTER\_LINEAR)
20. v = cv2.resize(v, (y.shape[1], y.shape[0]), interpolation=cv2.INTER\_LINEAR)
21. yuv = np.stack((y, u, v), axis=-1)
22. # convert yuv to rgb
23. rgb = cv2.cvtColor(yuv, cv2.COLOR\_YUV2RGB)
24. **return** rgb
26. def main():
27. # read YUV file path
28. yuv\_file\_path = 'video.yuv'
29. # set width and height
30. width, height = 1920, 1080
32. with open(yuv\_file\_path, 'rb') as file:
33. **while** True:
34. # red yuv frame
35. yuv\_frame = read\_yuv\_frame(file, width, height)
36. **if** yuv\_frame is None:
37. **break**
38. y, u, v = yuv\_frame
39. # convert yuv to rgb
40. rgb\_frame = yuv\_to\_rgb(y, u, v)
41. # display rgb frame
42. cv2.imshow('RGB Frame', rgb\_frame)
43. **if** cv2.waitKey(30) & 0xFF == ord('q'):
44. **break**
46. cv2.destroyAllWindows()
48. **if** \_\_name\_\_ == "\_\_main\_\_":
49. main()

A Python code for converting RGB to YUV:

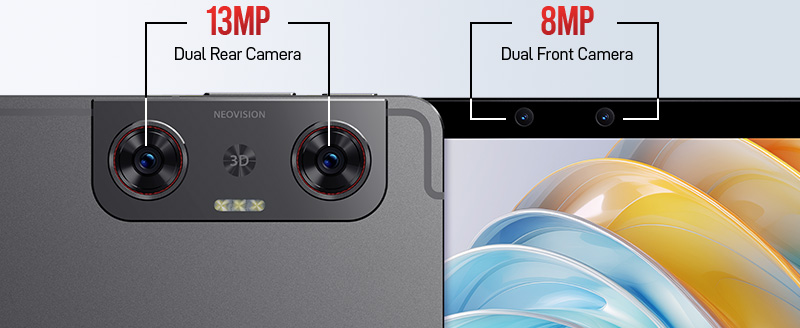
1. import cv2
2. import numpy as np
3. import time
5. def rgb\_to\_yuv420(frame):
6. # Using OpenCV for converting RGB to YUV format
7. yuv = cv2.cvtColor(frame, cv2.COLOR\_RGB2YUV\_I420)
8. **return** yuv
10. def main():
11. # read RGB video file path
12. rgb\_video\_path = 'video.mp4'
13. # set output YUV file path
14. yuv\_file\_path = 'output.yuv'
15. # open video file
16. cap = cv2.VideoCapture(rgb\_video\_path)
18. # check if video file is open
19. **if** not cap.isOpened():
20. print("Error: Could not open video.")
21. **return**
23. # get width and height of video file
24. width = **int**(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))
25. height = **int**(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))
26. print(f'Video dimensions: {width}x{height}')
28. # set framerate
29. fps = 60
30. frame\_duration = 1.0 / fps
32. with open(yuv\_file\_path, 'wb') as yuv\_file:
33. **while** cap.isOpened():
34. start\_time = time.time()
35. # read RGB frame
36. ret, frame = cap.read()
37. **if** not ret:
38. **break**
40. # convert RGB to YUV
41. yuv\_frame = rgb\_to\_yuv420(frame)
43. # write YUV frame to yuv file
44. yuv\_file.write(yuv\_frame)
46. elapsed\_time = time.time() - start\_time
47. wait\_time = max(1, **int**((frame\_duration - elapsed\_time) \* 1000))
48. **if** cv2.waitKey(wait\_time) & 0xFF == ord('q'):
49. **break**
51. cap.release()
52. cv2.destroyAllWindows()
54. **if** \_\_name\_\_ == "\_\_main\_\_":
55. main()
56. **Example**

The following is the test sequence that auto-converting a 2D video frame (BQTerrance 1920x1080, 60fps) to a stereoscopic 3D video (1920x1080 for each eye, 60fps) using CMCC’s 2D-to-Stereo3D conversion algorithm.



###### 2.1.3 Stereo 3D Video Capturing

Another option is to use a dual-lens camera to directly capture stereo 3D video. There are many mobile devices on the market with this capability. For example, SpatialLabs Eyes provided by Acer is a stereoscopic camera capable of capturing up to 8-MP (4K) per eye at 30 fps or 2K per eye at 60 fps. Or the ZTE Nubia Pad 3D II can capture stereo 3D video with the following specifications:



The main camera setup is the dual-camera systems includes two identical 13 MP lenses. These cameras capture slightly different perspectives of the same scene, mimicking the way human eyes perceive depth. The AI then processes these images to produced a coherent 3D representation.

|  |  |
| --- | --- |
| **Rear Camera** | **Specification** |
| Number of Cameras | 2 (Dual) |
| Resolution | 13 MP (wide); 13 MP (wide) |
| Autofocus | AF, AF |
| Video Recording | 1200 @ 30 fps |
| Others | LED Flash, panorama, HDR, Stereoscopic AI-powered 3D capture |

# The selfie camera setup features two lenses positioned near the center of the top bezel when the tablet is oriented horizontally (with the longer side on the top). Here are the relevant specs:

|  |  |
| --- | --- |
| **Front Camera** | **Specification** |
| Number of Cameras | 2 (Dual) |
| Resolution | 8 MP (ultra wide); 8 MP (ultra wide) |
| Aperture | f/2.2, f/2.2 |
| Field of View | 105°, 105° |
| Video Recording | 1200 @ 30 fps |
| Others | Stereoscopic AI-powered 3D capture |

The captured videos (in mp4 file) need further processing (e.g., reading the right/left views and concatenate them into one video frame) to generate the proper test sequences.

A Python code for doing these tasks:

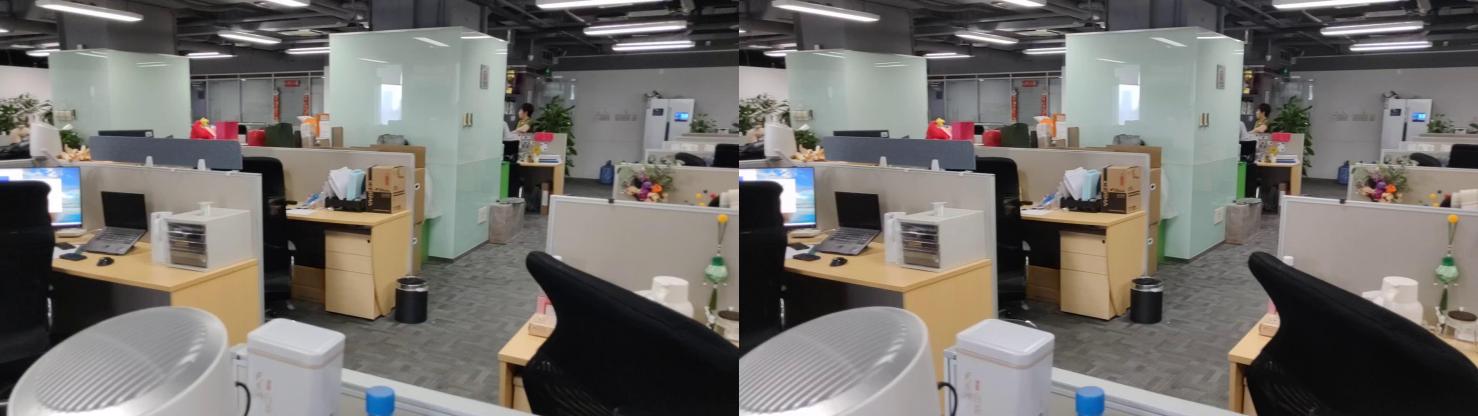
1. import numpy as np
2. import json
3. import subprocess
5. def videoInfo(filename):
6. proc = subprocess.run([
7. \*"ffprobe -v quiet -print\_format json -show\_format -show\_streams".split(),
8. filename
9. ], capture\_output=True)
10. proc.check\_returncode()
11. **return** json.loads(proc.stdout)
13. def readVideo(filename):
14. cmd = ["ffmpeg", "-i", filename]
15. streams = 0
16. **for** stream in videoInfo(filename)["streams"]:
17. index = stream["index"]
18. **if** stream["codec\_type"] == "video":
19. width = stream["width"]
20. height = stream["height"]
21. cmd += "-map", f"0:{index}"
22. streams = streams + 1
23. cmd += "-f", "rawvideo", "-pix\_fmt", "rgb24", "-"
24. shape = np.array([streams, height, width, 3])
25. with subprocess.Popen(cmd, stdout=subprocess.PIPE) as proc:
26. **while** True:
27. # One byte per each element
28. data = proc.stdout.read(shape.prod())
29. **if** not data:
30. **return**
31. yield np.frombuffer(data, dtype=np.uint8).reshape(shape)
33. **if** \_\_name\_\_ == '\_\_main\_\_':
34. import matplotlib.pyplot as plt
35. from PIL import Image
36. idx = 1
37. **for** left, right in readVideo("./StereoCaptured.mp4"):
38. # concatenate left and right
39. img = np.concatenate((left[0:1080,:], right[0:1080,:]), axis=1)
40. img = Image.fromarray(img)
41. img.save(f"imgs/{idx}.jpg")
42. idx += 1

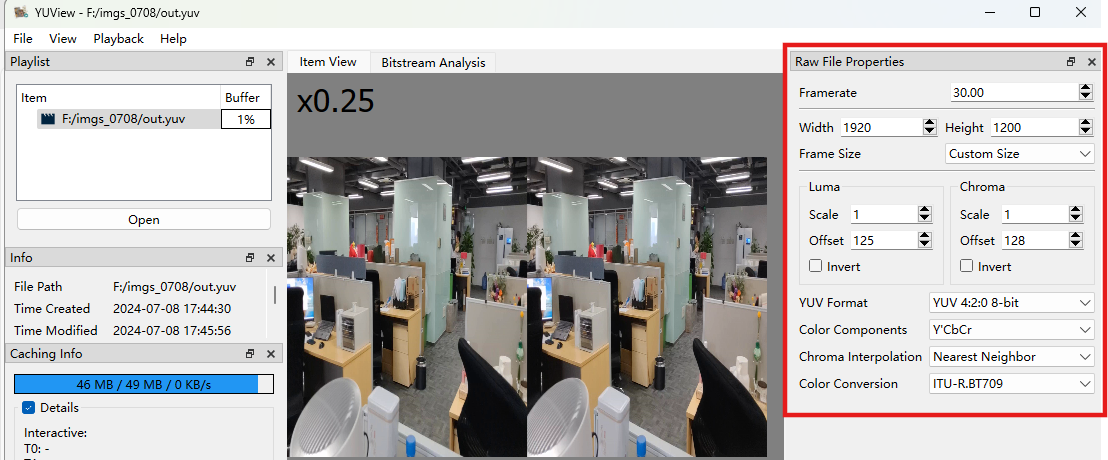
A FFMPEG command for storage them into .yuv

1. ffmpeg -start\_number 1 -r 30 -i %d.jpg -pix\_fmt yuv420p -s 1920x1200 out.yuv

**Example**

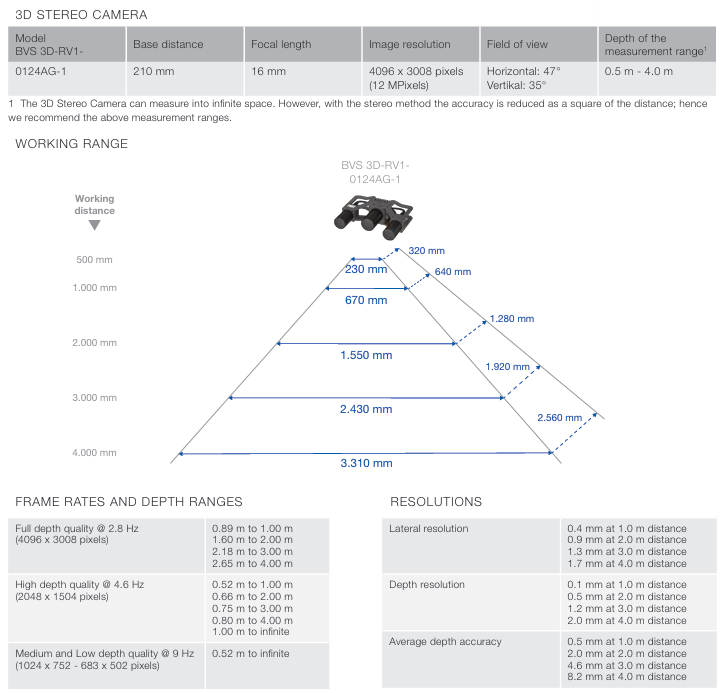
The following is the test sequence captured with ZTE Nubia Pad 3D II main camera (1920\*1200, 30fps) and post-processed by the above tools.





Others:

High resolution 3D stereo camera (specification: https://assets.balluff.com/WebBinary1/BVS\_3D\_RV1\_3D\_Stereo\_Camera\_EN.pdf)



# Proposal

1. Document clause 2 of this document to the Permanent Document of FS\_Beyond2D under the "UE-to-UE Beyond 2D Video Streaming" scenario as candidate source sequences options.
2. For test sequences, considering the following options:

- Public dataset with appropriate copyright authorization

- Captured Sequences

- Generated Sequences

# Reference

[1] S4aV240022, “[FS\_Beyond2D] Scenario UE-to-UE Beyond 2D Video Streaming”

[2] S4-241321 “ [FS\_Beyond2D] Permanent Document v0.0.2”

[3] Bao, W., Wang, W., Xu, Y. et al. InStereo2K: a large real dataset for stereo matching in indoor scenes. Sci. China Inf. Sci. 63, 212101 (2020). https://doi.org/10.1007/s11432-019-2803-x

[4] T. Schöps, J. L. Schönberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys, A. Geiger, "A Multi-View Stereo Benchmark with High-Resolution Images and Multi-Camera Videos", Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[5] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nesic, X. Wang, and P. Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In German Conference on Pattern Recognition (GCPR 2014), Münster, Germany, September 2014.

[6] Xie, J., Girshick, R., Farhadi, A. (2016). Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep Convolutional Neural Networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science(), vol 9908. Springer, Cham. https://doi.org/10.1007/978-3-319-46493-0\_51