EVALUATING THE QUALITY OF MULTI-CAMERA VIDEO CAPTURE AND VIEW-POINT INTERPOLATION FOR 6DOF AR/VR APPLICATIONS

Chris Varekamp

Philips Group Innovation, Research, The Netherlands

ABSTRACT

Producing live 6DoF video requires video capture with multiple cameras, real-time depth estimation, compression, streaming and playback. All of these components are under development and a ready-made solution is hard to find. To make the right choices during development there is a clear need to be able to predict in advance the effect that system parameters (e.g. baseline) and depth estimation algorithms have on image quality.

In this paper, I present a quality evaluation approach that uses ray-traced images of artificial scenes to simulate the acquisition for a given camera capture configuration. The images are passed to real-time depth estimation and view-synthesis software. Views are then synthesized for a pre-set viewing zone and the resulting images are compared with the ray-traced images. Modelling errors are isolated from depth estimation errors by comparing both the images synthesized from ground truth depth and images synthesized from estimated depth with the ray-traced images.

INTRODUCTION

With an increasing number of display devices supporting positional tracking and 3D interaction, the relevance of multi-camera capturing and 6DoF processing increases. Applications include live concerts, live sports and telepresence. The freedom of selecting one's own viewpoint enriches these applications by increasing the feeling of presence over regular video. Further into the future more immersive scenarios can be conceived where an observer may navigate and interact with a live captured scene. For broadcast applications we need real-time depth estimation on the production side and real-time view synthesis at the client device. Both depth estimation and view synthesis introduce errors and these errors depend on the implementation details of algorithms. Furthermore, the optimal camera configuration depends on the intended application and the 3D structure of the scene being captured. In the next sections, I introduce a ray-tracing approach to quality evaluation inside a target viewing zone. The approach is evaluated using our real-time multi-camera setup for live broadcast.

SIMULATION APPROACH

Blender [1] is a graphics rendering engine that is commonly used for film creation and game development. The Python interface for version 2.79 was used to create ray traced images for cameras located in a regular grid of 15×15 anchors with a spacing of 3cm. The resulting viewing zone allows an observer to move his/her head back-to-front and left-to-right (see Figure 1).

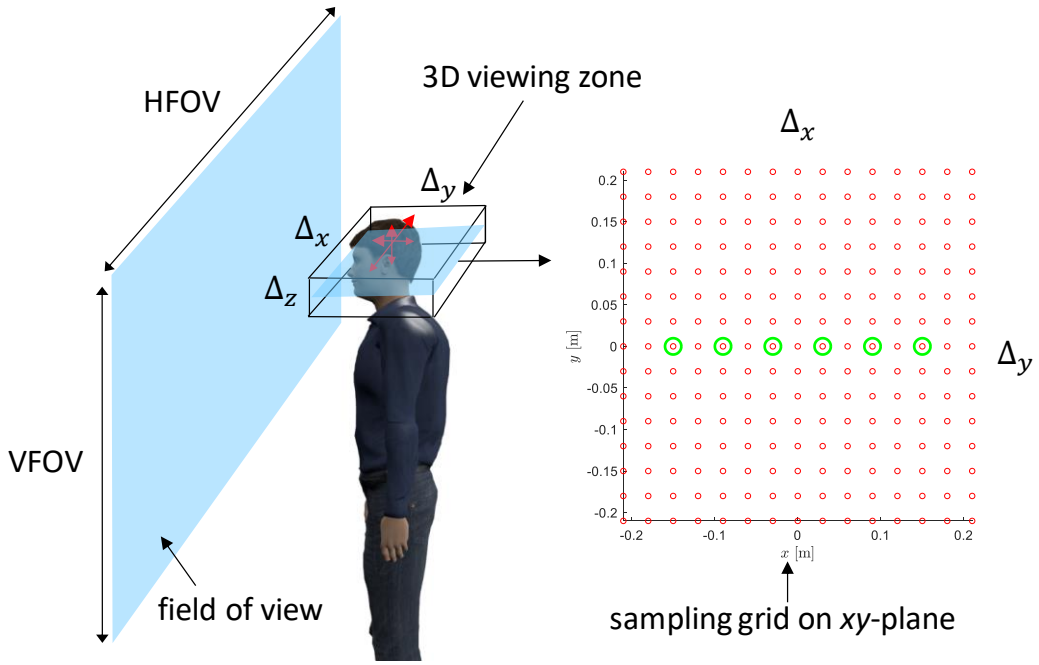


Figure 1 – Viewing zone for a standing person allowing limited head motion parallax. The quality of view synthesis from a given set of reference camera views (green circles) is evaluated on a uniform grid (red circles).

Python was used to automate the Blender raytracing of the 15×15 images. A sample spacing of 3cm was used in both *x*- and *y*-direction. One of the key parameters to pre-investigate for the design of a capture rig is the camera spacing (baseline). Using ray traced images allows us to find an optimal baseline for a given minimum quality level within the intended viewing zone. As representative scenes we investigated the capture of a person scene (*Human*) that we constructed using the MakeHuman software in [2], and a car scene (*Car*) for which we used one of the Blender demo files [3].

To compare system parameters with a simple measure we use Peak signal-to-noise ratio:

$$PSNR \equiv 10 \cdot \log_{10}\left(\frac{255^2}{MSE}\right),$$

where MSE is the mean squared error over RGB colour channels. While $PSNR$ may not be the best metric to evaluate absolute video quality, it is especially useful for comparing baseline within a single dataset. Next to using PSNR we visually compare synthesised images with the ground truth ray traced images.

REAL-TIME CAPTURE, DEPTH ESTIMATION, STREAMING AND PLAYBACK

Video capture, depth estimation, multi-view packing and compression

Figure 2(a) gives a system overview showing the algorithm blocks from capture to rendering on the client device. For the live broadcast case, depth estimation and multi-view registration are very similar to what we described for a static scene in reference [4]. Again, we used depth estimation and error classification as described in [5] and [6] but now implemented on a GPU to achieve real-time performance at 30Hz. See [7] for a review of real-time depth estimation algorithms. A temporal bilateral filter makes sure that the depth maps vary smoothly as a function of time such that depth errors are at least temporally not disturbing.

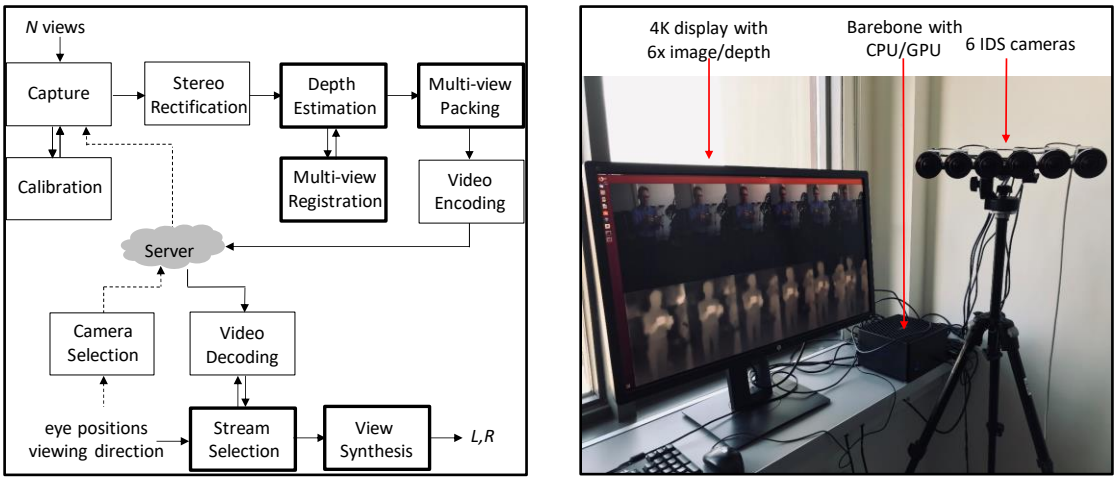


Figure 2 – (a) Components of our real-time system; (b) Photograph of our current system for live capture and conversion.

Our current system displayed in figure 2b consists of a 6-camera rig, a barebone computer (Magnus ZBOX-EN1080K) and a 4K display for monitoring estimated depth maps. The system processes 6-camera feeds of 640×1080 resolution, calculates 6 depth maps, packs 6 images and 6 depth maps together in a single 4K video frame and encodes this, all in real-time at 30 fps. Such a system thus forms a scalable low-cost (consumer hardware) solution for live-streaming: Depending on the target resolution, two, four or six cameras may be attached to a single PC and the output of each PC can stream to a common server. Frame synchronization of the multiple videos is dealt with at the capture side. The 4K output of each PC is encoded using the encoder chip that is present on the graphics card. The system can output normal H.264 or HEVC video or can directly produce HLS/MPEG-DASH video fragments to allow adaptive streaming.

Stream selection, view synthesis, view blending and display

At the client side, views are received as packed video and decoded using the platform specific hardware decoder. After decoding follows unpacking where the needed reference

views and depth maps are extracted from the packed frame. The depth maps are converted to a mesh using a vertex shader.

Stream selection (Figure 3) runs at the client device. We assume that the client has model matrices M_i of the reference views i available as metadata. Stream selection selects the two nearest reference viewpoints using the 4×4 view matrices V_{left} and V_{right} for each eye. The nearest viewpoint is calculated using:

$$i_{\text{nearest}} = \arg\min_i(|M_i\mathbf{p} - V\mathbf{p}|),$$

where M_i is the model matrix for view i, homogeneous coordinate $\mathbf{p} = (0,0,0,1)^{\text{t}}$ and V is the view matrix of either the left or the right eye.

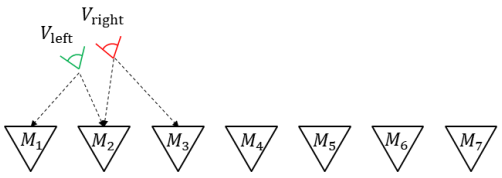


Figure 3 – Selection of the two nearest reference viewpoints for each eye.

At initialization, we create a fixed size regular triangular mesh. Via sampling of the depth map, the *vertex shader* converts each vertex of the mesh directly to a clip-space homogenous output position:

$$\mathbf{p} = PV_{\text{eye}}M_iQ_i\begin{bmatrix} u \\ v \\ D_i(u,v) \\ 1 \end{bmatrix},$$

where $D_i(u,v)$ is the disparity derived from the depth map at input texture coordinates (u,v), Q_i is the disparity to depth matrix and $PV_{\text{eye}}M_i$ is the product of model, view and projection matrix for a given eye. For the experiments in this paper we use a simple fragment shader, but it can in principle be used to do more advanced occlusion handling and/or blending for improved image quality. Both the nearest and the second nearest reference view are blended together to predict the final image. This allows in principle for a scalable solution to 6DoF video where only a limited sub-set of potentially very many views are streamed to the user while he/she is moving. In this paper, blending only depends on the proximity of reference views:

$$\mathbf{I}_{\text{eye}} = \frac{|x_2|}{|x_1|+|x_2|}\mathbf{I}_1 + \frac{|x_1|}{|x_1|+|x_2|}\mathbf{I}_2,$$

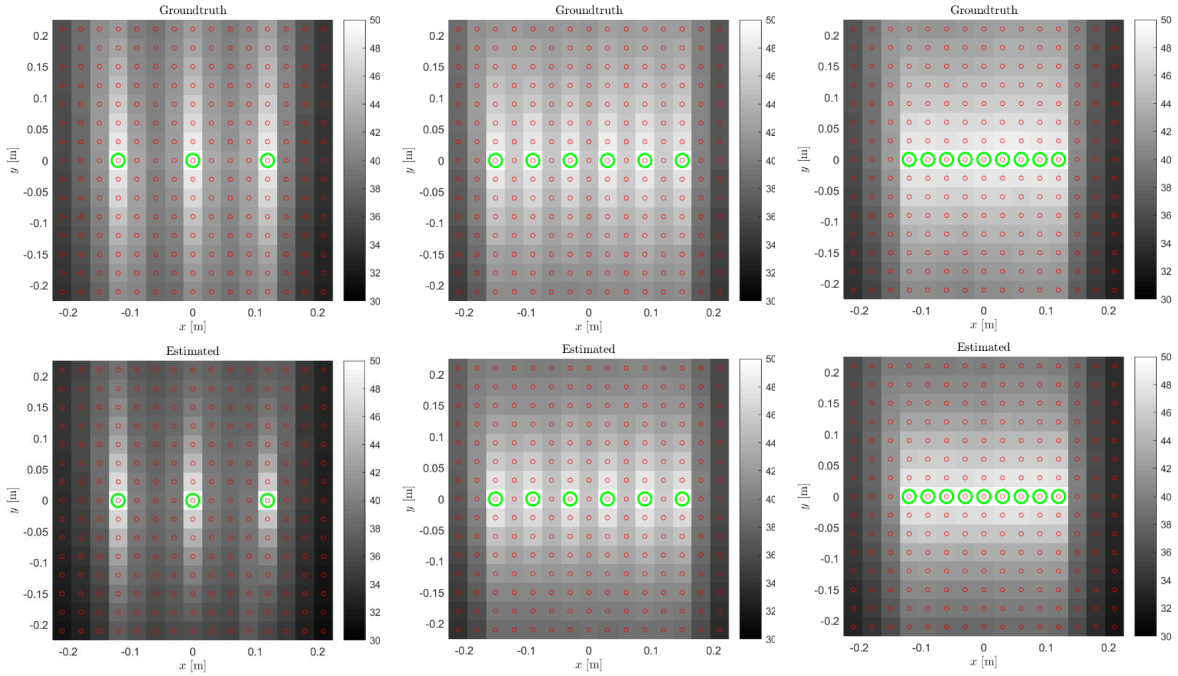
where x_1 and x_2 are the distances along the *x*-axis to the nearest and second nearest reference views. This simple blending equation represents a trade-off between perceptually

smooth transitions between reference views and slightly less accuracy of view synthesis in occlusion regions.

RESULTS AND DISCUSSION

Quantitative evaluation

Figures 4 and 5 show PSNR variations in the viewing zone for three different camera baselines (12cm, 6cm and 3cm). The top row of each figure was produced with ground truth depth maps while the bottom row was produced with estimated depth maps. Ground truth depth and estimated depth result in a similar pattern: the smaller the baseline the higher the PSNR in the viewing zone. Table 1 summarizes results for the two scenes where the minimum PSNR over a 24×24cm region is reported. It can be seen that PSNR values are systematically lower for the *Car* scene when compared to the *Human* scene. This is due to the transparent objects (windows) in the car for which the model of having a single depth value per pixel is clearly too simple. For the *Car* scene, the depth estimator can fail for shiny and/or transparent object parts.



$B = 0.12\text{m}$ $B = 0.06\text{m}$ $B = 0.03\text{m}$

Figure 4 – PSNR [dB] for *Human* inside the viewing zone on a scale from 30-50 dB for varying camera baseline using ground truth disparity (top row) compared with estimated disparity (bottom row). Green circles are reference cameras.

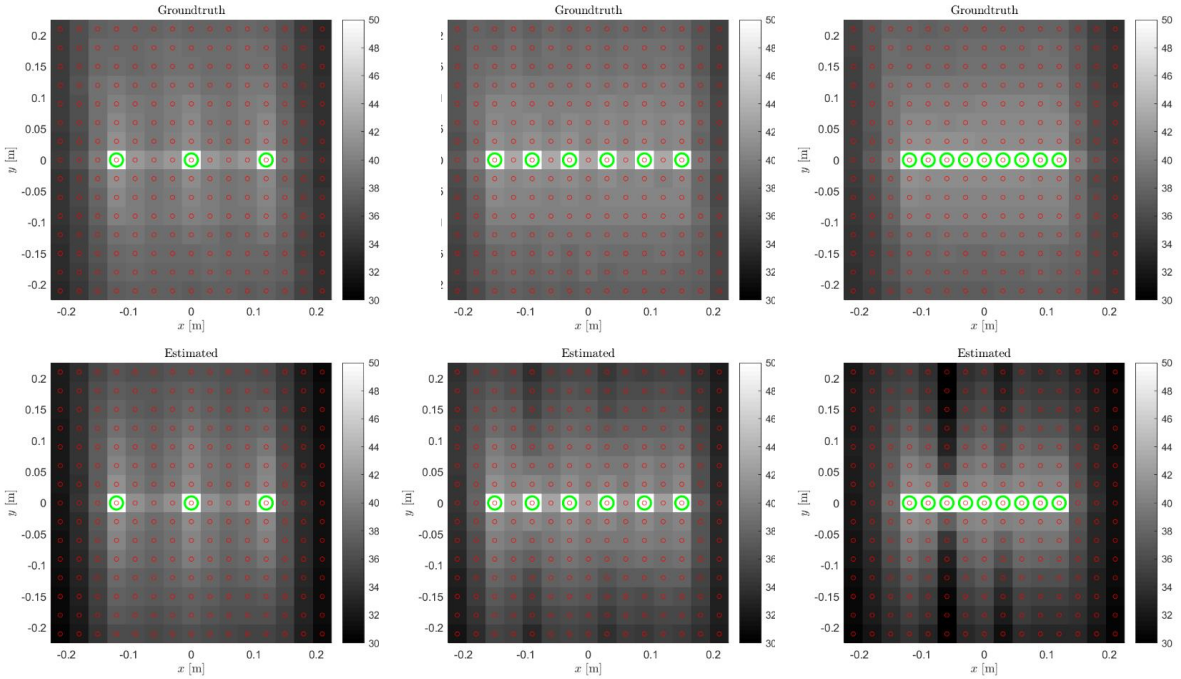


Figure 5 – PSNR [dB] for *Car* inside the viewing zone on a scale from 30-50 dB for varying camera spacing using ground truth disparity (top row) compared with estimated disparity (bottom row). Green circles are reference cameras.

	Human			**Car**		
	$B = 0.12\text{m}$	$B = 0.06\text{m}$	$B = 0.03\text{m}$	$B = 0.12\text{m}$	$B = 0.06\text{m}$	$B = 0.03\text{m}$
Ground Truth Depth	39.5	42.7	43.7	38.4	39.0	38.9
Estimated Depth	37.0	41.6	41.9	37.2	36.2	31.7

Table 1 - Minimum PSNR [dB] over central 24×24cm region of the viewing zone

Qualitative evaluation

The simulation approach allows direct comparison of ground truth depth with estimated depth. Figure 6 shows such a comparison for *Human*. It can be seen that a smaller baseline results in fewer disparity estimation errors. This is understandable since synthesis occurs from reference views at smaller spatial distance and occlusion/illumination differences are smaller for smaller baselines.

Since ray-traced ground truth images are available, a visual comparison can be done between ray-traced images, synthesized images using ground truth and synthesized images using depth estimation. Figure 7 shows such a comparison for the *Car* scene. There are hardly any visible differences between ray-traced images and synthesized images when using ground-truth depth. When using estimated depth, some image blurring results.

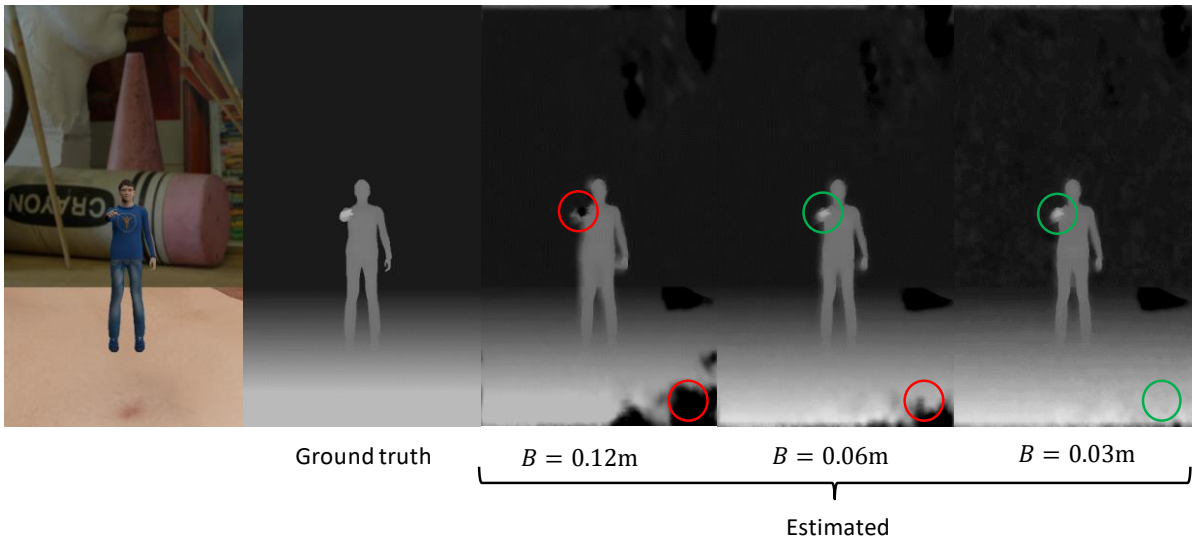


Figure 6 – Ground truth versus estimated disparity for different baselines. To produce the grey scale image, a scaling was applied to compensate for baseline difference. Errors at a larger baseline (red circle) disappear at a smaller baseline (green circle).

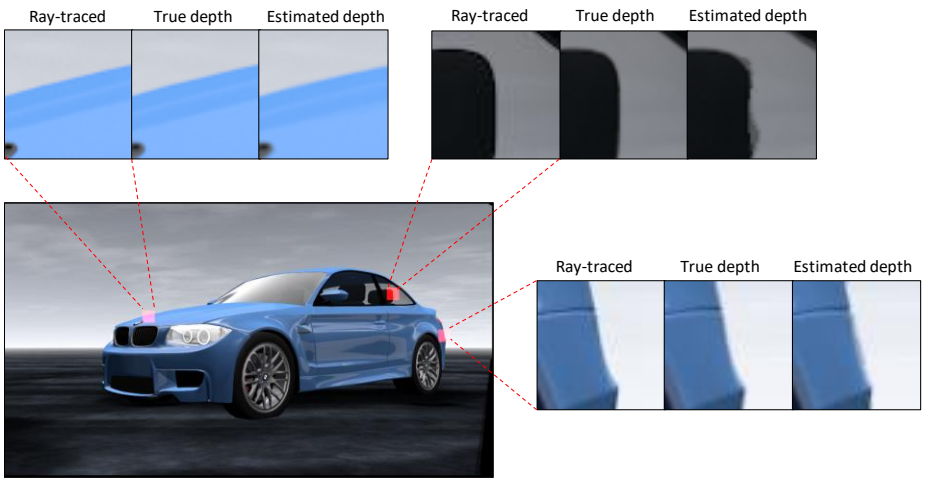


Figure 7 – Visual comparison between ray-traced, synthesized using ground truth depth and synthesized using estimated depth ($B = 0.03\mathrm{m}$) for images for different locations in the viewing zone.

CONCLUSIONS AND FUTURE WORK

We have demonstrated a simulation approach based on ray-traced images to predict the quality of a 6DoF video broadcast system and evaluated the approach using our current real-time six-camera multi-view camera system. Errors occur due to camera spacing, real-time depth estimation and view synthesis.

We have isolated modelling errors from estimation errors, which is useful when trying to improve depth estimation and view synthesis. The approach may be used for the design of more complex (360 degree) capture rigs or potentially very large camera arrays.

Both depth estimation errors and view synthesis errors can be reduced by using a reduced camera spacing. However this results in more video streams and hence a larger bandwidth. In future, we will therefore include the dynamics of view switching and adaptive streaming with realistic latencies into our simulation approach.

REFERENCES

1. Blender: https://www.blender.org/.

2. MakeHuman: http://www.makehumancommunity.org/.

3. Blender demo file: Car Viewport (Mike Pan, CC0 licence):

https://www.blender.org/download/demo-files/

4. C. Varekamp, B. Kroon, B. Sonneveldt, A. Willems. Depth based room-scale six degrees of freedom virtual reality capture and processing. *IBC 2018*.

5. G. de Haan, P.W.A.C. Biezen, H. Huijgen, O.A. Ojo. True-motion estimation with 3-D recursive search block matching. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 5, October 1993.

6. C. Varekamp, K. Hinnen, W. Simons. Detection and correction of disparity estimation errors via supervised learning. *International Conference on 3D Imaging*, 3-5 Dec. 2013.

7. L. Vosters, C. Varekamp, G. de Haan. Overview of efficient high-quality state-of-the-art depth enhancement methods by thorough design space exploration. *Journal of Real-Time Image Processing*, pp. 1–21, 2015.