EUROGRAPHICS 2016 / E. Grinspun, B. Bickel, and Y. Dobashi
(Guest Editors)

Volume 35 (2016), Number 7

Proxy-guided Image-based Rendering for Mobile Devices

Supplemental Material

Table 1: *Hardware specifications of the used devices.*

Device	CPU	RAM	GPU	Frame rate
Stick	4×1.33 GHz	2 GB	Intel HD Baytrail	35.9 fps
Stream	2×1.4 GHz	2 GB	Intel HD Haswell	66.3 fps
Pavilion	2×2.9 GHz	4 GB	Intel HD 4400	69.8 fps
Surface	4×2.9 GHz	8 GB	Intel HD 4400	107.0 fps

1. IBR algorithm pseudocode

The shading decision per pixel in Section 5.1 of the full paper can be written in pseudocode as follows:

if $\min(e_p, e_e) < \tau_{depth}$ **then**
  | fragmentDepth := 0.5 × fragmentDepth
else
  | fragmentDepth := 0.5 + 0.5 × fragmentDepth
end

if $e_p < e_e$ **then**
  | outputColor := primaryColor
else
  | outputColor := extraColor
end

2. Comparisons

Table 1 and Table 2 show the hardware specifications and framerates we obtained running our and the competing algorithms on the target devices.

Figure 1 shows a wireframe of a coarse mesh in comparison to a wireframe of our method.

Figure 2 shows the scenes used throughout the paper.

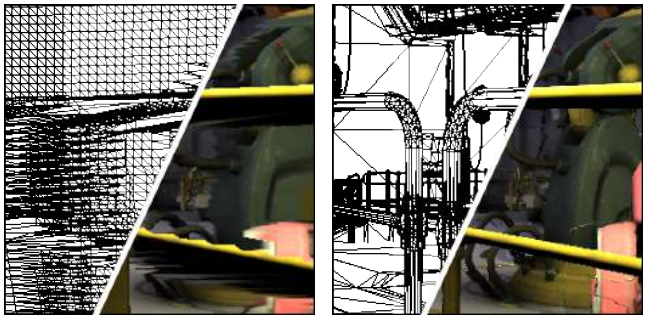
3. Perceptual Evaluation

We have assessed the effectiveness of our approach in a perceptual experiment, where it is compared to alternative approaches in terms of visual fidelity for a remote rendering scenario.

We consider five different approaches to be compared to ours that all use the same input and are rendered at the target frame rate on the Intel Compute Stick. The homography approach [Ocu15] consists of a single texture lookup per fragment assuming a constant,

Table 2: *Framerates for the used devices and techniques.*

Technique	Stick	Stream	Pavilion	Surface
Coarse mesh warp$_1$	1.9 fps	6.7 fps	12.5 fps	14.5 fps
Coarse mesh warp$_2$	6.9 fps	22.6 fps	39.1 fps	45.4 fps
Coarse mesh warp$_3$	14.4 fps	42.5 fps	64.4 fps	80.1 fps
Coarse mesh warp$_4$	23.3 fps	59.8 fps	81.7 fps	113.2 fps
Coarse mesh warp$_5$	31.3 fps	74.7 fps	91.4 fps	136.2 fps
Mesh warp	1.5 fps	5.3 fps	9.9 fps	10.1 fps
Homography	44.3 fps	143.0 fps	140.0 fps	238.8 fps
Quadtree	20.9 fps	50.7 fps	72.7 fps	93.7 fps
Point splatting	4.2 fps	15.7 fps	24.4 fps	26.6 fps
Iter. image warping	3.5 fps	15.4 fps	27.8 fps	27.7 fps
Our method	35.9 fps	66.3 fps	69.8 fps	107.0 fps



(a) Coarse mesh (b) Our method

Figure 1: *Comparison of coarse warp meshes, a classic IBR algorithm, and our approach. While our approach uses fewer primitives and is faster to draw – here shown as an overlay –, its primitives align better with the world geometry, resulting in a better novel view-image. Note, how our approach produces more accurate results with fewer triangles.*

VIKING VILLAGE ROBOT LAB COURTYARD

Figure 2: *Scenes we show throughout the paper.*

infinite depth per pixel. The mesh warp approach [MMB97] comprises of warping both views at full mesh resolution. The Outatime approach comprises of warping only the first view at a subsampled mesh resolution (factor 5) to reach a constant output frame rate of 31.3 fps. The Quadtree approach [DRE*10] is done by warping both views with a quadtree resulting in an output frame rate of 20.9 fps. We did not consider pixel splatting and Iterative image warping in our study as they did not produce favorable image qualities or require an intricate initialization. All approaches were used to extrapolate novel-views from the same pre-recorded input. Inputs are sequences of images with a spatial resolution of 2048×2048 pixels for the first and 1024×1024 pixels for the second view, showing snippets of 5 s of typical game-play motion in a scene with detailed architecture, textures, shadows and ambient occlusion as seen in Figure 1 of the full paper. Overall, two sequences for two scenes (VIKING VILLAGE, ROBOT LAB) are used. A capture is also seen in the supplemental material. Extrapolation was done to the current novel view from a stream with a delay of 128 ms arriving at 10 fps.

Subjects were asked to compare the result of our approach and a competing approach in a Two-alternative forced choice task. They were shown two image sequences in parallel in a randomized horizontal spatial layout. Both image sequences were shown on identical Dell U2412M monitors placed together in front of the participants. Sequences were played in a infinite loop, with a grey blank of 2 s before each repetition. At any moment subjects could indicate their preference by choosing "left" or "right" and providing optional feedback on a web form presented to them on a third monitor. The screens were blanked between each trial.

We recruited 30 subjects (25 M / 5 F, 25 ± 5.0 ys.) with normal or corrected-to-normal vision through different gaming related mailing lists in our institutions, as well as asking a few friends and colleagues. Our solicitation instructed participants to complete a pre-survey study and promised a meal coupon as an incentive for participation. From the pre-study survey, our 30 participants spend on average two hours and a half per day playing videogames. 85% of our participants consider themselves to be proficient or expert at videogames while 15% considered themselves to be beginners or just competent. They produced a total of 30 subjects $\times$ 2 scenes $\times$ 2 sequences $\times$ 4 video comparisons $=$ 480 binary answers.

Our method is compared to its competitors by a binomial test which found significant differences ($p < .0001$) in all cases as seen in Figure 11 of the original paper. Averaged across all other approaches, ours is preferred in 90.00% ($p < .0001$). We conclude, that subjects understand the question of visual fidelity in novel-view synthesis for remote rendering content and that our approach is preferred over other published alternatives.

The participants' feedback allowed us to better understand their choices. Participants were very often displeased by low framerate. However, small framerate differences such as those found between homography and our approach had little impact on the participants' decision. In fact, many participants perceived our approach to be "smoother" than homography. The presence of visible artifacts also caused the particpants to reject certain image sequences. It was very noticeable for the quadtree, as many of them quickly discarded it mentioning the presence of "glitches" and "distortion". Interestingly, two of the participants mentioned that the presence of artifacts was more acceptable in certain scenes. For example, the blur present in Outatime in disoccluded areas was considered "artistic" and "matching the scenario" of Viking Village while considered unacceptable for the Robot Lab. Perhaps for this reason our technique performed better against Outatime in the Robot Lab than in the Viking Village. In general the participants found our technique to produce "smoother" and "less distorted" visuals than our competitors.

Deterministic Forward warping Client-side rendering

Figure 3: *Alternatives for dynamic scenes.*

4. Dynamic scenes

A limitation of our method is the weak support for dynamic scenes. We implemented three different fallbacks for supporting certain kinds of dynamics, each having different advantages and drawbacks (Figure 3). Since it is difficult to show animations in the paper, we refer the reader to the `dynamics` directory in our supplementary material where we show videos demonstrating each of the three techniques described below.

Deterministic dynamics Our method directly supports non-static scene elements with *deterministic* motion, i.e., it is known in advance for all time steps and cannot be influenced by the user. In this case the client simply applies the motion to the scene proxy and accounts for it when projecting fragments into the source views. Examples of this type of motion include periodic motions (e.g., rotating blades of a wind mill) or noise-based animations (e.g., swaying trees).

Depth map warping The server can render the dynamic objects into a color / depth buffer that is transmitted to the client and forward-warped with one of the techniques described in Section 2 of the full paper. The motion and object complexity is not restricted in any way, but the animation uses the server frame rate and is delayed by the network latency.

Client-side rendering The client can render any object into the 3D scene, either before or after our method. The z-buffer culling will work normally. This method supports any kind of motion, and objects may react to user actions. However, it is limited to simple objects that can be rendered quickly on our target devices.

References

[DRE*10] DIDYK P., RITSCHEL T., EISEMANN E., MYSZKOWSKI K., SEIDEL H.-P.: Adaptive image-space stereo view synthesis. In *Proc. VMV* (2010), pp. 299–306. 2

[MMB97] MARK W. R., MCMILLAN L., BISHOP G.: Post-rendering 3D warping. In *Proc. i3D* (1997). 2

[Ocu15] OCULUS VR: Oculus mobile SDK documentation, 2015. 1