Real-Time Depth Image-based Rendering with Layered Dis-occlusion Compensation and Aliasing-Free Composition

Sergey Smirnov, Atanas Gotchev

Tampere University of Technology
Department of Signal Processing
Korkeakoulunkatu 1, 33720 Tampere, Finland

ABSTRACT

Depth Image-based Rendering (DIBR) is a popular view synthesis technique which utilizes the RGB+D image format, also referred to as view-plus-depth scene representation. Classical DIBR is prone to dis-occlusion artefacts, caused by the lack of information in areas behind foreground objects, which appear visible in the synthesized images. A number of recently proposed compensation techniques have addressed the problem of hole filling. However, their computational complexity does not allow for real-time view synthesis and may require additional user input. In this work, we propose a hole-compensation technique, which works fully automatically and in a perceptually-correct manner. The proposed technique applies a two-layer model of the given RGB+D imagery, which is specifically tailored for rendering with free viewpoint selection. The main two components of the proposed technique are an adaptive layering of depth into relative 'foreground' and 'background' layers to be rendered separately and an additional blending filtering aimed at creating a blending function for aliasing cancellation during the process of view composition. The proposed real-time implementation turns ordinary view-plus-depth images to true 3D scene representations, which allow visualization in the fly-around manner.

1. INTRODUCTION

RGB+D (also known as *view-plus-depth*) is an image format which combines color pixels with aligned depth values, thus representing the 3D structure of the visual scene from a particular viewpoint. Given one or more RGB+D images, new, *virtual* views at arbitrary camera locations can be synthesised by *Depth Image-based Rendering* (DIBR). DIBR is an essential tool in many 3D visual applications, such as free-viewpoint TV, visualization on multi-view 3D displays, multi-view video coding, etc. In contrast with computer graphics, DIBR is essentially about visualizing natural scenes, where depth maps have been captured by dedicated depth sensors, or estimated through *Depth-from-Motion* or *Depth-from-Stereo* methods.

Since an RGB+D image is related with a particular camera position (viewpoint), dis-occlusion artefacts usually occur during view synthesis. Dis-occlusion artefacts appear in image regions which are hidden in the given view, and become visible in the synthesized one. They appear as holes in the rendered image and their location and size depend on the 3D scene geometry and the virtual camera position. In order to minimize the effect of dis-occlusions, one can acquire and use two or more RGB+D input views, and limit the location of the virtual camera between them. Generating such virtual view is also known as *view interpolation*.[1,2] However, in the more general *view extrapolation* case, or when single *view-plus-depth* image is available, the dis-occlusion problem still exists. In order to minimize the perceptual effect of dis-occluded areas in the rendered images, the corresponding holes should be filled in with consistent colors or textures. However, classical hole-filling methods are usually costly and not suitable for real-time applications. Real-time performance is an important requirement which imposes stringent demands for the implemented methods.

Another important aspect of the problem is that dis-occlusion artefacts are anisotropic. Due to perspective re-projection, displacements of areas being closer to the camera (e.g. foreground) are larger than those of distant ones (background). Hence, dis-occluded pixels are likely belonging to the background, and should be compensated accordingly (with the respective background colors or textures). Background-favoured compensation can be achieved by applying directional compensation methods,[3] however such methods work only for the case of

E-mail: firstname.secondname@tut.fi

stereoscopic rendering (i.e. when the virtual camera is shifted only horizontally from the given input view). In contrast to stereoscopic rendering, *free-viewpoint rendering* creates holes at any side of the foreground object, thus invalidating the directional compensation approach.

In this paper, we propose a simple yet effective dis-occlusion compensation method, specifically aimed at free-viewpoint rendering, and targeting real-time implementation. The proposed method utilizes an automatic segmentation of the given RGB+D image in local foreground and background areas accomplished by a locally-adaptive thresholding of the inverse depth maps[4] . We compare existing hole-filling approaches aimed at real-time with the new one as to validate the proposed layered compensation scheme. We also present a complete OpenGL-based solution for free-viewpoint DIBR capable of rendering FullHD video at 25 frames per second (FPS) on commodity hardware.

The rest of the paper is organized as follows. In Section 2, we overview existing dis-occlusion compensation methods and discuss their complexity, while in Section 3, we describe the proposed compensation scheme and discuss its real time implementation in Section 4. Then, in Section 5, we compare the performance of the proposed approach against other real-time oriented approaches in terms of PSNR and perceived visual quality. Section 6 summarizes the presented work.

2. PRIOR ART

Early attempts to handle dis-occlusions have been aimed at pre-processing the given depth map in order to avoid dis-occlusions. A simple Gaussian pre-filtering of the given depth map eliminates large discontinuities and hense minimizes the size of dis-occlusion holes. The price paid for this is the deformed scene geometry.[5,6] More comprehensive adaptive smoothing filters can only mitigate the geometric distortions without completely removing them.[7]

2.1 Linear hole-filling

One of the first dis-occlusion compensation methods aimed at free-viewpoint rendering was the *linear hole-filling*.[8] This real-time oriented approach suggests blurring the rendered image with a fixed-kernel smoothing filter, and then replacing the valid pixels with their original values. Only valid pixels are accumulated during the averaging operation. Let I denotes the rendered image and N denotes the binary map of valid pixels (also called *normalization field*). The linear hole-filling method can be expressed as:

$$\tilde{I}(x,y) = \begin{cases} \frac{\sum_{(u,v)\in\Omega_{x,y}} I(u,v)\cdot N(u,v)}{\sum_{(u,v)\in\Omega_{x,y}} N(u,v)}, & \text{if } N(x,y)=0 \\ I(x,y) & \text{otherwise} \end{cases}, \tag{1}$$

where $\Omega_{x,y}$ is the support of a fixed kernel around the processed pixel (x,y).

This method can be implemented by constant-complexity filtering based on Summed Area Tables[4] . However, the overall complexity is not fixed since the actual size of dis-occlusion holes is not known. Filtering with large kernels may cause over-smoothing in areas of small holes, while filtering with small kernels may lead to incomplete compensation, and impose the necessity of applying the compensation method several times.

2.2 Hierarchical hole-filling

The *Hierarchical Hole-Filling* method (HHF)[9] was proposed as a technique capable of treating holes of unknown size with an acceptable complexity. In order to fill in missing pixels, the rendered image is decomposed in a Gaussian pyramid by consecutive fixed-kernel averagings $F_{5\times5}\{\cdot\}$, followed by image decimation. In the notations of the linear hole-filtering, HHF can be expressed as:

$$\begin{aligned} I^{i+1} &= \frac{F_{5\times5}\{I^i \cdot N^i\}}{F_{5\times5}\{N^i\}} \downarrow_2 \\ N^{i+1} &= F_{5\times5}\{N^i\} \downarrow_2 > 0, \end{aligned} \tag{2}$$

where the normalization field N^i is recalculated for each pyramid level i.

A compensated version of the rendered image is reconstructed by iterative image up-sampling from the coarser level to the finer one and replacing valid pixels by their original values:

$$\tilde{I}^i(x,y) = \begin{cases} \tilde{I}^{i+1} \uparrow_2 (x,y), & \text{if } N^i(x,y) = 0 \\ I^i(x,y) & \text{otherwise} \end{cases} . \tag{3}$$

This coarse-to-fine technique allows for filtering with arbitrary-large kernels for no extra computational cost, while at the same time offers reasonable smoothing of smaller holes.

2.3 Depth-adaptive hierarchical hole filling

The dis-occlusion compensation performed by the HHF method is deficient in the sense that it introduces blurring, due to its inability to differentiate background from foreground pixels. *Depth-Adaptive Hierarchical Hole-Filling* (DA-HHF)[10] was proposed to resolve this problem. Essentially, the method suggests penalizing weights for the foreground colors in the averaging filter. An additional weight map is proposed for controlling the weights of pixels with different disparity:

$$W(x,y) = \frac{\gamma}{\sigma}\left(1 - e^{-\frac{(\beta_{max}+\delta)}{D(x,y)}}\right), \tag{4}$$

where parameters are derived as follows: $\gamma = \frac{3}{2}\beta_{center} + \beta_{min}$, $\sigma = \frac{4}{3}\beta_{center} + \beta_{min}$ and $\delta = \frac{1}{2}\beta_{center} + \beta_{min}$, and β_{min}, β_{max} and β_{center} are minimum disparity, maximum disparity, and the central disparity respectively.

The input image I in Equation 3 is substituted by a pre-weighted one: $I^W(x,y) = I(x,y) \cdot W(x,y)$. It is then compensated by the conventional HHF method. The weighting map W is also compensated thus getting two hole-filled components: $\tilde{I}^W$ and $\tilde{W}$. The desired image is reconstructed by a subsequent inverse weighting:

$$\tilde{I}(x,y) = \frac{\tilde{I}^W(x,y)}{\tilde{W}(x,y)}. \tag{5}$$

The DA-HHF scheme significantly reduces the weights of foreground colors in compensated areas, however, blending still exists in areas having complex foreground details.

2.4 Recursive hole-filling

In contrast with conventional finite impulse response (FIR) filters, the *infinite impulse response* (IIR) filtering can inherently solve the problem with unknown hole sizes, in just one filtering pass. A *recursive hole-filling* technique has been proposed in,[11] which utilises an IIR-based *recursive bilateral filter* as a tool for dis-occlusion compensation.

In its simplest form, the recursive bilateral filter is expressed as one-tap IIR filter, applied in casual ($\overrightarrow{y}$) and anti-casual ($\overleftarrow{y}$) direction for every row of the desired image:

$$y[i] = x[i] + \mu_{i,i-1} \cdot y[i-1], \tag{6}$$

where the color-weight component $\mu_{i,i-1}$ is calculated from the corresponding color image I:

$$\mu_{i,i-1} = \min_{j \in (R,G,B)} e^{-\frac{|I^j[i] - I^j[i-1]|}{\sigma_c}}, \tag{7}$$

i denotes a particular index in the row or in the column of an image and the parameter σ_c regulates the strength of the filtering.

The two responses are merged $\bar{y}[i] = \overleftarrow{y}[i] + \overrightarrow{y}[i] - x[i]$, and then the same process is applied to each column of the horizontally filtered image. Normalization has to be done, since colors become over-saturated due to the accumulation operation. Since the rendered image has no color values in the dis-occluded areas, an additional weighting mechanism has been proposed to handle those pixels:

$$\mu_{i,i-1} = \begin{cases} 1 & \text{if } |D[i] - D[i-1]| < T_D \\ 0 & \text{otherwise} \end{cases} , \tag{8}$$

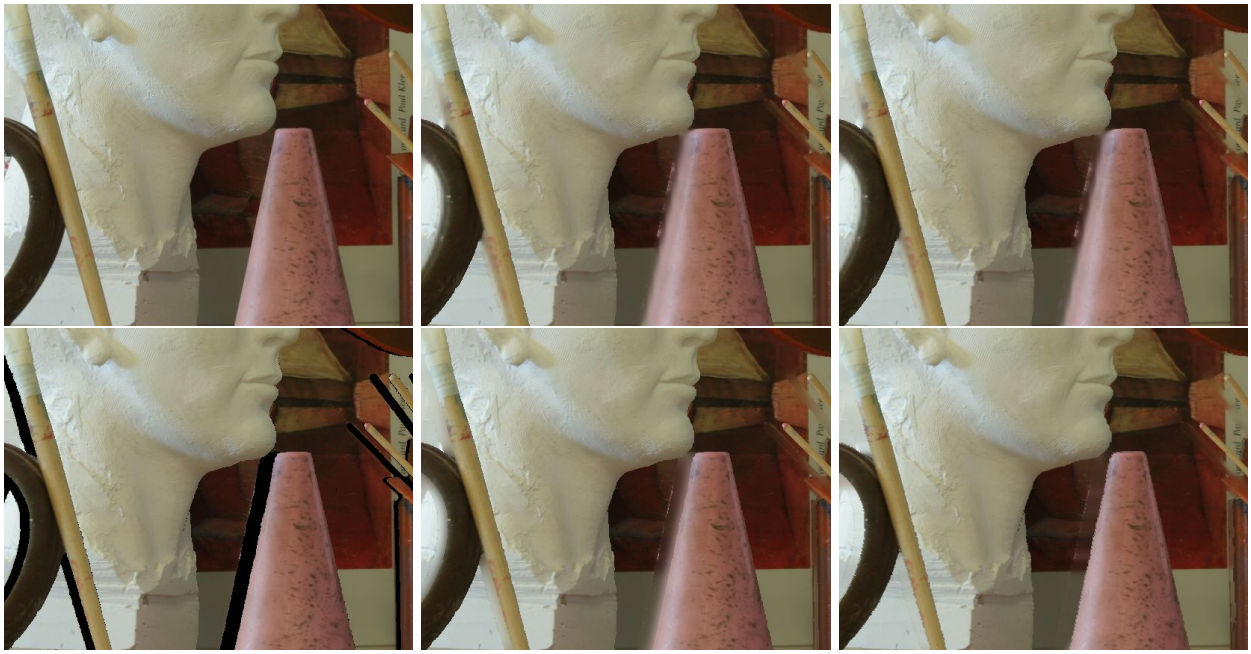

Figure 1. Hole-filling results for *Art* dataset (enlarged). First row: true desired image, HHF,[9] DA-HHF[10]; second row: no compensation, recursive hole-filling, proposed method.

where D is a predicted disparity value at missing pixel and T_D is a threshold value, empirically set to 4.

The recursive implementation of the bilateral filter is significantly faster than the conventional one, still the complexity is prohibitive for real-time implementation. Furthermore, the method requires predicted disparity values at positions of missing pixels, therefore it cannot be directly used for rendering from single RGB+D image, unless some depth prediction has been applied.

Figure 1 illustrates the quality of rendered images applying the three methods presented above (along with the proposed method, to be discussed futher), for the *Art* dataset.

2.5 Non real-time layered approaches

Some recent works[12,13] have suggested separating the rendered image into background and foregrounds layers and an independent background hole-filling. These techniques lead to an improvement of the resulting visual quality of compensated images, however, blur and inconsistencies introduced by the rendering step can be significantly amplified during the hole-filling procedure. Furthermore, in ref.[12] , the foreground extraction is made by an absolute depth map thresholding. This is equivalent to extracting only areas close to the camera. Other areas causing significant dis-occlusion artefacts might be missed.

Other layered decompositions have been proposed for achieving visually-meaningful dis-occlusion compensation and maintaining the consistency between multiple rendered views.[14,15] However, the complexity of the layer extraction procedure is quite high and requires additional user input. Therefore, these approaches are not considered for real-time use and are out of scope of our comparisons. Another high-quality approach is based on color texture inpainting, which is however also computationally demanding.

3. PROPOSED APPROACH

There are two problems in applying hole-filling approaches based on background/foreground segmentation. The first problem is to tackle the increased complexity.[12,13] The second problem is to find an adequate segmentation mechanism, which would separate only areas where the dis-occlusion happens.[12] This is related also with the question how many layers are sufficient for the underlined task.

Figure 2. Foreground segmentation by locally-adaptive thresholding for the *Art* dataset. First row: Given disparity map $D(\cdot)$, foreground mask obtained by absolute thresholding as in,[12] corresponding background layer image; second row: positive part of the residual image $(D - D_{avg})$, foreground mask obtained by locally-adaptive thresholding, corresponding background layer image

We propose using a simple two-layer model of image composition, similar to what have been used in *image matting*. The model composes the input image from a background image of the scene and foreground objects:

$$I(x,y) = \alpha(x,y) \cdot I^F(x,y) + (1 - \alpha(x,y)) \cdot I^B(x,y), \tag{9}$$

where α is some unknown matte, I^B is a background image of the scene and I^F is a foreground image. Correspondingly, background and foreground layers have their own depth maps D^B and D^F.

As the scene may contain overlapping foreground objects, one can extend the model to accommodate higher number of layers[14] and arrange different layers as locally-connected zones.[15] We favour the *two-layer* model based on proper locally-adaptive decomposition, which provides visually meaningful hole compensation for reasonable computation cost.

3.1 Layered decomposition

Depth maps are normally used in order to separate foreground scene objects from the background area.[12–15] However, a global segmentation[12] based on single threshold does not always properly mark the areas causing the most significant dis-occlusions. We propose applying locally-adaptive thresholding[4] on the *inverse depth map* (also known as *generalized disparity map*[16])

$$M^F(x,y) = D(x,y) > (D_{avg}(x,y) + \beta_F), \tag{10}$$

where M^F is a binary foreground mask and D_{avg} is a low-pass filtered version of the inverse depth map D. The mask M^F is used in the image formation model in Equation 9 at the place of matte factor α.

The so-introduced segmentation is local. It underlines areas which are convex with respect to the local neighborhood. Our assumption is based on the fact that pixel shifts introduced during *3D image warping* are proportional to their disparity values, hence, local disparity maxima better represent foreground occluders than global foreground segments, which, instead, represent close-to-camera areas. The threshold value β_F is the

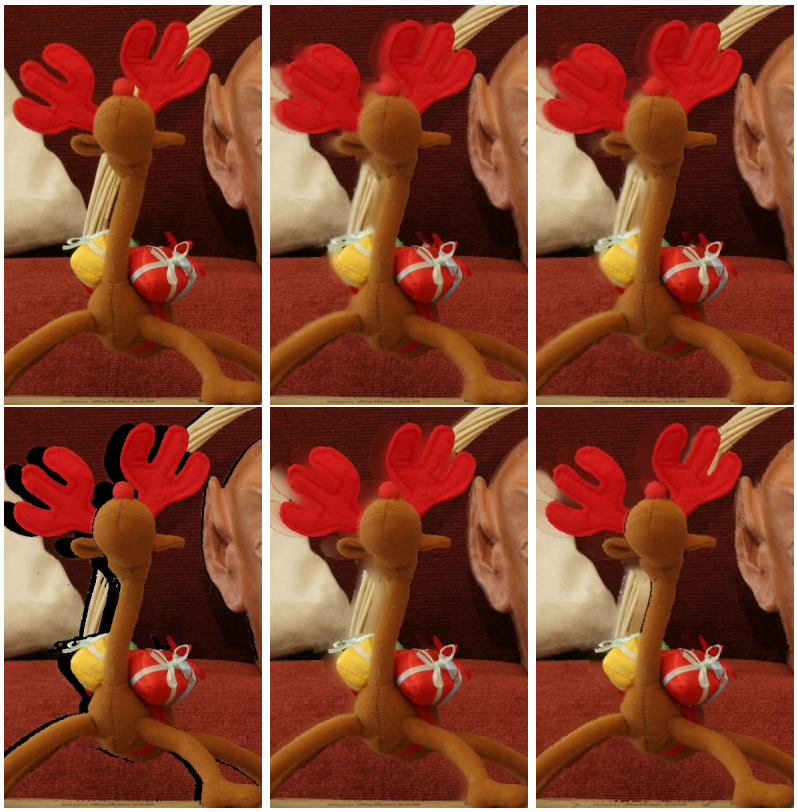

Figure 3. Hole-filling results for *Reindeer* dataset (enlarged). First row: true desired image, HHF, DA-HHF; second row: no compensation, recursive hole-filling, proposed method.

minimum disparity step-up value considered as a significant occluder. Holes caused by lower disparity shifts can be compensated by the conventional methods. Figure 2 illustrates the locally-adaptive foreground/background segmentation for the *Art* scene.[17] In contrast to the absolute thresholding method, the proposed segmentation nicely separates fine foreground details from the background image areas. In the case of an overlap between foreground objects, the resulting foreground mask completely covers the closest object, and does not cover those in the immediate neighborhood around it (see the green boxes in the figure). This property helps in compensating holes in partially occluded objects, while they all otherwise appear as foreground. Therefore, the proposed locally-adaptive segmentation allows treating multiple overlapped foreground objects with just two-layer decomposition.

3.2 Layered rendering

After the above-described segmentation, the extracted background layer is represented as an RBG+D frame, with some missing pixels. Similarly to the initial hole-filling problem, the hole sizes are content-dependent and unknown in advance. However, in the segmented background image, the foreground textures have been completely removed, thus making the hole-filling significantly simpler.

Conventional methods, such as HHF or DA-HHF can be applied to the background layer (both to the colour and depth images) in order to reconstruct both missing color and depth values. Since the reconstructed depth values may appear blurred, the rendering of the background layer will result in decreased amount of dis-occlusion holes. Some depth discontinuities may be strong enough to introduce dis-occlusions and normal post-rendering hole-filling still has be applied, e.g. by a second pass of HHF or DA-HHF. At this stage, the background textures may interfere with the remaining foreground parts of the scene, but their influence is significantly reduced.

The second rendering pass is for the foreground layer and should take into account the foreground mask $M^F(\cdot)$. The mask also undergoes a 3D image warping guided by the desired viewpoint position. The imposed

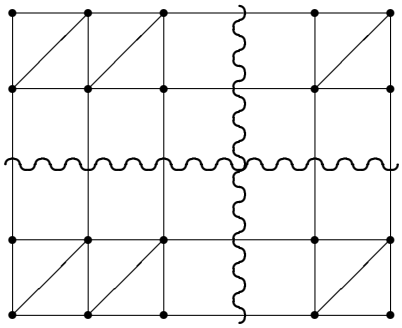

Figure 4. Construction of polygonal mesh from nodes of regular grid.

interpolation turns the initial binary mask into a grayscale one (i.e. containing non-integer values). Using the projected (non-integer) foreground mask facilitates mixing the rendered background and foreground images (see Equation 9). Such blending helps to avoid aliasing artefacts, usually appearing in the synthesized images between background and foreground objects. As illustrated in Figures 1 and 3, the proposed layered approach handles problematic areas quite well. Other methods compensate the same areas with foreground colors instead.

3.3 Complexity considerations

The proposed approach includes additional processing steps, namely the local foreground/background segmentation, initial background hole-filling, and second rendering pass. Still, the asymptotic complexity is linear with regard to the number of pixels in the frame.

The algorithm offers several options for accelerating the execution time. First, the depth-smoothing filter used in the proposed foreground segmentation procedure can be selected in different ways depending on the targeted speed and used computing platform. For GPU implementation, a hardware-accelerated Gaussian pyramid decomposition can be used for smoothing the disparity image. For CPU implementation one can select between box averaging, recursive Gaussian filter, or any other fast smoothing filter. The original proposal includes an initial hole-filling in the background image before the actual rendering. This step can be omitted if the speed is of primary importance. The hole-filling will be applied only after the rendering thus resulting in slightly inferior results. Furthermore, the foreground rendering pass is applied on relatively small number of pixels, which also contributes to the efficiency of the overall procedure.

4. REAL-TIME IMPLEMENTATION

Recent tools for GPU programming (e.g. shaders) are quite suitable for implementing high-quality DIBR in real-time. In terms of the OpenGL programming language, each pixel of a given RGB+D image can be represented as a vertex of polygonal model and transmitted to the GPU. Polygons (triangles) can be constructed out of four neighboring pixels by pairs, as illustrated in Figure 4. Each vertex of the constructed model has one-to-one correspondence with a pixel in the underlying color image, thus making reverse texture warping simple.

Memory buffers with texture coordinates and vertex indexes can be initialized at the beginning and kept in the GPU memory for further rendering, with no need for recalculation at every frame. With this setting, the DIBR process will nearly correspond to classical computer graphics rendering passes, where the actual 3D coordinate of each vertex has to be calculated in *vertex shader program* using values of the transmitted depth texture. The position of each vertex in the desired camera viewpoint can be found by re-projecting the scaled homogeneous vector of its coordinates at the input camera $\bar{\mathbf{x}}_{inp} = z_{inp} \cdot (u_{inp}, v_{inp}, 1, \frac{1}{z_{inp}})^T$:

$$\bar{\mathbf{x}}_{virt} = C_{virt} \cdot C_{inp}^{-1} \cdot \bar{\mathbf{x}}_{inp}, \tag{11}$$

where C_{inp}^{-1} represents the inverse camera matrix of the given camera and C_{virt} represents the camera matrix of the desired view (OpenGL *Normalized Device Coordinates* should be taken into account).

Such polygonal surface will automatically fill in holes in the rendered image, however in perceptually wrong manner. The *geometry shader program* can remove some particular triangles, which appear at depth discontinuities, in order to keep the resulting dis-occlusion holes for later compensation, applying better hole-filling algorithms. The hardware rasterization can resolve some difficulties, inherent for CPU-based DIBR implementations, such as the appearance of cracks and varying density of projected pixels of different depth. In order to further process the rendered image (e.g. to compensate resulting dis-occlusion holes) one should use a *Render to a Texture* technique to be able to transmit the resulted image to the next rendering pass.

Implementing visually-correct dis-occlusion compensation in GPU-based DIBR imposes some challenges. Despite the reduced complexity of some of the above presented dis-occlusion filling methods, namely HHF,[9] DA-HHF[10] and recursive HF,[11] their direct GPU implementation is still problematic. In order to accelerate their performance we have simplified some steps and unified some others for all hole-filling methods. We implemented the HHF and DA-HHF methods on GPU. Due to low algorithmic parallelism, the recursive hole filling was implemented only for CPU-based DIBR. In this way, we got performance of the modified algorithms sometimes better than what was reported in corresponding publications.

4.1 Implementation of HHF and DA-HHF

The HHF and DA-HHF methods are attractive for GPU-based DIBR implementation, since modern GPUs offer hierarchical decomposition of given textures in a hardware-accelerated manner. Different levels of the decomposition pyramid can be accessed from both *vertex* and *fragment shaders*, which is also known as sampling the *levels of detail* (LOD) or MIPMAP levels. The requirement to use fixed-kernel filtering at each decomposition step decreases the efficiency of this approach. Another difficulty is related with the inability to process each level of the image pyramid inclusively. In order to overcome these difficulties, we propose avoiding fixed-kernel averaging when modifying the normalization at the pyramid-sampling method in the *fragment shader program*.

For the simplified HHF, the RGB texture of the previously rendered frame in floating-point format along with the normalization field are transmitted to the GPU memory for consecutive generation of pyramid levels. In order to reconstruct the compensated image, the fragment shader is utilized to sample the pyramids of the two transmitted images

$$\hat{I}_{HHF}(x,y) = \begin{cases} \frac{\sum_{i=1..L} \gamma^i \cdot I^i(x,y) \cdot N^i(x,y)}{\sum_{i=1..L} \gamma^i \cdot N^i(x,y)} & if N(x,y) = 0 \\ I(x,y) & otherwise \end{cases}, \tag{12}$$

where I^i and N^i are i-th levels of the pyramids constructed from the color image and normalization field; L is the maximum number of levels and γ is an additional weighting parameter used to reduce oversmoothing of smaller holes, and empirically set to 0.25 .

The GPU implementation of DA-HHF may require an additional rendering pass to calculate depth-adaptive weights and the pre-weighted rendered image, however, with a modern GPU extension, called *Multiple Render Targets* (MRT), these can be obtained by the first rendering pass. Another difficulty of DA-HHF GPU implementation is the considerable complexity of weigth calculation (Eq. 4) as it involves the exponent function and numerous floating-point divisions. With no significant performance drawback, the depth-adaptive weights can be simply re-defined as being proportional to the depth itself:

$$W(x,y) = \frac{Z(x,y) - Z_{near}}{Z_{far} - Z_{near}}, \tag{13}$$

where $Z(x,y)$ is the provided (not inverted) depth map.

In fact, due to the *normalized device coordinates* used in OpenGL, the resulting depth of the fragment can directly be used as weight value.

Our particular GPU implementation of DA-HHF has achieved rendering at a rate of about 30 FPS for input video of Full HD resolution while rendering at user-controlled virtual camera position.

4.2 Implementation of recursive hole-filling

Due to the low parallelism of the algorithm, the recursive hole-filling[11] cannot be directly implemented on GPU. The algorithm requires predicted depth values at the positions of the missing pixels and has relatively high complexity because of the bilateral filter weights. These issues make its CPU-based implementation also problematic in terms of real-time.

On the other hand, IIR filters are quite attractive for hole compensation due to their infinite support. Therefore, we have considered the recursive implementation of a Gaussian filter as an alternative to the recursive bilateral filter (Eq. 6). In contrast to the latter, the former offers constant accumulation weight $\mu_{i,i-1}$. Moreover, the sought depth adaptivity can be achieved by using linear depth weights (Eq. 13) instead of the normalization field (Eq. 8).

4.3 Implementation of proposed layered rendering

Despite the additional processing steps, required for foreground/background segmentation and pre-processing of the background layer, the GPU-based implementation of the proposed layered DIBR can still be implemented in two rendering passes, which makes it competitive to the HHF and DA-HHF methods. The input color and depth textures can be pre-processed by hardware-accelerated hierarchical decomposition. Thus, the background layer extraction, its HHF-based compensation and subsequent rendering to a virtual camera position can be done during the first rendering pass. The rendered background layer image can be transmitted as an additional input texture for the second rendering pass, which renders directly on the screen. In the *fragment shader* of the second rendering pass, the rendered foreground layer can be blended with the provided rendered background layer. The foreground mask, rendered along with the foreground image plays the role of image matte, and is used for blending the foreground area with the underlying background image.

Our particular GPU implementation has achieved rendering at a rate of 25 FPS for Full HD input video while rendering at user-controlled virtual camera location.

5. EXPERIMENTS

In order to validate the proposed layered rendering technique, we have made experiments with four multi-view datasets from the Middlebury collection,[17] namely *Art, Reindeer, Moebius and Dolls*. They contain ground truth depth maps for at least one view. Since some of the provided depth maps are incomplete, we semi-manually processed them in order to fill in the missing data. Thus, we formed reference (input) RGB+D frames. Given the input frames, we rendered six other images, at positions of existing views. We estimated the PSNR between the ground true images and the rendered ones. Prior to PSNR calculation, the rendered images were cropped from the left side or from the right to remove missing pixel areas, generated by the differences in the field of view of the reference and desired cameras. Figures 1 and 3 present zoomed areas of images hole-compensated by the selected methods.

We have implemented the conventional and layered DIBR rendering approaches combined with three hole-filling methods (HHF, DA-HHF and Recursive) with the proposed simplifications. For comparison purposes, the methods have been implemented as CPU-based C/MEX interface functions for the Matlab environment. Algorithmically, they correspond to the proposed GPU-based implementations (except for the recursive hole filling). Due to the lack of rasterization, our CPU-based DIBR and layered DIBR implementations slightly differ from the GPU-based ones, however for small camera displacements, their performance is nearly identical.

Figure 5* contains plots showing the performance in terms of PSNR. The horizontal axes show indexes of available camera views in the dataset, where index 0 corresponds to the reference image. PSNR values for the reference camera positions were not calculated. One can see that for reasonable displacement of the virtual camera, the proposed method significantly outperforms the other methods. For higher shifts, PSNR stops giving adequate results, still our method maintains a better performance.

*Full color figures are available in the electronic version of the paper.

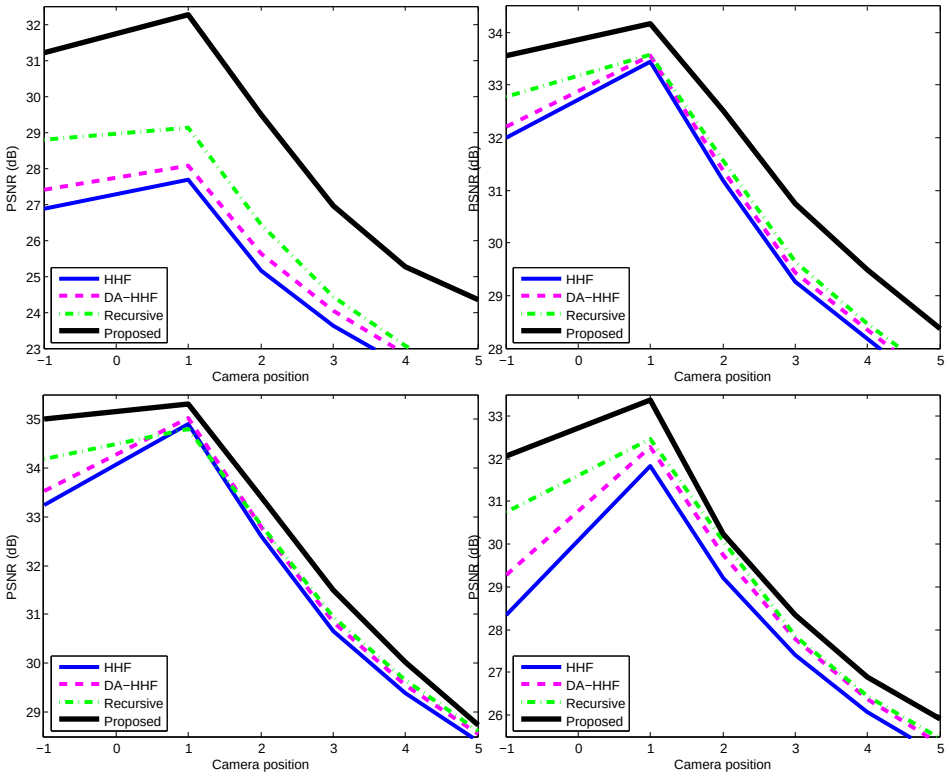

Figure 5. Hole-filling performance in terms of PSNR for selected approaches. Top row: *Art* and *Dolls* datasets, second row: *Moebius* and *Reindeer* datasets.

6. CONCLUSIONS

In this work, a depth image-based rendering method with a layered dis-occlusion compensation scheme has been proposed. Three state-of-the-art low-complexity approaches have been considered for comparison and re-implemented with real-time performance assumptions in mind. The proposed method and two other methods have been implemented on GPU using OpenGL programming language and tools. Their implementations have shown comparative rendering rates at 25 - 30 FPS for FullHD input video source while rendering at user-controlled virtual camera location. Comparison of all methods has been done involving their CPU-based implementations for easier control of parameters and fairer interpretation of results. The numerical and visual results prove the superiority of the proposed approach.

Acknowledgement

This work has been supported by a collaborative project with Nokia Research Center co-funded by the Finnish Funding Agency for Technology and Innovation (TEKES).

REFERENCES

[1] Muller, K., Merkle, P., and Wiegand, T., "3-D Video Representation Using Depth Maps," *Proceedings of the IEEE* **99**(4), 643 – 656 (2011).

[2] Herrera, D. C., Kannala, J., and Heikkila, J., "Multi-View Alpha Matte for Free Viewpoint Rendering," in [*Computer Vision. Computer Graphics Collaboration Techniques*], 98–109 (2011).

[3] Po, L.-M., Zhang, S., Xu, X., and Zhu, Y., "A new multidirectional extrapolation hole-filling method for Depth-Image-Based Rendering," in [*Proc. Int. Conf. Image Processing (ICIP)*], 2589–2592 (2011).

[4] Bradley, D. and Roth, G., "Adaptive thresholding using the integral image," *Journal of Graphics, GPU, and Game Tools* **12**(2), 13–21 (2007).

[5] Zhang, L. and Tam, W., "Stereoscopic image generation based on depth images for 3D TV," *IEEE Transactions on Broadcasting* **51**(2), 191 – 199 (2005).

[6] Cho, H.-W., Chung, S.-W., Song, M.-K., and Song, W.-J., "Depth-Image-Based 3D Rendering with Edge Dependent Preprocessing," in [*Int. Midwest Symposium on Circuits and Systems (MWSCAS)*], 1–4 (2011).

[7] Lee, S.-B., Kim, S.-Y., and Ho, Y.-S., "Multi-view Image Generation from Depth Map and Texture Image Using Adaptive Smoothing Filters," in [*Int. Conf. on Embedded Systems and Intelligent Technology*], (2008).

[8] McMillan Jr, L., [*An Image-based Approach to Three-dimensional Computer Graphics*], University of North Carolina at Chapel Hill (1997).

[9] Solh, M. and AlRegib, G., "Hierarchical Hole-Filling (HHF): Depth Image Based Rendering without Depth Map Filtering for 3D-TV," in [*Proc. Multimedia Signal Processing (MMSP)*], 87 – 92 (2010).

[10] Solh, M. and AlRegib, G., "Hierarchical Hole-Filling For Depth-Based View Synthesis in FTV and 3D Video," *IEEE Journal of Selected Topics in Signal Processing* **6**(5), 495–504 (2012).

[11] Cigla, C. and Alatan, A., "An efficient hole filling for depth image based rendering," in [*Proc. ICME*], 1–6 (2013).

[12] Wang, K., An, P., Cheng, H., Li, H., and Zhang, Z., "A New Method of DIBR Based on Background Inpainting," *Advances on Digital Television and Wireless Multimedia Communications* **331**, 478–484 (2012).

[13] Lu, Z., Zhu, Y., and Chen, J., "A Novel Filling Disocclusion Method Based on Background Extraction in Depth-Image-Based-Rendering," in [*Int. Conf. on Applied Informatics and Computing Theory (AICT)*], 65–70 (2012).

[14] Świrski, L., Richardt, C., and Dodgson, N. A., "Layered Photo Pop-up," in [*SIGGRAPH Posters*], 36:1–36:1, ACM (August 2011).

[15] Wolinski, D., Le Meur, O., and Gautier, J., "3D View Synthesis with Inter-view Consistency," in [*Proc. 21st International Conference on Multimedia*], 669–672, ACM (October 2013).

[16] Scharstein, D. and Szeliski, R., "A Taxonomy and Evaluation of Dense Two-frame Stereo Correspondence Algorithms," *International Journal of Computer Vision (IJCV)* **47**(7), 7–42 (2002).

[17] Scharstein, D. and Pal, C., "Learning Conditional Random Fields for Stereo," in [*Proc. Computer Vision and Pattern Recognition (CVPR)*], 1–8 (2007).