ORIGINAL ARTICLE



Ray tracing-based construction of 3D background model for real-time stereoscopic rendering of live immersive video

Youngwook Kim¹ · Jehyeong Yun² · Joungil Yun³ · Sangwoon Kwak³ · Insung Ihm¹

Received: 15 January 2023 / Accepted: 7 December 2023 © The Author(s) 2024

Abstract

Immersive video stored in multiview video-plus-depth format can provide viewers with vivid immersive experiences. However, rendering such video in real time in immersive environments remains a challenging task due to the high resolution and refresh rate demanded by recent extended reality displays. An essential issue in this immersive rendering is the disocclusion problem that inevitably occurs when virtual views are synthesized via the de facto standard 3D warping technique. In this paper, we present a novel virtual view synthesis framework that, from a live immersive video stream, renders stereoscopic images in real time for a freely moving virtual viewer. The main difference from previous approaches is that the surrounding background environment of the immersive video's virtual scene is progressively reproduced on the fly directly in the 3D space while the input stream is being rendered. To allow this, we propose a new 3D background modeling scheme that, based on GPU-accelerated real-time ray tracing, efficiently and incrementally builds the background model in compact 3D triangular mesh. Then, we demonstrate that the 3D background environment can effectively alleviate the critical disocclusion problem in the immersive rendering, eventually reducing spatial and temporal aliasing artifacts. It is also suggested that the 3D representation of background environment enables extension of the virtual environment of immersive video by interactively adding 3D visual effects during rendering.

Keywords Immersive video \cdot Real-time stereoscopic rendering \cdot Disocclusion \cdot 3D background model \cdot Ray tracing \cdot Virtual reality

Insung Ihm ihm@sogang.ac.kr

> Youngwook Kim kimyu7@sogang.ac.kr

Jehyeong Yun yjh1121@midasit.com

Joungil Yun sigipus@etri.re.kr

Sangwoon Kwak s.kwak@etri.re.kr

- ¹ Department of Computer Science and Engineering, Sogang University, Seoul, Republic of Korea
- ² CIM Dev. 1 Team, MIDAS Information Technology, Seongnam, Republic of Korea
- ³ Media Research Division, Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea

1 Introduction

In computer graphics and computer vision, the so-called immersive video in the multiview video-plus-depth format (Vadakital et al. 2022; Müller et al. 2011), originated from 3DTV and free-viewpoint video (Fehn 2004; Smolic 2011), has attracted considerable attention in recent years. This is partially due to its potential ability to create immersive experiences in virtual reality, augmented reality, and mixed reality environments (hereinafter collectively referred to as extended reality (XR) environments). Commonly available consumer-grade head-mounted displays (HMDs) are capable of tracking in real time the viewer's position and orientation in 3D space. These XR devices provide tools for a viewer to freely navigate the real or virtual 3D scene, captured in immersive video, with vivid immersive experiences.

The multiple depth-augmented reference views of immersive video have been rendered or synthesized for an arbitrary virtual view using the traditional depth-image-based rendering (DIBR) method (Fehn 2004). Although it will continue to play a key role in rendering of immersive video, the realtime task of synthesizing stereoscopic free-viewpoint images becomes more challenging in immersive environments. First, its computation must be fast enough to satisfy the high refresh rates of the HMDs (e.g.,, 80 Hz on the Oculus Rift S headset) to avoid any potential motion sickness. Efficient stereoscopic rendering is essential also because the display resolution of the recent XR headsets has become higher than ever. Second, the effective handling of the disocclusions that occur when a viewer moves away from reference views is one of the most critical issues in DIBR. In many XR applications, the viewer is basically allowed to move freely in a 3D workspace, leading to camera motions that are usually larger than typically assumed in the conventional 3D video systems. The disocclusion problem becomes worse with increasing viewing baselines between the virtual and reference cameras, often resulting in visually annoying artifacts from larger holes. This is the case in the situation assumed in the present work in which the disocclusions must be filled very quickly in real time during rendering. Third, when an XR application is developed for a streaming immersive video, it is highly desirable to represent the progressively increasing 3D geometrical information in the video in an appropriate form and make effective use of it. For example, the extracted 3D geometry may be utilized for implementing such important 3D graphics operations as collision detection and delivering a realistic haptic response, helping increase immersiveness in the virtual environment.

In this paper, we present a novel real-time framework for interactively rendering a streaming immersive video in an immersive XR environment. Specifically, we develop a virtual view synthesis system that, from a live immersive video stream, generates stereoscopic images in real time for presenting on a freely moving HMD. The main contribution of our work is to progressively reconstruct a 3D background model on the fly from an immersive video stream. The idea of building a background model to reduce disocclusions during virtual view synthesis has also been proposed (refer to Sect. 2 for discussion on related work). However, the previous approaches usually represented the extracted background information in the form of RGB or RGB-D images, which restricts their applicability to the development of 3D applications. In addition, due to the complexity of their algorithms, they often do not allow fast computations. By contrast, the proposed real-time method incrementally reconstructs 3D triangular meshes for probable background surfaces so that the created background model can immediately be applied in 3D graphics rendering pipelines for effective rendering of immersive video. In particular, our approach differs from previous work in that it utilizes the GPU-assisted real-time ray tracing technology (NVIDIA 2021) to efficiently and progressively refine the background model directly in 3D space.

Figure 1 overviews how effectively our approach of utilizing the 3D background model enhances the real-time rendering of live immersive video. The image (b) shows an OpenGL rendering of the 3D background environment that was progressively built in 3D triangular mesh up to the 176th video frame. In this test, a rather large movement of a virtual camera from the reference cameras was used, which is visible in the image (c) that was rendered for the left eye using only one closest reference camera. The two images (d) and (e) compare the 3D warping results produced using five closest reference cameras without and with an application of the 3D background environment built thus far. Clearly, the disoccluded regions were diminished significantly thanks to the use of the 3D background model.

When the holes were finally filled using a real-time disocclusion filling method, the eventual effect of the presented 3D background modeling scheme is clearly observed through the corresponding two images (f) and (g) that were produced respectively from the previous two images. The last image (h) illustrates how easily such an interesting projector effect can be created interactively with the 3D background model during rendering, effectively augmenting the virtual environment in the video stream.

Using experiments with several immersive videos, we show that the proposed 3D background model can significantly reduce disocclusions during rendering before applying an inherently incomplete real-time hole-filling algorithm. This is particularly important in developing XR applications because it allows the use of more ground-truth information inherent in immersive video, thereby eventually suppressing visually annoying artifacts in the final rendering despite relatively large movements of a viewer in the 3D workspace. In addition to the enhancement of the rendering quality, we also show that the use of the 3D background model enables us to feed only a few reference views into our real-time virtual view synthesis pipeline, significantly reducing the computational burden for nontrivial immersive videos and highresolution HMDs. Finally, we suggest through an example how effectively the triangular mesh representation of the 3D background environment may easily create interesting 3D effects in immersive environments.

2 Previous work

First of all, this work focuses on the disocclusion-reduced, real-time rendering of immersive videos in the multiview plus depth video representation which is the main target format of the MPEG Immersive Video Standard (Vadakital et al. 2022). Therefore, other video representation and rendering strategies such as the multiplane image (e.g., Zhou et al. (2018); Flynn et al. (2019)) and the multiplane

Fig. 1 Rendering of live immersive video using a 3D background model. The RGB-D views of this *Hijack* sequence (Doré 2018) were generated by 10 fixed reference cameras at 30 Hz where each view had 2048 × 2048 pixels



(a) A VR user in action



(b) 3D background model





(c) Current virtual view



(d) After 3D warping 1



(e) After 3D warping 2



(f) Final rendering 1





(g) Final rendering 2

(h) 3D projector effect

image (e.g.,, Broxton et al. (2020)) are outside the scope of this work.

Immersive video, which is rooted in such visual media as 3DTV, 3D video, and free-viewpoint video (Müller et al. 2011; Smolic 2011), effectively provides a viewer with a 3D depth impression on captured sceneries. To render or synthesize arbitrary virtual views from multiple depth-augmented reference views of immersive video, the DIBR scheme (Fehn 2004; Zhu et al. 2013) has conventionally been applied, in which 3D image warping (or simply 3D warping) (Mark et al. 1997) and disocclusion filling (or hole filling) are two essential components. In particular, the ultimate goal of disocclusion filling is to reproduce the ground-truth views against the missing image regions. It is, however, difficult or often impossible to accomplish due to incomplete information from input video streams.

To achieve faithful view synthesis results, it is important to effectively exploit the inherent information that exists spatially and temporally in the captured 3D scenes. In the spatial domain, since the introduction of the exemplar-based inpainting technique by Criminisi et al. (2004), a variety of image-based and geometry-based approaches for recovering the disoccluded regions have been proposed. On the other hand, temporal correlation between different time frames has been exploited to acquire more in-depth insights into the disoccluded regions and also alleviate the temporal flickering artifacts in the view synthesis results. For instance, in Huang and Zhang (2008), the uncovered background information was extracted on the fly from temporally neighboring frames through a disparity analysis. Motion vectors were also approximated from preceding frames in Chen et al. (2010) to fill disoccluded regions in the virtual view.

A notable approach in the temporal domain is to incrementally model or reconstruct the background of the scene from temporally neighboring frames, onto which foreground elements are composited. Such background information allows the use of more ground-truth data for synthesizing the background area for a virtual view, thereby reducing the disoccluded regions effectively. Criminisi et al. (2007) represented a temporal background model in terms of a disparity map and two color images for synthesizing virtual views from synchronized stereo video streams. A background image was incrementally updated from streaming 3D videos for a pair of static cameras in Schmeing and Jiang (2010), where pixels that are determined to depict static objects contributed to the background image. A temporally updated sprite model was used by Ndjiki-Nya et al. (2011) to store the background pixels with depth values below a contentadaptive threshold. A background update method with scene adaptivity was also presented by Sun et al. (2012), in which the switchable Gaussian model improved both accuracy and efficiency in building a background model. In addition, to enhance the stability of the background update process, Yao et al. (2014) proposed another probabilistic technique that combines the Gaussian mixture model (GMM) and foreground depth correlation. Then, Rahaman and Paul (2018) presented an improved GMM method to separate background and foreground pixels. Luo et al. (2020) compared three different background reconstruction models including their own modified GMM method (Luo et al. 2016) in a general framework that enables moving cameras.

While effective, these previous approaches reproduced the background information basically in image space, which limited the applicability of their methods to the development of immersive 3D applications. Moreover, due to the complexities of their algorithms, real-time processing was hardly possible. Recently, an interesting approach was proposed by Lee et al. (2021), where the 3D surfaces of the scene background were constructed from RGB-D video streams in terms of a point cloud in 3D world space. This 3D geometry was efficiently processed in the current 3D real-time rendering pipeline to markedly alleviate the disocclusion problem. However, their scheme needed considerable preprocessing time to build the 3D geometry, making it difficult to apply to live immersive video streams.

3 Our method

3.1 Problem specification

The proposed real-time stereoscopic virtual view synthesis system takes two asynchronous input streams in parallel. The first is an immersive video stream that is produced by multiple calibrated and synchronized reference cameras, capturing a given 3D scene at the video input rate of $f_{\rm vin}$ fps. The second is an independent stream of six degrees-of-freedom (6DoF) poses of a virtual stereo camera that moves freely in a 3D workspace defined around the reference cameras. Then, the problem to solve is how to effectively render virtual stereoscopic views on the fly against these two input streams.

In the present work, our rendering scheme targets the dimension of immersive video that amounts to the nontrivial test sequences provided in the MPEG Test Model for Immersive Video (TMIV) dataset (Boyce et al. 2021), in which each time frame consists of 10–25 reference views of 2–8 megapixels in RGB-D format each.

To achieve high efficiency in rendering, it is often required to restrict the number of reference views of high resolution that participate in the actual virtual view synthesis process. Therefore, our renderer dynamically selects n_{rev}^{render} views from the entire reference views of the current time frame whose field of view overlaps the most with that of the virtual camera. Although the selection algorithm must be adapted to the specific camera constellation structure, it was often enough for the tested immersive videos to choose the reference cameras whose view directions are the most similar (in terms of inner product) to that of the virtual camera. Using fewer cameras, however, would aggravate the disocclusion problem further as less geometry information is utilized in the rendering process (see Fig. 2). To ease this problem, we apply the proposed 3D background model to prevent disocclusions from occurring as much as possible before filling them later in a rather incomplete manner.



Fig. 2 Occurrence of disocclusions. When only two reference cameras RC_7 and RC_8 are chosen for view synthesis, the surfaces marked in red curve become uncovered against the virtual camera VC. The unselected reference view of RC_6 of the current time frame would provide surface information on some part of the hole regions, or some of them may be visible by one of the selected cameras in a past video frame before a moving foreground object occluded them. The holes would decrease significantly if we could use all the reference views captured up to the current time by the entire reference cameras

3.2 Overview of our virtual view synthesis system

Figure 3 illustrates the computational pipeline of the presented real-time rendering system that is performed entirely on the GPU. In the back end (3D Background Model Update), a double buffering framework is employed to perform data acquisition and rendering in parallel where the front buffer, swapped every $\frac{1}{f_{vin}}$ second, always holds current views streaming from $n_{\rm rc}$ reference cameras (Update Current Reference Views). Before the RGB-D images in the front buffer are used for view synthesis, they are preprocessed to update a 3D geometry structure called the probable background surface (PBS) (Update PBS). This is our 3D background model that aims to decrease disoccluded regions in the 3D warped images. This geometry structure is then rendered with respect to an imaginary camera whose view volume tightly covers the current view volumes of the virtual stereo camera of the XR headset, producing an RGB-D image called the probable background image (PBI) (Render *PBI*). Once the new PBS and PBI are prepared, the front buffer is ready for use in rendering for the next time period.

Meanwhile, the front end (*Stereoscopic Image Rendering*), repeatedly synthesizes stereoscopic virtual views against the current 6DoF pose of a viewer. Each of the left and right views are rendered basically using the conventional 3D warping (*Perform 3D Warping*) followed by a hole inpainting process (*Fill Disocclusions*). In this computational pipeline, triangular meshes are first reconstructed in 3D world space from dynamically selected reference views (*Select Subset of Ref. Cameras*) using depth values, and then projected to each view plane of the stereoscopic virtual camera. The resulting stereo images are then sent to the virtual camera for immersive display (*Present Stereo Images on HMD*).

Two major issues in developing our rendering system were (i) how to progressively extract and accumulate the background environment information from a streaming immersive video (via the PBS), and (ii) how to efficiently exploit the incremental PBS to prevent holes appearing during rendering as much as possible (via the PBI). In the next



Fig. 3 Real-time stereoscopic rendering pipeline. A video frame, made of $n_{\rm rc}$ reference views in RGB-D format, is repeatedly produced at the rate of $f_{\rm vin}$ fps. In parallel, responding to a freely moving XR headset with display frame rate of $f_{\rm ren}$ fps, stereoscopic virtual views are rendered using (partially selected) reference views of the current video frame. The main contribution of our work, highlighted in red, is to incrementally build a triangular mesh, called the *probable back*-

ground surface (PBS) and exploit it for generating an RGB-D image, called the *probable background image* (PBI) in every video frame. Thanks to the backdrop image, created from the PBI in every rendering frame, the disocclusions that occur in the 3D warping stage decrease significantly, eventually leading to markedly improved rendering results. To allow fast computation for handling the PBS and PBI, we exploit the ray tracing hardware provided by the latest GPUs

two subsections, we discuss our solutions in detail. Note that the presented rendering pipeline is implemented primarily on the CUDA platform while the OpenGL rendering system is exploited for optimized 3D warping and disocclusion filling. Notably, the ray tracing operations, which are essential for real-time manipulation of the PBS and PBI, are accelerated using the NVIDA OptiX GPU ray tracing engine (NVIDIA 2021). (This work exploits the ray tracing capability provided by the recent GPUs for fast computation. However, it should be noted that the ray tracing operations employed in the presented algorithms can equally be implemented using the CUDA capability).

3.3 Progressive update of 3D background environment

For now, we only consider static reference cameras. Our method is then extended to also handle moving reference cameras in Sect. 4.2.

3.3.1 Hybrid structure for PBS

Although possibly incomplete, the triangular mesh of the PBS aims to represent the surrounding background of the 3D scene captured by the input immersive video stream received up to the current time frame. In our scheme, this is made up of two submeshes: the first, denoted by S-PBS ('S' in static), is built once in the initialization stage from the reference views of the first input frame, and remains unchanged during the entire period of video streaming. The second, denoted by P-PBS ('P' in progressive), is dynamically updated for each video input frame, thereby progressively refining the background information with the incoming reference views. These two triangular meshes, respectively stored in bounding volume hierarchy (BVH), constitute an instance acceleration structure (IAS) (NVIDIA 2021) on the NVIDIA OptiX ray tracing framework that allows effective GPUassisted ray tracing against immersive video data.

3.3.2 Initialization of PBS with S-PBS

The function of the S-PBS is to offer a base triangular mesh for the surrounding 3D environment of a given scene, which is employed not only for view synthesis but also for updating the P-PBS subsequently. It is built in two steps by selecting proper triangles from those created by the input reference views of the first time frame. First, triangles are gathered in the world space by back-projecting the 2D triangular grid of the RGB-D image of each reference view based on its intrinsic and extrinsic parameters (see Fig. 4 for the triangle generation).

Note that the view frustums of the reference cameras usually overlap with each other, often making this process



Fig. 4 Generation of triangles from reference views. The depth values enable to generate triangular meshes in 3D world space by back-projecting the 2D triangles in the pixel grid, where each pixel is responsible for two triangles in the northeastern quadrant. The invalid, long and thin, back-projected triangles along the borders of objects are culled in OpenGL geometry shader based on the depths of incident vertices

produce highly redundant triangles around the same surface areas. In order to prevent the size of the resulting S-PBS from growing excessively, we usually choose a subset of reference cameras of size n_{rv}^{init} whose views overlap as little as possible while covering the entire view space. The camera selection procedure is done as follows. Starting from an arbitrarily selected reference camera, the 2D triangular grids of the cameras selected thus far are 3D warped respectively to the remaining reference camera spaces, and one with the smallest number of projected pixels are chosen next. This process is repeated until all needed reference cameras are selected.

Even with such selection of reference cameras, however, multiple triangles are still often mapped to the same surface regions if they are collected carelessly. To decrease such redundancy further, we add a triangle to the collection only if it represents a surface area farther than those of previously added triangles. This computation can easily be parallelized on the CUDA platform by first transforming each triangle from the current reference view to the respective image spaces of previously processed reference views, and accepting it only if the transformed depths of the three vertices are sufficiently greater than those of the corresponding pixels.

Now, the collected triangles largely represent the surrounding background surfaces in the 3D scene although some of them may be from foreground objects. Culling such foreground triangles from the S-PBS is important because the foreground objects may move unpredictably as time proceeds, possibly causing a ghost-like effect in view synthesis results. Therefore, we take another step, namely *foreground culling*, to remove the foreground triangles as much as possible. In this procedure, the gathered triangle set is first organized in a BVH on the NVIDIA OptiX framework for efficient ray tracing against them. Then, the participating reference cameras are visited one by one, casting two rays through each pixel of their view planes. (The two rays are generated by slightly offsetting the pixel center toward the two 2D triangles for which the pixel is responsible. See Fig. 4.) Here, in contrast to standard ray tracing, each ray is traced in an opposite direction starting from a location outside the scene's bounding volume, marking the closest-hit triangle, if any, as valid. Then, all triangles except the valid ones are culled from the triangle collection (refer to Fig. 5).

While the above foreground culling algorithm catches most of the background triangles from the triangle collection, it often misses some due to error-prone depth values of immersive video. To cope with this numerical problem, another complementary ray is cast from just before each closest-hit point to search for any missing background triangles in the vicinity of the closest-hit point (refer to Fig. 6a for details on this extra computation). After the follow-up ray tracing is finished, all valid triangles are compacted and reorganized in a BVH for the S-PBS. Then, together with a null P-PBS, it forms the IAS of the initial PBS. This is our 3D representation of the background environment of



Fig. 5 Foreground culling. For each participating reference camera, rays are cast from outside the scene toward the pixels of its view plane, marking the closest-hit triangles as valid. When all reference cameras are processed, the unmarked, foreground triangles are removed from the initial collection. For example, the two rays R_0 and R_1 , generated by the reference camera RC_p , successfully cull the foreground triangles T_0 to T_3 . Due to imperfect information from the input immersive video stream, however, some rays, e.g., R_2 and R_3 , fail to identify foreground triangles T_4 and T_5 , often making them remain after the foreground regions become uncovered by some reference cameras



(a) Complementary rays for fore- (b) Triangle filtering ground culling

Fig. 6 Two ray tracing operations for manipulating the PBS. a Ideally, the background triangles generated from a reference view must be hit by the rays cast by its camera during the foreground culling process. For example, the triangle T_{a} must be hit by the ray R_{a} cast by the reference camera RC_{q} . However, due to imprecise depth values of immersive video, a wrong triangle (e.g.,, T_{p}) is often hit by R_{q} , leading to visually annoying, irregular holes on the surfaces of background geometry. To handle such z-fighting problem, after a closesthit point is found by a ray, we move the point back a short distance $\frac{\delta}{2}$, and cast another complementary ray along the same direction. Then, all triangles hit in the short interval $[0, \delta]$ are additionally marked as valid. This follow-up ray casting can easily be implemented using the any-hit program of the recent ray tracing engines such as NVIDIA OptiX. **b** When new triangles (e.g.,, T_p and T_q) are to be added in the current 3D background model, a ray is first cast from the reference camera through the center of each of them and intersected with the current PBS, represented by blue triangles. Then, a candidate triangle is added in the current PBS via the P-PBS update only when it is more than distance δ behind the closest-hit point (e.g., T_{p}). Otherwise, it is simply removed from consideration (e.g.,, T_q). This ensures that only necessary triangles are to be added to the 3D background model

the scene observed by the reference cameras in the first time frame.

3.3.3 Update of the PBS through P-PBS

Once the PBS is initialized, our system starts processing the immersive video stream. Given input reference views of the current time frame, if there has been a movement of a foreground object since the last time frame, parts of surfaces that were not originally visible may become disoccluded in some relevant reference views. The newly visible farther surfaces can be used to improve the PBS either by extending the surfaces of the surrounding 3D background or by culling existing foreground surfaces therefrom. However, restructuring the entire PBS every video frame to reflect small differences between successive time frames is not feasible because the large size of the S-PBS component would hinder fast update. Therefore, our method utilizes another BVH of (usually) small size, i.e., the P-PBS, to process the differential changes that occur over time.

For efficient implementation, each reference camera is associated with an RGB-D image of the same resolution, called the maximum depth image. Each pixel of this image, initially set to that of the reference view of the first time frame, stores the color and depth to the farthest surface observed through the pixel from the camera up to now. For every video input frame, each pixel of the maximum depth image is modified with the color and depth of the current reference view only if the new depth is farther by a nontrivial amount. The modified pixels indicate the regions in the reference view where farther surfaces become disoccluded. Such newly visible surfaces can be reconstructed in the 3D world space by generating triangles through the back-projection from the 2D triangles with at least one updated pixel (refer to Fig. 4 again). Note that some of the new triangles may represent background surfaces that are already represented by the current PBS. Thus, to prevent the size of the PBS from growing excessively, the unnecessary triangles are filtered out as described in Fig. 6b. The cost of this additional ray tracing operation is quite low because the number of newly visible triangles in each time frame is rather limited.

Now, after all the triangles of the disoccluded surfaces are gathered from the current reference views, they are merged with those in the current P-PBS, creating a temporary P-PBS in the BVH format. Then, the ray tracing-based foreground-culling procedure is carried out on the NVIDIA OptiX framework with respect to reference cameras using the IAS made of the S-PBS and the temporary P-PBS. As noted, we do not restructure the S-PBS part of the PBS due to its large size. Instead, the triangles in the S-PBS that are found to be foreground triangles are simply marked as invalid in the triangle list so that they are simply ignored during ray tracing in the later stage. Meanwhile, foreground triangles are culled from the triangle list of the P-PBS, and the survived triangles in the P-PBS are reorganized in a P-PBS again. Then, the new P-PBS replaces the old P-PBS part in the IAS, completing the PBS update (refer to the *Update PBS* component in Fig. 3).

Finally, when there are a rather large number of reference cameras in the system, updating the PBS with all of the high-resolution reference views in each video input frame would slow down the entire rendering process. In this case, we partition the reference views into groups of acceptable size of n_{rv}^{update} , and use one group for each PBS update in a circular manner. (The workload of our renderer could also be reduced by performing the PBS update every few frames.) Note that even if a reference view does not participate in the PBS update of the current time frame, its maximum depth image is still updated in the background so that newly disoccluded surfaces are eventually considered in the PBS update in a later time frame (see Fig. 7). Refer to Fig. 8 to take a look at the examples of the 3D background model built during rendering.

3.4 Rendering with 3D background environment

3.4.1 Generation of PBI from PBS

The main usage of the PBS is to help effectively create backdrop images during rendering, on which virtual views are generated with reduced holes. The PBS could be used directly to render the background images for current stereoscopic virtual cameras. However, it is not very efficient to render the BVH data structure, which is often very large in size and managed by the NVIDIA OptiX platform, on the OpenGL system for every rendering frame. (Note that we



(a) After 0th frame

(b) After 20th frame

(c) After 40th frame

Fig. 7 Possible lag in the PBS update (Hijack). In this test, only one reference frame out of 10 was used alternatively in the PBS update every video frame. This implies that there might be some delay in reflecting newly uncovered farther surfaces in the PBS. When the woman (marked by an arrow) started moving after the 0th frame (a),

some parts of her body (marked by a circle) remained in the rendering images for a short period of some video frames (**b**). However, they disappeared quickly as their triangles were removed eventually during the PBS update a few frames later (**c**). In general, it was difficult to notice such ghost-like effect on the tested HMD Fig. 8 Progressive update of the PBS (*Museum*). The OpenGL renderings in **a** and **b** display the PBS in the virtual 3D space. As foreground persons move over time, either newly visible farther surfaces are added or falsely existing foreground surfaces are removed. The image **c** shows the actual triangular meshes in the PBS



(a) After 0th frame

(b) After 100th frame (c) After 100th frame

assume the frame rate of demanded rendering to be higher than that of video input.) Therefore, we generate an intermediate RGB-D image, i.e., the PBI, from the updated PBS once per every video input frame, transfer it to the OpenGL pipeline as a light 2D texture, and use the 3D background information for creating the backdrop images during rendering that occurs for the next video input period.

To produce the PBI, we place in the virtual 3D space an imaginary camera just behind the current virtual camera of the XR headset. The view frustum of the imaginary camera is set up to be slightly larger than those of the stereoscopic view frustums of the virtual camera in order to consider a possible movement of the viewer until the next PBI update. Then, ray tracing is performed against the PBS by casting rays from outside the 3D scene toward the view plane, finding the farthest hits seen from the imaginary camera. The resulting PBI is then a compact representation of the 3D background information that is actually needed for synthesizing virtual views for the next video input period. Once the PBI rendering is finished (Render PBI in Fig. 3), the entire update for the current video frame is completed in the CUDA and OptiX platforms, and the current reference views and the PBI are handed to the OpenGL pipeline for rendering.

3.4.2 Creation of backdrop images with PBI

Now, the actual rendering begins against the current 6DoF pose of the viewer, whose procedure is basically the same as the conventional virtual view synthesis except for the use of the 3D background information compacted in the PBI. In the initial step, the RGB-D image of the PBI is 3D warped to each view plane of the stereoscopic virtual camera to create the respective backdrop images (*Render Backdrops* in Fig. 3). Then, a selected set of current reference views are 3D warped onto these background images. Note that the

z-fighting problem can occur during the 3D warping process. Therefore, the depth values of the original backdrop images are offset in the framebuffer so that the visible surfaces are moved slightly backward.

As noted, the use of the PBI and the backdrops resulting therefrom enables us to reduce the occurrence of disocclusion holes, which significantly enhances the quality of rendering images. Just as importantly, it also allows for the use of fewer reference views even with a nontrivial baseline between the virtual and reference cameras, which markedly decreases the computational burden on the GPU.

3.4.3 Fast disocclusion filling by inpainting

In the final stage of rendering, the disocclusions that occur despite the use of the backdrop images need to be filled by a fast inpainting algorithm (*Fill Disocclusions* in Fig. 3). Because the aim of this work is focused on reducing occurrence of disocclusions during virtual view synthesis, we apply a simple pixel-based technique for hole filling that is well suited when the sizes of holes are rather small, as in the case of our renderer. More specifically, an OpenGL fragment shader is run on the image area in which the color of each fragment in a hole area is blended from the colors of neighboring pixels in nonhole areas, with those having farther depths more weighted.

Note that numerous approaches, ranging from the traditional patch- or diffusion-based to the newer deep learningbased, have been reported in the literature for image inpainting (Elharrouss et al. 2020; Jam et al. 2021; Qin et al. 2021). Although these methods usually outperform our simple hole filling algorithm, they are often inappropriate for use in our real-time immersive video renderer because their complexities are far beyond the computational budget allowed in this last stage. Finding an appropriate one for the presented immersive rendering is left as a future work.

4 Experimental results

To evaluate the effectiveness of the 3D background model, we implemented the presented real-time stereoscopic view synthesis pipeline in Fig. 3 on a PC with dual NVIDIA GeForce RTX 3090 GPUs that provide dedicated ray tracing hardware. While rendering, our system generated stereo images of 1280×1440 pixels per eye to display on an Oculus Rift S headset with 80 Hz refresh rate.

4.1 Test for scenarios of fixed reference cameras

Table 1 summarizes the video sequences from the TMIV (Boyce et al. 2021) that were used for performance evaluation. All these sequences were captured with fixed reference cameras. For the *Hijack* sequence, we used a down-sampled version of the original data that had 4096×4096 pixels per reference view. Note that the depth values of the *Painter* and *Frog* sequences were estimated on the respective natural contents with an image-based depth estimation method, and were thereby contaminated by noises.

In the presented experiment, the five nearest reference cameras were dynamically selected for rendering ($n_{rv}^{render} = 5$ in Sect. 3.1). Here, a rather large baseline between the virtual viewer and the reference cameras was tested to see the effect of the proposed 3D background model. Then, six reference views were used to initially build the S-PBS ($n_{rv}^{init} = 6$ in Sect. 3.3.2). One reference view was selected for updating the P-PBS in a circular manner in each video input frame ($n_{rv}^{update} = 1$ in Sect. 3.3.3). In addition, the threshold δ , used in the foreground culling process (see Fig. 6 again), was set to 0.2. This corresponds to 0.2 m as the depth ranges are [0.5 m, 25 m] (*Hijack* and *Museum*), [1 m, 10 m] (*Painter*), and [0.3 m, 1.62 m] (*Frog*).

We first compared the presented technique with the conventional DIBR approach that simply uses all current reference views for virtual view synthesis. Figure 9 demonstrates how effectively the disocclusion artifacts in the 3D warped images were decreased with an application of the 3D background model. Then, the timing results in Table 2 show that the use of the 3D background model significantly reduces



Fig. 9 Comparison of 3D warping results with the conventional DIBR approach. These images were created for the left eye in the rendering frame just after the 299th frame of video input. **a** The traditional 3D warping method was first tested against all reference views of the video frame. **b** The 3D warping process was extended with the 3D background model scheme and tested using only five selected views. Thanks to the 3D data structure, the disocclusion artifacts usually decreased noticeably, although our method used fewer reference views. Despite the overhead of handling the 3D background model, our method turned out to be markedly faster

Table 1Test immersive videoscreated from fixed referencecameras

Dataset	Views	Resolution	Projection	Content	Ref
Museum	24	2048×2048	Omnidirectional	CG	Doré (2018)
ацаск Painter	10 16	2048×2048 2048×1088	Perspective	Natural	Dore (2018) Doyen et al. (2017)
Frog	13	1920×1080	Perspective	Natural	Salahieh et al. (2018)

Museum and *Hijack* are computer-generated contents with close-to-perfect depth maps. By contrast, *Painter* and *Frog* are natural contents with estimated depth. All these videos were generated at the rate of 30 Hz. Refer to Jung and Kroon (2022) for the details of these sequences

Table 2 Comparison of stereoscopic view synthesis times with the conventional DIBR approach

3.30)								
1.86)								
(b) Final rendering with disocclusion filling								
5.60)								
2.22)								

For each test case, we provide two rendering times in milliseconds. The first is the pure computing time taken when the Oculus headset was not synced. The second in parentheses was taken when the headset was synced, in which the frame rate slowed down markedly due to the synchronization overhead. These results indicate that our method using five reference views performed rendering more efficiently with fewer disocclusion artifacts. Equally importantly, it progressively reconstructs the 3D background environment in triangular mesh from live immersive video stream, enabling the creation of 3D effects in immersive environments

the view synthesis time despite the overhead of manipulating the 3D data structure. Note that, for the scene with only fixed reference cameras, the effect of applying the 3D background model becomes greater when foreground objects move extensively. In the *Museum* scene, however, the people move only slightly in their places, thereby preventing new background surfaces from being added to the 3D background model. In this scene, the conventional approach using all reference views may result in fewer disocclusion artifacts. Even in such a case, our method using only five views was able to achieve much faster frame rates that are appropriate for stereoscopic display on the VR headset while reducing the hole areas as much as possible using the 3D background model.

Figures 1 and 10 demonstrate how effectively the 3D background model enhances the view synthesis process when only five closest reference views were selected for rendering. The backdrops, generated using our 3D background model in each rendering frame, greatly helped mitigate the occurrence of disocclusion holes in the 3D warping stage. This means that the areas in the rendering image that had to be filled by the subsequent real-time inpainting process were reduced significantly, eventually leading to improved rendering results.

Figure 11 then analyzes the computational cost for our renderer. The graphs in (a) and (b) reveal the overheads incurred when including the 3D background model scheme in the conventional virtual view synthesis pipeline (Museum, Hijack, Painter, and Frog from top to bottom). In the beginning, it took 1.27, 0.33, 0.26, and 0.30 s to build the initial S-PBS structures that contained 32.82, 9.05, 6.30, and 5.39 million triangles, respectively. Then, each pair of curves in (a) indicates, in millions, the size of newly exposed triangles simply accumulated without the foreground culling and the actual size of the P-PBS resulted from the foreground culling, respectively. For the Museum sequence, the two curves were almost the same. This was because almost all newly exposed surfaces actually turned out to be background surfaces, with only a few triangles removed by the foreground culling process. For Museum and Hijack with reliable depth values, reasonable numbers of triangles were newly uncovered in every

Fig. 10 Effect of 3D background model (3DBM) in virtual view synthesis. These images were generated for the left eye in the immersive rendering frames just after the 260th (Museum), 249th (Painter), and 300th (Frog) frames of video input



(no 3DBM)

(no 3DBM)

(b) 3D warping (ours)

🖉 Springer

(ours)



Fig. 11 Computational cost. The graphs in \mathbf{a} and \mathbf{b} indicate the spatial and temporal overheads of handling the proposed 3D background model. On the other hand, the graphs in \mathbf{c} compare the total stereoscopic rendering times. Please see the text for details

video input frame. By contrast, for *Painter* and *Frog* with estimated depths, more new triangles were involved in the P-PBS update, mainly due to the noise in the depth map. In both cases, however, our culling method was able to effectively remove foreground surfaces from the 3D background environment as expected. As shown in (b), this led to the PBS update time that stably fell below one time interval of video frames (1/30 sec).

On the other hand, the graphs in (c) presents the time for the entire stereoscopic rendering, where two adjacent columns indicate the times respectively taken when the sync on the Oculus headset was turned on and off. Here, the two pairs A & B and C & D compare the average rendering times spent when five closest reference cameras were used without and with an application of the 3D background model, respectively. The two in each pair respectively represent the times consumed without and with an application of disocclusion filling, indicating that only a little extra time was needed for the applied real-time hole-filling operation. Overall, we could almost achieve the highest possible frame rate on the tested HMD (80 Hz = 12.5 ms) although the GPU resources had to be shared between the PBS update and rendering tasks.

Finally, when three reference views were used with the 3D background model (E & F), the average rendering time decreased slightly. Depending on the camera constellation, the use of the 3D background model often produced the rendering quality that almost matches that of the five view case.

Note that a large portion of the PBS update time was spent on rebuilding the P-PBS into a BVH after the foreground culling, which had to be carried out through the time-consuming *acceleration structure build* operation of the OptiX ray tracing engine. In our implementation, we split the P-PBS into multiple BVHs, one per reference camera, where the update task was performed to the corresponding tree. Although this might slightly slow down the ray tracing performance due to a possible deterioration in the BVH quality, it effectively prevented an excessive delay in the rebuild operation. As a result, the PBS update time was shown to stably fall below one time interval of video frames for the four test sequences. This implies that, at most, one time interval lag was incurred in the video input, which was in fact hardly noticeable.

4.2 Extension for moving reference cameras

So far, we have only considered static reference cameras. To allow for a dynamic movement of reference camera during the generation of immersive video, the P-PBS update process described in Sect. 3.3.3 needs to be extended slightly. When a reference camera has moved since the last frame, its maximum depth image is first reset with the depth of the current reference view. Then, we examine if there are newly visible surfaces in the current view that can be added to the 3D background model as follows. Each pixel of the current view is back-projected to a 3D surface point in the world space, which is then projected onto the 2D image plane of the previous time frame. Then, using the depth of the maximum depth image of the previous frame, we check if the projected 2D pixel sees the same 3D surface point as the current pixel. If this is the case, no further processing is necessary (this occurs for most pixels because the movements of the camera and/or objects between two sequential frames are differential).

If this is not the case, it is possible that the current pixel sees a new surface point that may augment the current 3D background model. To establish this, a ray is cast through the pixel of the current view, locating the farthest hit against the current PBS. (Note that this farthest hit is the surface point observed through the current pixel in the background environment built thus far.) Only if the depth of the current pixel is greater than the distance to the farthest hit, it is marked as newly visible. After all the pixels in the current view are examined, the triangles reconstructed via back-projection from the marked pixels are merged with those in the current P-PBS, from which the subsequent foreground culling is carried out to build the final P-PBS (refer to Sect. 3.3.3 again).

To evaluate the extended P-PBS update algorithm, we synthesized a video sequence, *Gymnasium-a*, made of 250 frames using four fixed and one moving reference cameras (see Fig. 12a). Each of the omnidirectional cameras produced an RGB-D image of 2048×2048 pixels per reference view. During visualizing the immersive video, we used two dynamically selected fixed cameras for rendering ($n_{rv}^{render} = 2$), four fixed cameras for initially building the S-PBS ($n_{rv}^{init} = 4$), and 0.2 m for the foreground culling ($\delta = 0.2$). Then, one reference view was selected from the participating cameras for the P-PBS update in a circular manner in each video input frame ($n_{rv}^{update} = 1$). Here, two views were selected for rendering from the four fixed cameras whereas the views from the moving camera were used only for the P-PBS update.

When the moving camera was used, large numbers of triangles were newly exposed by the moving camera during every period of five video input frames as indicated by the blue curve in Fig. 12b. (The moving camera was selected every fifth frame among the five participating cameras for the P-PBS update as n_{rv}^{update} was set to 1.) In fact, large portions of such triangles actually corresponded to foreground surfaces with respect to some fixed reference cameras. Our foreground culling method was very effective in preventing such triangles from being added to the P-PBS as implied

Fig. 12 Enabling a moving reference camera for immersive video. a In this video sequence. four fixed reference cameras were located near the center of the 3D scene looking around in the four orthogonal directions. In addition, one reference camera moved linearly in the air looking down the gymnasium. b When the moving camera was used, the presented foreground culling method was very effective in keeping the size of the P-PBS from growing excessively



by the orange curve. This eventually suppressed the excessive (and unnecessary) growth of the 3D background model, allowing us to keep the PBS update time stably under 1/30 sec.

In our experiment, we tested two situations where the reference views from the moving camera were not utilized and utilized for rendering the *Gymnasium-a* video (refer to Fig. 13a, b). As clearly seen, the effect of using the moving camera was substantial. It continuously scanned the 3D scene from a different viewpoint to those of the fixed reference cameras. These extra views allowed us to effectively augment the 3D background model, leading to 3D warping with markedly reduced hole areas. When only the fixed cameras were used for rendering (a), considerable holes remained even after the 249th input frame despite the use of the 3D background model. With the extra moving camera overlooking the scene (b), the disoccluded regions effectively diminished particularly as time proceeded. On

average, it took 3.51 ms (13.22 ms) and 4.6 ms (15.57 ms) for the entire rendering without and with the additional moving camera. (Again, the figures in parentheses indicate timings taken when the Oculus headset was synced.)

To further test the presented P-PBS update algorithm against the dynamic scene, we generated another video sequence, *Gymnasium-b*, where the *Gymnasium-a* sequence was slightly modified in such a way that a male runner moves faster than others. Interestingly enough, the effect of using the 3D background model was more observable around the fast runner as can be identified in Fig. 13c. That was because, around the fast moving object, larger surface regions often became newly visible more quickly to a reference camera, whether static or dynamic, in each frame. As a result, those newly visible surfaces, added to the 3D background model, reduced the hole areas more effectively.

Before we wrap up, it should be noted that the sizes of the 3D scene spaces of the *Gymnasium* and *Museum* sequences are similar to each other, whereas five and

Fig. 13 Effect of using a moving reference camera for immersive video. The images in (a) and (b) show the 3D warping results produced with the presented 3D background model for two different virtual views just after the 4th and 249th frames of video input. Unlike as in **a**, the moving reference camera that was added during generating the immersive video **b** allowed to effectively reduce the disoccluded regions (marked in light blue) during 3D warping. Each pair of images in c compare the 3D warping results generated without and with the 3D background model. They suggest that the effect of employing the 3D background model in this dynamic scene can be stronger around a fast moving object as larger surface areas may become newly visible to a reference camera per each frame, leading to a faster buildup of the 3D background model. Please watch the attached videos



After 4th frame After 249th frame (a) After 3D warping with the fixed cameras only (*Gymnasium-a*)







After 4th frameAfter 249th frame(b) After 3D warping with the extra moving camera (Gymnasium-a)







After 143rd frameAfter 170th frame(c) After 3D warping with the extra moving camera (Gymnasium-b)

Table 3 Immersive rendering performance on three selected **GPUs**

	DIBR		Ours		
	Single GPU	Dual GPUs	Single GPU	Dual GPUs	
3090 (2020)	30.36 (46.32)	16.28 (33.59)	8.93 (15.31)	5.13 (13.13)	
A6000 (2020)	31.46 (41.03)	16.57 (27.06)	9.10 (14.14)	5.51 (12.79)	
4090 (2022)	16.26 (25.00)	-	4.33 (12.50)	_	

To further investigate the computational burden for including the 3D background model in the rendering of immersive video, we tested with three recent NVIDIA GPUs supporting hardware-accelerated ray tracing for synthesizing stereoscopic views for the Museum sequence. Here, '3090' and '4090' represent the NVIDIA GeForce RTX 3090 and 4090 GPUs, respectively, while 'A6000' represents the NVIDIA Quadro RTX A6000 GPU. (Please refer to Table 2 again to understand what these figures in milliseconds indicate).

24 reference cameras were employed for generating the respective videos. This test suggests that a moving reference camera can be a useful tool for reducing the size of immersive video as it significantly decreases the demand for fixed cameras even at improved rendering quality.

_ 3

5 Concluding remarks

In this paper, we have demonstrated that the progressively refined 3D background model that is built on the fly in the virtual 3D space of immersive video enables us to effectively exploit more ground-truth image data in the virtual view synthesis process. This eased the burden of the inevitably incomplete real-time hole-filling task, thereby significantly alleviating both spatial and temporal aliasing artifacts in the final immersive rendering. Thanks to the 3D background model, we were able to produce improved view synthesis results using a subset of input reference views in each time frame, which is important when a virtual view is to be synthesized from reference views of high resolution. Unlike the previous view synthesis methods that usually assumed static reference cameras only, e.g., Vadakital et al. (2022), we demonstrated that our scheme of 3D background model is well suited for immersive videos made of both static and dynamic reference cameras. In addition, we also showed that the compact 3D representation of the background environment allows us to easily create traditional 3D effects, at least partially, in the immersive 3D world.

Another possible GPU-based method for producing the 3D background information would be to exploit the OpenGL functionality on the GPUs. If depth buffering with GL GREATER option is used, the resulting model would become quite dependent on the camera views and image resolutions selected for this OpenGL computation. (Multiple camera views would be necessary for spanning the entire 3D space.) In this case, it is more possible that foreground surfaces falsely remain as background surfaces. For instance, in Fig. 5, if the camera for building the 3D background model is placed in the middle of RC_p and RC_q , and the OpenGL rendering is carried out for finding back-most surfaces, the triangle such as T_2 easily remains as a background surface as there is no surface behind T_2 relative to the selected camera view. In addition, the depth values of immersive videos are often imperfect due to insufficient precision and/or inaccurate estimation. If the OpenGL framework is employed, the inevitable z-fighting problem will produce a probable background image, PBI, of poor quality with annoying visual artifacts.

By contrast, the developed ray tracing-based method keeps the triangles in the vicinity of background surfaces, as implied by Fig. 6, as a part of 3D background model. This enables a more effective blending operation when the PBI is created using ray tracing from the probable background surface, PBS. Furthermore, the current hardware-accelerated ray tracing allows us to perform a more sophisticated foreground culling process against reference cameras to remove false background triangles more effectively. Through the GPU-assisted ray tracing capability, the triangles of 3D background model are also organized easily and quickly into a BVH tree during foreground culling, in turn enabling efficient 3D graphics operations in immersive 3D environment.

The developed method may appear limited in applicability in that it demands GPUs that support hardware-assisted ray tracing. However, because such GPUs have recently become increasing common, this requirement will not be a problem. Table 3 compares the computational cost to render the Museum sequence with disocclusion filling on three recent GPUs (refer to Table 2b again). As these statistics show, the newest GPU ('4090' in the table) provides significantly better ray tracing (CUDA and OpenGL also) performance, which strongly implies that the presented framework for the real-time stereoscopic rendering of immersive video will be easily implementable on commodity GPUs in the very near future.

Currently, the presented immersive rendering scheme assumes static lighting only in input video as in many studies that exploit the temporal correlation between video frames for background modeling. This may cause temporal aliasing when the lighting condition varies over time. To overcome this problem, we are going to investigate how to efficiently reflect dynamic lighting changes during the PBS update for rendering natural images. A possible solution would be to partially replace the colors of visible vertices in the BVH of the PBS with those of 3D warped reference views of a new time frame.

Furthermore, the constructed 3D background model may contain some foreground surfaces, often temporally, during rendering without being removed by the foreground culling process. This problem is partially due to the numerical inaccuracy encountered in the GPU-assisted real-time ray tracing or, more seriously, the insufficient scene information provided by video streams. An additional application of a more sophisticated foreground removal algorithm, at least periodically, to the PBS structure would be helpful. Last but not least, the developed method tries to store only the farthest surfaces in the 3D background model, which may result in disocclusions on the surfaces of objects in the middle of scene. While it is quite desirable to keep multi-layers of surfaces in the 3D background model, the development of such a method that spends only a little time in the realtime immersive rendering pipeline remains for future work.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10055-023-00921-w.

Author Contributions YK, JY, SK, II helped in conceptualization; YK, JY, JY, SK, II formally analyzed and investigated; YK, JeY, II involved in implementation; II helped in writing and funding acquisition.

Funding This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2020R1A2C2011709) and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2022-0-00022).

Data availability The MPEG Test Model for Immersive Video (TMIV) dataset including *Museum*, *Hijack*, *Painter*, and *Frog* is available in the MPEG Immersive Video (MIV) repository, https://mpeg-miv.org. The immersive videos *Gymnasium-a & b* generated by the authors are available from the corresponding author on reasonable request.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

- Boyce JM, Doré R, Dziembowski A, Fleureau J, Jung J, Kroon B, Salahieh B, Vadakital VKM, Yu L (2021) MPEG immersive video coding standard. Proc IEEE 109(9):1521–1536
- Broxton M, Flynn J, Overbeck R, Erickson D, Hedman P, Duvall M, Dourgarian J, Busch J, Whalen M, Debevec P (2020) Immersive light field video with a layered mesh representation. ACM Trans Gr 39(4):86
- Chen KY, Tsung PK, Lin PC, Yang HJ, Chen LG (2010) Hybrid motion/depth-oriented inpainting for virtual view synthesis in multiview applications. In: 2010 3DTV-conference: the true vision - capture, transmission and display of 3D video, pp. 1–4
- Criminisi A, Perez P, Toyama K (2004) Region filling and object removal by exemplar-based image inpainting. IEEE Trans Image Process 13(9):1200–1212
- Criminisi A, Blake A, Rother C, Shotton J, Torr PHS (2007) Efficient dense stereo with occlusions for new view-synthesis by fourstate dynamic programming. Int J Comput Vis 71(1):89–110
- Doré, R.: Technicolor 3DoF+ test materials. Standard ISO/IEC JTC1/SC29/WG11 MPEG/M42349 (2018)
- Doyen D et al. (2017) Light field content from 16-camera rig. Standard ISO/IEC JTC1/SC29/WG11 MPEG/M40010
- Elharrouss O, Almaadeed N, Al-Maadeed S, Akbari Y (2020) Image inpainting: a review. Neural Process Lett 51(2):2007–2028
- Fehn, C.: Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV. In: Proceedings of stereoscopic displays and virtual reality systems XI (SPIE), vol. 5291, pp. 93–104 (2004)
- Flynn J, Broxton M, Debevec P, DuVall M, Fyffe G, Overbeck R, Snavely N, Tucker R (2019) DeepView: view synthesis with learned gradient descent. In: 2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR), pp. 2362– 2371 (2019)
- Huang Y, Zhang C (2008) A layered method of visibility resolving in depth image-based rendering. In: 2008 19th international conference on pattern recognition, pp. 1–4
- Jam J, Kendrick C, Walker K, Drouard V, Hsu J, Yap MH (2021) A comprehensive review of past and present image inpainting methods. Comput Vis Image Underst 203:103147
- Jung J, Kroon B (2022) Common test conditions for MPEG immersive video. Standard ISO/IED JTC1/SC29/WG04 MPEG/N0203
- Lee J, Kim Y, Yun J, Yun J, Cheong WS, Ihm I (2021) Disocclusion-reducing geometry for multiple RGB-D video streams. In: 2021 IEEE conference on virtual reality and 3D user interfaces abstracts and workshops (VRW), pp. 603–604
- Luo G, Zhu Y, Weng Z, Li Z (2020) A disocclusion inpainting framework for depth-based view synthesis. IEEE Trans Pattern Anal Mach Intell 42(6):1289–1302
- Luo G, Zhu Y, Li Z, Zhang L (2016) A hole filling approach based on background reconstruction for view synthesis in 3D video. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR), pp. 1781–1789
- Mark WR, McMillan L, Bishop G (1997) Post-rendering 3D warping. In: Proceedings of the 1997 symposium on interactive 3D graphics (I3D '97), pp. 7–16
- Müller K, Merkle P, Wiegand T (2011) 3-D video representation using depth maps. Proc IEEE 99(4):643–656
- Ndjiki-Nya P, Köppel M, Doshkov D, Lakshman H, Merkle P, Muller K, Wiegand T (2011) Depth image-based rendering with advanced texture synthesis for 3-D video. IEEE Trans Multimed 13(3):453–465
- NVIDIA: NVIDIA OptiX 7.4 Programming Guide (2021)
- Qin Z, Zeng Q, Zong Y, Xu F (2021) Image inpainting based on deep learning: a review. Displays 69:102028

- Rahaman DMM, Paul M (2018) Virtual view synthesis for free viewpoint video and multiview video compression using Gaussian mixture modeling. IEEE Trans Image Process 27(3):1190–1201
- Salahieh B et al. (2018) Kermit test sequence for windowed 6DoF activities. Standard ISO/IEC JTC1/SC29/WG11 MPEG/ M43748
- Schmeing M, Jiang X (2010) Depth image based rendering: A faithful approach for the disocclusion problem. In: 2010 3DTV conference: the true vision—capture, transmission and display of 3D video, pp. 1–4
- Smolic A (2011) 3D video and free viewpoint video—from capture to display. Pattern Recognit 44(9):1958–1968
- Sun W, Au OC, Xu L, Li Y, Hu W (2012) Novel temporal domain hole filling based on background modeling for view synthesis. In: 2012 19th IEEE international conference on image processing, pp. 2721–2724
- Vadakital VKM, Dziembowski A, Lafruit G, Thudor F, Lee G, Alface PR (2022) The MPEG immersive video standard-current status and future outlook. IEEE Multimed 29(3):101–111

- Yao C, Tillo T, Zhao Y, Xiao J, Bai H, Lin C (2014) Depth map driven hole filling algorithm exploiting temporal correlation information. IEEE Trans Broadcast 60(2):394–404
- Zhou T, Tucker R, Flynn J, Fyffe G, Snavely N (2018) Stereo magnification: learning view synthesis using multiplane images. ACM Trans Gr 37(4):65
- Zhu C, Zhao Y, Yu L, Tanimoto M (eds) (2013) 3D-TV system with depth-image-based rendering: architectures, techniques and challenges. Springer, New York

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.