

Chapter

View Synthesis Tool for VR Immersive Video

*Sarah Fachada, Daniele Bonatto, Mehrdad Teratani
and Gauthier Lafruit*

Abstract

This chapter addresses the view synthesis of natural scenes in virtual reality (VR) using depth image-based rendering (DIBR). This method reaches photorealistic results as it directly warps photos to obtain the output, avoiding the need to photograph every possible viewpoint or to make a 3D reconstruction of a scene followed by a ray-tracing rendering. An overview of the DIBR approach and frequently encountered challenges (disocclusion and ghosting artifacts, multi-view blending, handling of non-Lambertian objects) are described. Such technology finds applications in VR immersive displays and holography. Finally, a comprehensive manual of the Reference View Synthesis software (RVS), an open-source tool tested on open datasets and recognized by the MPEG-I standardization activities (where “I” refers to “immersive”) is described for hands-on practicing.

Keywords: DIBR, RVS, view synthesis, depth map, virtual reality, rendering, 3D geometry, light field, non-Lambertian

1. Introduction

Photography has a vast history as it is used to preserve our lives’ most important memories. As such, it tries to conserve a scene as realistically as possible. During the years, it evolved from the camera obscura [1, 2] where scenes were captured only for a brief moment, to black and white photography, requiring to stay still in front of the camera for long hours, to nowadays imaging devices, where the picture is captured instantaneously, digitalized, and the colors are close to what our eyes perceive [3].

Though it preserves the content of the scene, the immersion is lost, as well as the depth information, since the camera projects the scene from 3D to 2D.

To increase the immersion, the next step is to recreate the parallax of the scene, giving the opportunity to the viewer to move freely and see different perspectives, exactly as if the subject was miniaturized in front of our eyes, or the environment virtually rendered around us. Despite this desire, no device capable of acquiring the scene in its entirety directly in 3D has been designed so far.

Creating the parallax effect assumes capturing the scene from all the possible viewpoints and selecting the viewpoint to display on demand for the user’s viewing position. This is physically impossible; instead, we may synthesize any viewpoint from only a couple of captured viewpoints generating all missing information following some basic assumptions [4, 5].



Figure 1.

DIBR brings photographs to 3D by using depth information to create new viewpoints. It preserves photorealism and allows the user to experience motion parallax.

There exist many approaches to generate novel viewpoints from input views. Early methods were based on 3D reconstruction [6, 7] to render the obtained 3D model. More recently, neural radiance fields (NeRF) [8] used machine learning methods to recreate a volumetric representation of the scene. Other methods avoid the explicit 3D information reconstruction, such as depth image-based rendering (DIBR) [9] that will be described in this chapter, or multiplane images [10, 11]. Finally, novel viewpoints can be synthesized by an intelligent interpolation using physical invariants (the epipolar plane image), rather than interpolating directly the image’s colors. Representatives of this last category are the shearlet transform [12] and techniques using deep learning [13].

This chapter provides comprehensive elements to bring photographs of a natural scene to the third dimension, for example, making the captured scene immersive, through holography or virtual reality (VR). The presented 3D rendering technique differs from traditional computer graphics by its input—instead of modeling 3D objects with their geometry and materials that interact with light sources, we use photographs that are warped to follow the viewer gaze direction using the reference view synthesis (RVS) [14–16] software that follows the view synthesis process of **Figure 1**.

RVS has been developed during the exploration and standardization activities of MPEG-I – where “I” refers to Immersive – focusing on developing new compression and file formats for immersive video.

The chapter is structured as follows—the first part explains the principles of depth image-based rendering, gives an overview of the possible artifacts that can be encountered when creating a DIBR implementation, and finally, implementation details of RVS are described. The second part provides practical advice for using RVS on some example datasets.

2. Principles of depth image-based rendering

To recreate the parallax effect, we use the depth image-based rendering [9] method. It warps or distorts the input color image as a function of its associated depth map, which itself stores, for every pixel, the distance between the camera and the projected point along the camera optical axis. This method is based on the observation that a stereoscopic pair of images, for example, taken with a few centimeters shift

between each other, carry the depth information of the photographed subject. As shown in **Figure 2**, the relative shift d , aka. the disparity, of foreground objects is larger than for background objects.

2.1 Projection equation and disparity

Let us consider two pinhole cameras facing an object at distance D (see **Figure 2**). The projection of this object on each image will have a disparity d . Using the similar triangles ratios of $\frac{f}{D}$ and $\frac{d_1+d_2}{B}$, we obtain:

$$d = \frac{B \times f}{D} \quad (1)$$

where B is the baseline, i.e., the distance between the two camera centers, and f their focal length.

This implies that, given two images and their depth maps, we can create a virtual view in the middle, between the inputs, by shifting the pixels over half their disparity.

Eq. 1 can be generalized to any camera settings using the pose (translation and rotation – extrinsic parameters, Eq. 2) and internal camera parameters (focal length f_x and f_y expressed in pixels in the x and y directions, and principal point (pp_x, pp_y) – intrinsic parameters, Eq. (3).

We call an input image and its camera parameters an *input* viewpoint. We aim to recreate a new virtual view with given new parameters, called *target* viewpoint. For this, we deproject (i.e., from 2D to 3D) the pixels of the input image to 3D, and reproject (i.e., from 3D to 2D) them to the target image using the projection equation.

Let $P = (R|t)$ be the inverse (i.e., world to camera) 3×4 pose matrix of a camera with R the rotation matrix and t the translation:

$$P = (R|t) = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix}, \quad (2)$$

and K its 3×3 intrinsic matrix:

$$K = \begin{pmatrix} f_x & 0 & pp_x \\ 0 & f_y & pp_y \\ 0 & 0 & 1 \end{pmatrix}. \quad (3)$$

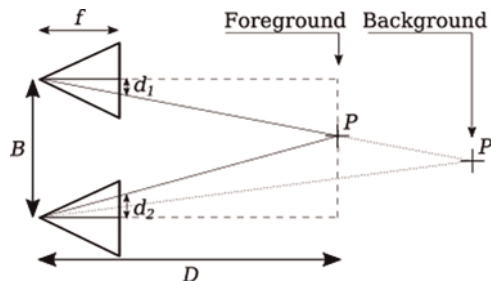


Figure 2. Disparity $d = d_2 - d_1$ between two pixels representing the same projected point.

In homogeneous coordinates, a point $X = (x, y, z, 1)^t$ at depth D from the input camera $p_{in} = (u, v, 1)^t$ is projected to a pixel $p_{in} = (u, v, 1)^t$ following the projection Equation [6]:

$$Dp_{in} = K_{in}P_{in}X \quad (4)$$

Hence, given the input image and the depth value of the pixel, we can deproject X :

$$X = D(R_{in}|-R_{in}^{-1}t_{in})^T K_{in}^{-1}p_{in} \quad (5)$$

Eventually, this allows to reproject X in the new camera, using Eq. 4 with P_{out} and K_{out} :

$$p_{out} \propto DK_{out}P_{out}(R_{in}|-R_{in}^{-1}t_{in})^T K_{in}^{-1}p_{in} \quad (6)$$

To obtain the pixel value, we divide the obtained vector by the third coordinate (i.e., the depth of the point in the new camera).

Applying this operation to every pixel of the input image creates a novel view.

The core principle of DIBR is to apply this deprojection and reprojection to all the pixels of the input images, using a depth map (i.e., a single-channel image encoding the depth value of each pixel). RVS uses this basic principle, but of course, there are many pitfalls one should handle correctly. This is further explained in the following sections.

2.2 Frequent artifacts

We now know the basic principles of DIBR. Unfortunately, simply shifting the pixels of an input image in the function of their depth does not create a photo-realistic result.

The first problem is occlusion handling. When an object is visible in the input image but hidden by an object lying more in the foreground in the target, it is occluded and its pixels should not appear in the rendered image. This can be solved by choosing, among all the pixels from various objects ending up in the same pixel on the screen, the pixel with the minimal depth. A more critical problem is the one of disocclusions, for example, when an object should be visible in the target image but does not appear in the input image because it is hidden (**Figure 3a**). In that case, a

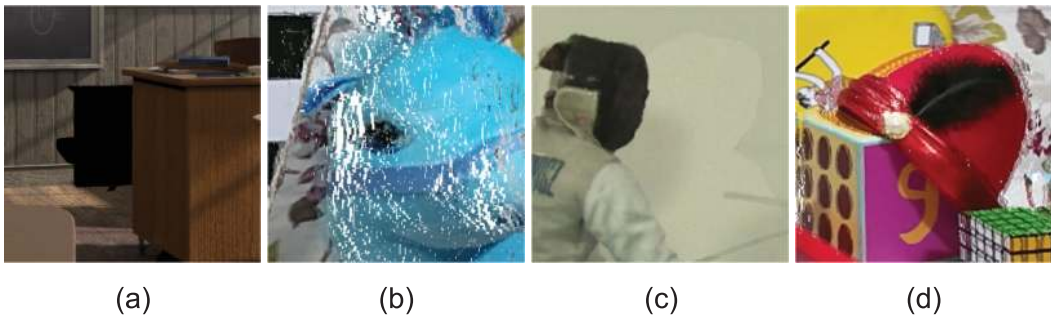


Figure 3. (a) Disocclusion artifact (classroom dataset), (b) crack artifacts (Toystable dataset), (c) Artifacts due to inconsistency in color among the input images. (dataset fencing, courtesy of Poznan University of Technology), (d) ghosting artifacts (dataset Toystable).

hole is created in the rendered image. One solution is to add more input images in the hope to obtain this missing information [15, 17]. Another approach is to inpaint the empty pixels [18, 19]. In RVS, it is possible to choose any number of viewpoints and a basic inpainting fills the remaining disocclusions.

Cracks and dilation are other frequent DIBR artifacts. We can observe them in **Figure 3b**. They are created as the user moves forward (step-in), increases the resolution (zoom), or observes slanted objects. Those cracks correspond to pixels in the target that do not have a preimage in the input view (i.e., no input pixel is mapped to them). However, as their neighboring pixels have a preimage, their color can be interpolated. In other words, the input pixels should be mapped to more than one pixel to compensate for this effect. This can be done using superpixels [20], adapting the pixels size to the camera movement [21], or linking neighboring pixels for rasterization [15, 16] (chosen solution in RVS: adjacent pixels are grouped into triangles that are colorized).

Even if increasing the number of input images can reduce the number of disocclusions, it brings new challenges, as those views need to be consistent in color, in estimated geometry, and in estimated pose. Notably, the depth estimation and the blending of multiple views together rely on consistent colors between the images. As not all camera sensors are equal, small differences in color rapidly generate incoherent depth estimations or nonhomogeneous color patches during view blending (**Figure 3c**). Color correction is usually needed prior to the view synthesis [22, 23] or during the blending step [24, 25].

Moreover, as DIBR relies on the depth information, errors in the depth estimation, a misalignment between the color image and the depth map, or errors in the camera pose estimation lead to ghosting artifacts. When several views are blended together, these artifacts make the objects or their borders appear doubled (**Figure 3d**). A depth map refinement [26, 27] is one way to solve this problem. Another is to choose weighted blending coefficients based on the reliability of each input image [11, 16] (chosen solution in RVS).

Finally, DIBR is structurally limited to the rendering of diffuse objects. Indeed shifting the pixels in the function of their depth assumes that they do not present view-dependent aspects, such as transparency or specularities. When such objects, so-called non-Lambertian, are present in the scene, the linear hypothesis in pixel displacement in the function of the camera displacement is not valid anymore. Adapting the DIBR principles to non-Lambertian objects is nevertheless possible by exploiting additional information, such as structure, normal, and indexes of refraction [28], or a more accurate approximation of the pixel displacement [29–31] (chosen solution in RVS).

2.3 RVS in practice

The DIBR software RVS is designed to render novel viewpoints from any number of input images and depth maps and their camera parameters, without suffering from the above limitations. In order to create a novel view, the input images are warped sequentially. The obtained result is then blended into an image accumulating the outputs of each reprojected input image. This pipeline is shown in **Figure 4**. The warping and blending operations are performed alternatively for each input image using OpenGL [32] or on the CPU [15].

To obtain high-quality results, it is recommended to select candidate input views properly. Therefore, the first step in RVS is an optional view selection. The n views the

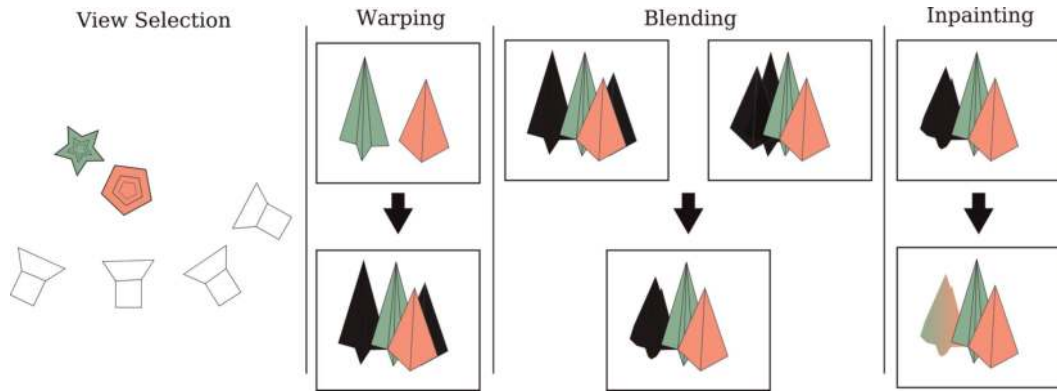


Figure 4. Overview of the processing pipeline. (1) view selection (optional), (2) warping, (3) blending, (4) Inpainting (optional).

closest to the target image are selected in order to reduce the computation time. Otherwise, all the input images are used to create a new viewpoint.

The second step in RVS is the warping phase. Each input image is divided into a grid of triangles whose vertices are adjacent pixels (**Figure 5a**). Each of these vertices is reprojected to fit to the new camera pose and parameters (**Figure 5b**) and rasterized to avoid the cracks artifacts of **Figure 3b**. Then, each new triangle is given a score that will be used in the blending phase. This score describes the quality of a warped triangle—if the pixels lie on a disocclusion area, their triangle will be stretched and should hence be discarded from the final result (black areas in **Figure 4**-warping and **Figure 5c**). The remaining triangles are then rasterized according to their vertex color in the input image (**Figure 5c**). A depth test prioritizes the pixels with the lowest depth.

When the input images are warped to the target viewpoint, the results need to be blended into one single image. For a given pixel, the final output color c is the weighted mean of the color c_i of each warped input:

$$c = \frac{1}{\sum_i w_i} \sum_i w_i c_i \quad (7)$$

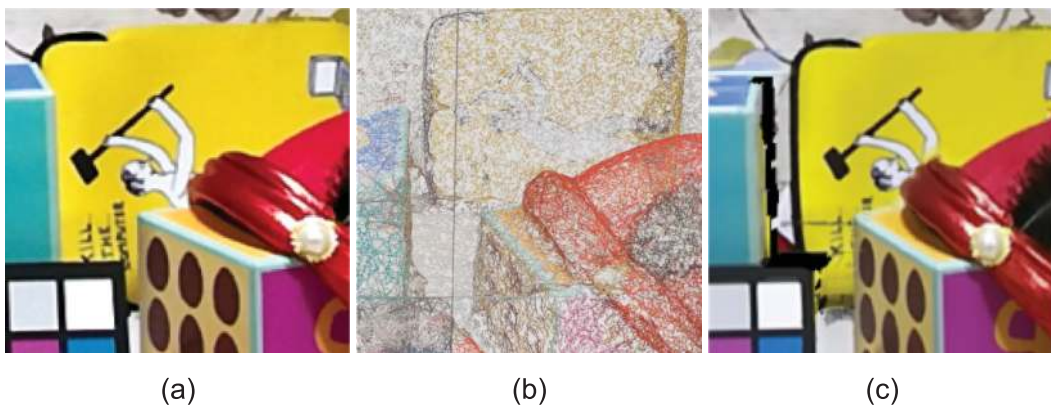


Figure 5. Adjacent pixels of an input image (a) are grouped into triangles independently of their depth before being reprojected to their new image location (b). Triangles detected as lying on a disocclusion are discarded, resulting in a new warped image (c).

where w_i is a weight representing the quality of a triangle [16], prioritizing foreground objects and highest quality triangles.

Finally, as shown in the inpainting of **Figure 4**, when multiple views are blended together, several occluded regions remain; if the occlusions are small enough, a basic inpainting process can be applied to remove them. Of course, the quality of the inpainting can compromise the overall image quality, hence, inpainting is not recommended. In RVS, the inpainting is not automatic but can be activated. In that case, the empty pixels take the color of the nearest non-empty pixel.

2.3.1 Non-Lambertian case

In the general case, DIBR uses depth maps to predict pixel displacement. However, a point on a non-Lambertian surface does not have a proper color (its color can rapidly change with a change in viewing direction); its appearance is a function of the surrounding scene, the normal at that point, and the index of refraction for refractive surfaces (see **Figure 6**). This not only makes depth estimation through stereo matching impossible but also implies that even with a correct depth map, the object cannot be rendered by a simple pixel shifting.

Alternatively, to model the non-Lambertian surface in itself, it is possible to track its feature movements on the surface [29, 33, 34]. DIBR can be generalized to non-Lambertian objects by replacing the usual depth maps with the coefficients of a polynomial approximating the non-Lambertian features displacement [30, 31]. To clearly understand what this means, let us start with what happens for diffuse objects, where for a lateral camera movement (x, y) , the new position (u, v) of a pixel (u_0, v_0) is given by:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + \frac{f}{D} \begin{pmatrix} x \\ y \end{pmatrix}. \quad (8)$$

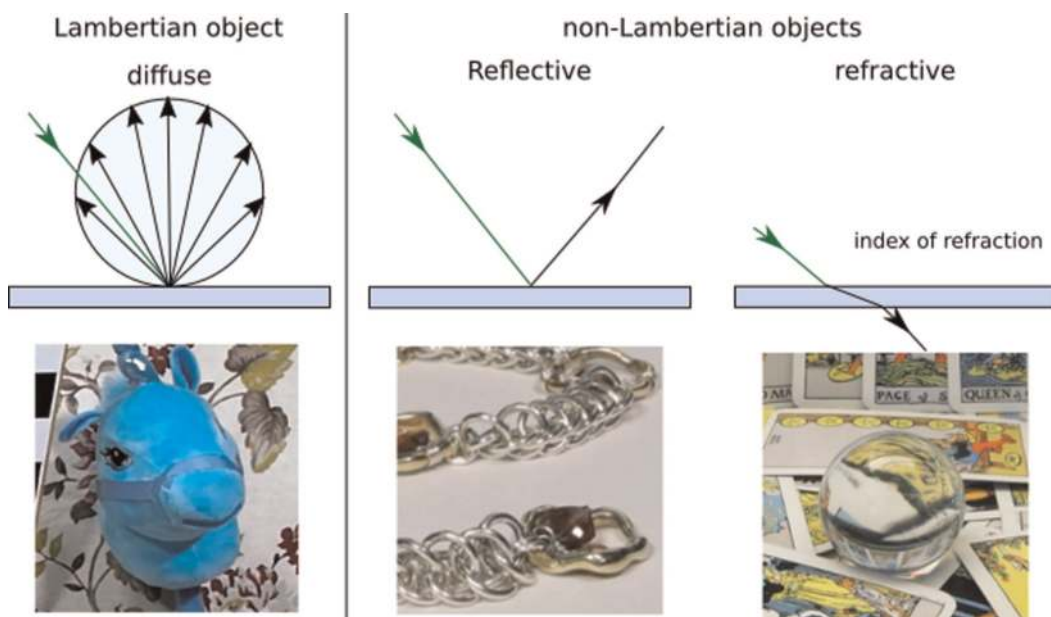


Figure 6. The aspect of non-Lambertian objects is view dependent—Their surface does not appear the same color in each viewing direction.

We extend this equation for non-Lambertian objects using polynomials:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + \begin{pmatrix} P_u(x, y) \\ P_v(x, y) \end{pmatrix}, \quad (9)$$

with $P_u(x, y) = \sum_i \sum_j a_{ij} x^i y^j$ and $P_v(x, y) = \sum_i \sum_j b_{ij} x^i y^j$. Clearly, the diffuse case corresponds to $a_{1,0} = b_{0,1} = \frac{f}{D}$ and all other coefficients a_{ij}, b_{ij} set to zero.

Consequently, Eq. 9 approximates by a polynomial the nonlinear displacement of a refracted or reflected feature moving on non-Lambertian objects.

However, the polynomial expression rapidly diverges in extrapolation (e.g., when synthesizing a target view that is outside of the input images' hull). The computed feature displacement becomes greater than the inverse of the non-Lambertian object's depth, making the feature to be rendered outside of the non-Lambertian surface. This approximation is hence designed for interpolation and small extrapolation only.

Furthermore, these polynomials are not directly related to the physical reality of the non-Lambertian object. Hence, contrary to the simple relation linking the depth to the disparity of a diffuse object cf. **Figure 2** and Eq. (1), the polynomials of Eq. (9) do not give the object geometry or the index of refraction.

The polynomial is rather designed to “track” non-Lambertian features that move nonlinearly across the input images. It nevertheless encounters the following limitations. For content with semi-transparent objects, the maps should be divided into several layers before applying the polynomial or depth image-based rendering. Scenes with glints and glossiness make it difficult to track features on their surfaces, often leading to a failure case of the proposed method.

3. Reference view synthesis (RVS) software

This section provides practical recommendations for the use of the reference view synthesizer (RVS) [14–16, 32, 35] (<https://gitlab.com/mpeg-i-visual/rvs>) developed as a DIBR-based view synthesizer for the MPEG immersive video (MIV) standard (<https://mpeg-miv.org>). Without further details on the compression and storage of immersive content [36], we give a comprehensive method to practically use the software on some test sequences (also provided to the MPEG community while developing RVS).

The following paragraphs give documentation on the image format, the axis system, and the data structure to synthesize new viewpoints from available ready-to-use datasets and/or new content users may provide.

3.1 Input images

RVS can accept any number of input images with depth maps, the only limitation being the computer memory. Each input color image must be provided along with its corresponding depth map.

3.1.1 Color images

The color images can be encoded on three RGB color channels, with 8-bit integers each, in any image format readable by OpenCV, for example, PNG or JPEG format.

Additionally, raw images in YUV can be used, with a bit depth of 8, 10, or 16 bits. In this case, multi-frame raw video can be used, applying the view synthesis on all specified frames.

3.1.2 Depth maps

The depth maps represent the depth coordinate of every point in the image following the forward axis of the camera. Similar to color images, they can be provided in different formats. They have to match the resolution of the input images, but they use only one channel.

The first option is to use the OpenEXR format. In that case, the software reads the depth value in float and uses it directly for reprojection.

In the case of integer coded formats, such as YUV or PNG, the precision can be set to 8, 10, or 16 bit per depth value—the bit depth. YUV files can be encoded in YUV420 or YUV400 format, only the Y channel being used. However, the quantization does not allow to directly use the integer as a depth value. Indeed, it would be impossible to use a depth map in meter units for objects in the range of a few meters or centimeters from the cameras.

To overcome this problem, the depth value is encoded into MPEG's disparity format, mapping the closest object to $2^{\text{bitdepth}} - 1$, and the farthest to 1. To obtain the actual depth value, first we divide the encoded depth map value by $2^{\text{bitdepth}} - 1$ to obtain a value d in the range of $[0, 1]$, then remap the value in the range $[n, f]$ using:

$$d' = \frac{f \times n}{n + d \times (f - n)}. \quad (10)$$

With n and f the near and far values of the scene and d' the depth value lying in $[n, f]$.

For very far objects, this equation is simplified ($f \geq 1000$) to

$$d' = \frac{n}{d} \quad (11)$$

The value 0 in the encoded depth maps is considered as invalid depth. It corresponds, for example, to disocclusions in a depth-sensing device-acquired map.

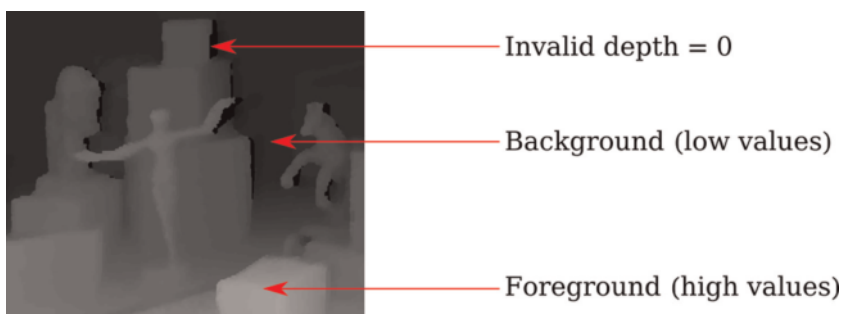


Figure 7. Encoded depth map on integer values. Due to the shift between the color sensor and the depth sensor, the depth map reprojected to the color image misses some information, leaving invalid pixels, encoded on 0. The foreground objects are encoded on high disparities, while the background objects are encoded on low disparities.

Figure 7 shows an encoded depth map with invalid pixels and objects at different depths. Clearly, the foreground has high values, which corresponds to being a disparity value, that is, the inverse of a depth, cf. Eq. (11).

In the case of polynomial maps for non-Lambertian objects, it is possible to encode up to degree 3 polynomials, hence 18 coefficients, and pass an additional depth map and mask for the non-Lambertian objects. Those coefficients are encoded similarly to the depth maps, using EXR (directly the float value) or YUV (normalized) format. The polynomial maps are numbered from 0 to 19 as follows.

$$\begin{aligned} P_u(x, y) &= a_0x^3 + a_1x^2y + a_2xy^2 + a_3y^3 + a_4x^2 + a_5xy + a_6y^2 + a_7x + a_8y, \\ P_v(x, y) &= b_0x^3 + b_1x^2y + b_2xy^2 + b_3y^3 + b_4x^2 + b_5xy + b_6y^2 + b_7x + b_8y \end{aligned} \quad (12)$$

with a_i corresponding to the map i and b_i to the map $10 + i$. The remaining map 9 is used to encode the depth map for Lambertian objects and the map 19 is used as a mask representing non-Lambertian objects (0 for Lambertian, 1 for non-Lambertian). The coefficients not used are left to 0. If the coefficients are encoded in YUV format, the depth (map n°9) is normalized using Eq. 10, the mask (map n°19) has 0 and 1 values and the other coefficients are linearly normalized between minimal m and maximal M values: $a_i = (M - m)a_i + m$.

3.2 Camera parameters

Additionally to the input images, the camera parameters must be known to create a novel view with DIBR and RVS. The extrinsic parameters describe the position and the rotation of the camera (Eq. 2), while the intrinsic parameters describe the projection matrix (Eq. 3). Perspective and equirectangular projections are also supported, requiring a slightly different description, as explained hereafter.

3.2.1 Extrinsic parameters

Common graphics processing software and APIs, such as Blender [37], COLMAP [38], OpenGL [39], Vulkan [40], specify their own coordinate system, often admitting different axes and directions, and different image coordinates. Transferring data from one application to the other requires then several coordinate transformation steps, which will be summarized here. We use the Omnidirectional Media Format (OMAF) [41] coordinate system of MPEG-I, combined with yaw-pitch-roll angles.

OMAF is the first industry standard for VR. It specifies the coordinate system used in VR applications, the projection and rectangular region-wise packing methods, the metadata storage, encapsulation, signaling, and streaming of omnidirectional data, and finally the media profiles and presentation profiles. For these reasons, it has been adopted in the camera configuration files of RVS.

The OMAF coordinate system is described in **Figure 8**. The axes are defined as follows:

- X: Back-to-front, forward
- Y: Lateral, left

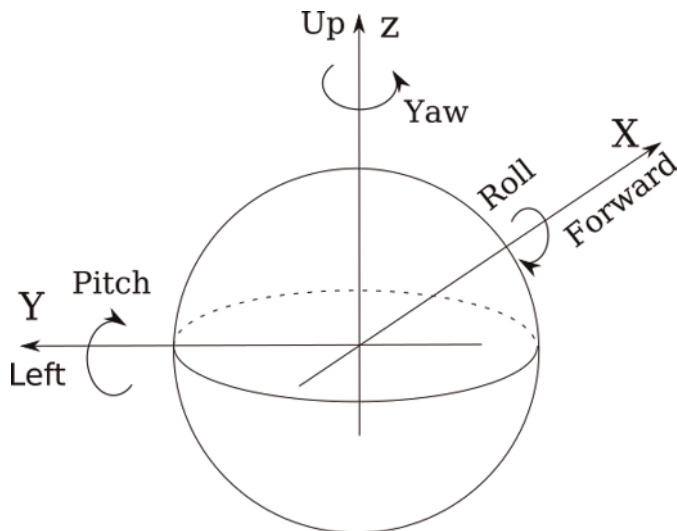


Figure 8.
 The omnidirectional media format coordinate system.

- Z: Vertical, up

The rotations in degrees are defined with the Yaw-pitch-roll:

- Yaw: Around the vertical axis
- Pitch: Around the lateral axis
- Roll: Around the back-to-front axis

A camera facing forward has all its rotation angles set to 0. The rotation matrix of the camera (world to camera) in our axis coordinate system is then given by:

$$R = R_z(\text{yaw})R_y(\text{pitch})R_x(\text{roll}) \quad (13)$$

In order to transform a coordinate system from an application to OMAF, one needs to define the coordinate change matrix that matches the three axes, for example:

$$P = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} \quad (14)$$

This matrix sets (x', y', z') (OMAF) = $(-z, x, -y)$ (application), that is, it represents a coordinate system with the axes (left, down, backward). To transfer from this system to OMAF, we apply it to the rotation and position as follow:

$$\begin{aligned} R' &= P.R.P^T \\ p' &= P.p \end{aligned} \quad (15)$$

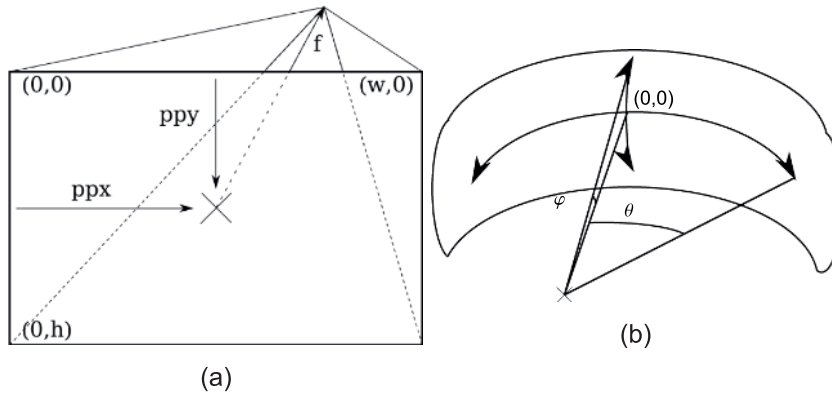


Figure 9. Intrinsic parameters of the camera for (a) a pinhole projection, (b) an equirectangular projection.

where R' and p' are the rotation and position of the OMAF system, while R and p the rotation and position in the old coordinates.

The unit of the coordinate system does not have any prerequisite but must correspond to one of the depth maps.

RVS handles any number of input and target cameras, each of them can have its own parameters and projection types. In the case of a stereoscopic head-mounted display for VR, two target views – one for each eye – need to be synthesized with a relative position (interpupillary distance) corresponding to the eye distance, usually given by the headset's framework along with the intrinsic parameters.

3.2.2 Intrinsic parameters

The intrinsic parameters can be defined for perspective or equirectangular projections. In both cases, the resolution needs to be specified.

For perspective projection, the input images need to be undistorted. In that way, only the focal length and the principal point need to be specified. Those values are in pixel units, the sensor size corresponding to the image resolution.

The focals are given by (f_x, f_y) , corresponding to the horizontal and vertical axis.

The principal point (pp_x, pp_y) is defined from the top-left corner of the image as described in **Figure 9(a)**. A principal point at the center of the image has a value of half the resolution.

In the case of equirectangular projection (**Figure 9b**), the horizontal and vertical viewing range must be specified in degrees. For a full 360° panoramic image, the horizontal range is $[-180, 180]$ and the vertical range is $[-90, 90]$. For a 180° image, the horizontal and vertical ranges are $[-90, 90]$.

3.2.3 Camera file

The image specifications and camera parameters are specified in a *json* file with informative headers. An example with a perspective and an equirectangular camera is given here.

Listing 1.1 Cameras calibration file. Cameras.Json.

```
{
  "Version": "3.0",
  "Content_name": "dataset_name",
  "BoundingBox_center": [0, 0, 0],
  "Fps": 30,
  "Frames_number": number_of_frames_in_yuv_file,
  "Informative": {
    [...]
  },
  "lengthsInMeters": true,
  "sourceCameraNames": [
    "input_camera1_name",
    "input_camera2_name",
    [...]
  ],
  "cameras": [
    {
      "Name": "input_camera1_perspective_name",
      "Position": [x, y, z],
      "Rotation": [ yaw, pitch, roll ],
      "Depthmap": 1,
      "Background": 0,
      "Depth_range": [near, far],
      "Resolution": [w, h],
      "Projection": "Perspective",
      "Focal": [  $f_x$ ,  $f_y$  ],
      "Principle_point": [  $pp_x$ ,  $pp_y$  ],
      "BitDepthColor": 8,
      "BitDepthDepth": 8,
      "ColorSpace": "YUV420",
      "DepthColorSpace": "YUV420"
    },
    {
      "Name": "input_camera2_equirectangular_name",
      "Position": [x, y, z],
      "Rotation": [ yaw, pitch, roll ],
      "Depth_range": [near, far],
      "Resolution": [w, h],
      "Projection": "Equirectangular",
      "Hor_range": [  $\theta_{min}$ ,  $\theta_{max}$  ],
      "Ver_range": [  $\Phi_{min}$ ,  $\Phi_{max}$  ],
      [...]
    },
    [...]
  ]
}
```

An optional parameter, `DisplacementMethod`, can be set to `Polynomial` instead of default parameter `Depth` to specify that, instead of a depth map (Eq. 10), RVS reads a displacement map (Eq. 12). In that case, similarly to the `Depth_range`, a `Multi_depth_range` can be specified for the polynomial coefficients in YUV format.

3.3 View synthesis file

In order to perform the view synthesis, an experiment setup file is created. It gives camera views specifications (which views to synthesize given the input viewpoints) in an easy to use *json* format. The file contains:

- Input and target camera parameters file paths—path to the camera file described in the previous subsection. The same file can be used twice if all the input and target cameras are in the same file;
- Input and target camera names matched with the camera names contained in the camera files. Any number of inputs and outputs can be specified;
- Input images, output images, and depth maps file paths. In the case of polynomial maps, numbered from 0 to 19, the number is replaced by a *;
- Number of output frames. Useful for uncompressed YUV video files. The synthesized number of frames can exceed the number of frames in the input videos by specifying an optional `NumberOfOutputFrames`. In that case, the video will be played back and forth until the desired number of frames is reached;
- Precision: super-resolution factor to reach sub-pixel accuracy;
- Colorspace: internal working color space, can be YUV or RGB. Following the color space used, the result may present small color variations;
- Blending specifications: the method can be `Simple` (for CPU and GPU usage) or `Multispectral` (for CPU). `Multispectral` blending detects the borders in the images, to blend them with a hard threshold and therefore avoids ghosting. The factor represents the power on the weights of Eq. 7.

Listing 1.2. Experiment configuration file. `Experiment.Json`.

```
{
  "Version": "2.0",
  "InputCameraParameterFile": "input_cameras.json",
  "VirtualCameraParameterFile": "target_cameras.json",
  "InputCameraNames": [
    "input_camera1_name",
    "input_camera2_name"
  ],
}
```

```
{
  "Version": "2.0",
  "InputCameraParameterFile": "input_cameras.json",
  "VirtualCameraParameterFile": "target_cameras.json",
  "InputCameraNames": [
    "input_camera1_name",
    "input_camera2_name"
  ],
  "VirtualCameraNames": [
    "output_camera1_name",
    "output_camera2_name"
  ],
  "ViewImageNames": [
    "path_to_texture1.yuv",
    "path_to_texture2.yuv"
  ],
  "DepthMapNames": [
    "path_to_depthmap1.yuv",
    "path_to_depthmap2.yuv"
  ],
  "OutputFiles": [
    "path_to_output_filename1.yuv",
    "path_to_output_filename2.yuv"
  ],
  "StartFrame": 0,
  "NumberOfFrames": number_of_frames_to_synthesize,
  "Precision": 1.0,
  "ColorSpace": "RGB",
  "BlendingMethod": "Simple",
  "BlendingFactor": 5.0
}
```

3.4 Datasets

To test the view synthesis, we provide references to datasets that are provided with their cameras configuration *json* files. Publicly available datasets are available at the following addresses, while others have been provided as test scenes for MPEG-I immersive video exploration and standardization activities.

- Toystable [42, 43]: a natural dataset with perspective cameras (**Figure 10a**) <https://zenodo.org/record/5055542>
- Magritte [44–48]: a synthetic dataset with polynomial non-Lambertian maps (**Figure 10b**) <https://zenodo.org/record/4488242>, <https://zenodo.org/record/5047238>, <https://zenodo.org/record/5047676>, <https://zenodo.org/record/5047769>
- Rabbit [49, 50]: the subaperture views of a multi-plenoptic camera dataset (**Figure 10c**) <https://zenodo.org/record/5053770>

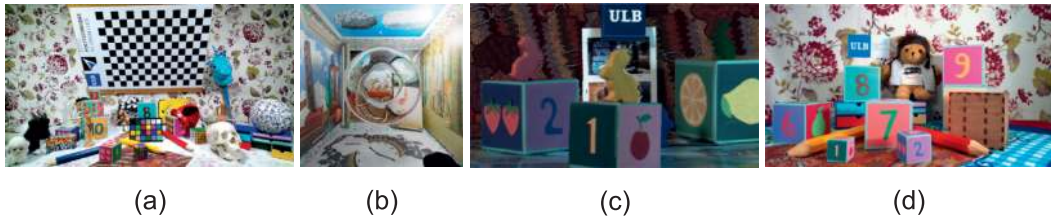


Figure 10.

Overview of the open-source datasets. (a) Toystable consists of two camera arrays at 25 cm (5×5 cameras) and 55 cm (3×5 cameras) from the scene. (b) Magritte is a 21×21 camera array. (c) Rabbit is a 3×7 array of 5×5 subaperture images of a plenoptic camera. (d) Bear (4×8 cameras) is a dataset captured by a Lidar camera with estimated and sensed depth maps.

- Bear [51]: a natural dataset with perspective cameras with estimated and Lidar-sensed depth maps (**Figure 10d**): <https://zenodo.org/record/5047463>
- MPEG test sequences (including [52–55]): <http://mpegx.int-evry.fr/mpegcontent/>

4. Displays

We provide in **Figure 11** results obtained with the RVS software on various display types—autostereoscopic or light field screen (Courtesy ETRO-VUB, Belgium), holographic stereograms [56], and head-mounted displays. Additional videos can be found at the following links: <https://youtu.be/ikJb9JaaE54> (holographic stereogram) and <https://youtu.be/vavw-TcbHf4> (head-mounted-display).

Displaying a dynamic scene in VR requires real-time view synthesis, preferably at 90 frames per second and at a minimum of 30 frames per second for each eye.

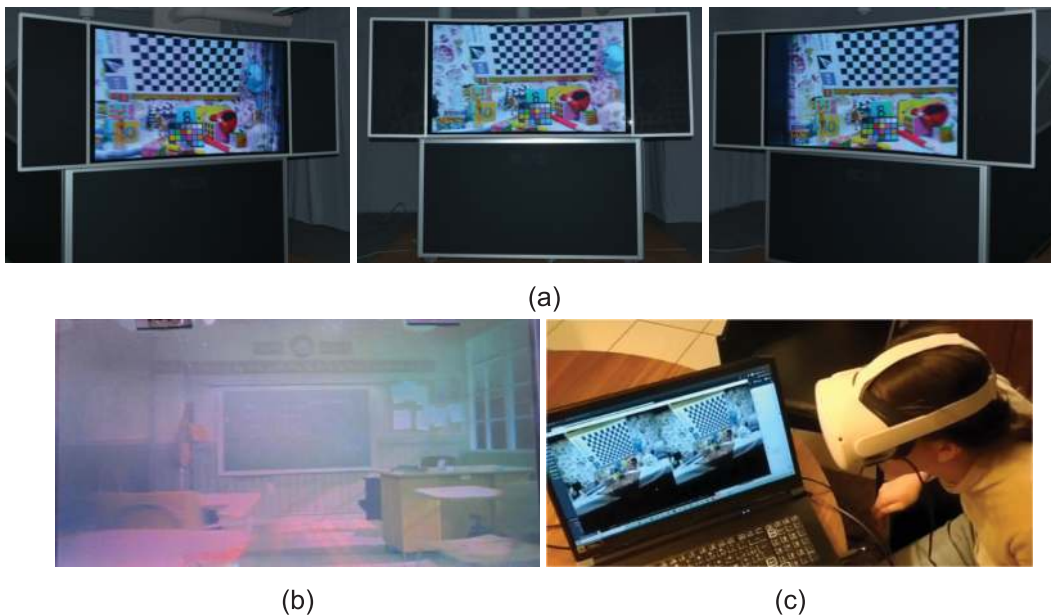


Figure 11.

Instead of acquiring the 100 of views needed for the different kinds of display, RVS recreates them using four input images. (a) Autostereoscopic screen, (b) holographic stereogram, (c) head-mounted display.

Dataset	Resolution	Number of input images							
		1	2	3	4	5	6	7	8
Toystable [42, 43]	1920 × 1080	90	51	46	49	37	36	31	29
Fencing [52]	1920 × 1080	90	90	88	56	52	47	46	40
Painter [53]	2048 × 1088	90	90	90	80	60	56	53	46
Museum [54]	2048 × 2048	52	53	47	31	22	21	16	16
Classroom [55]	4096 × 2048	55	31	30	19	15	11	9	8

Table 1.

The frame rate for view synthesis in VR depends on the number of input images and their resolution. The output images all have the resolution of oculus rift (i.e., 1080 × 1200 pixels). Those results have been obtained on a windows PC with Intel Xeon E5-2680@2.7GHz CPU and NVIDIA GTX 1080TI GPU.

However, the processing time depends on the number of input images and their resolution – since their pixels form the mesh – resulting in different frame rates [16] (see **Table 1**). Using a NVIDIA GTX 1080TI GPU, around four input images at a full HD resolution can be processed to obtain a high visual quality while reaching real-time navigation.

In the case of the currently developed light field head-mounted display [57], the constraint is double—in addition to the real-time requirement, all the light rays reaching the user’s pupils need to be displayed to make the eye accommodation possible on the close objects, that is, not only one image per eye but all the micro-parallax views around the eye position are rendered.

4.1 Additional tools

In this section, we provide references for additional tools, which are not directly involved in the view synthesis but are nevertheless useful to prepare a dataset.

4.1.1 Camera calibration

The first step prior to DIBR is finding the camera parameters. Accurate intrinsic parameters, including distortion parameters, can be found using a calibration checkerboard-pattern, if the scene has a large enough baseline, or directly during the scene reconstruction (structure-from-motion (SfM) with the retrieval of intrinsic parameters). Using a pattern gives more accurate results but requires a supplementary preprocessing step. There exists open-source software such as Kalibr [58] and OpenCV [59] for camera calibration.

Extrinsic parameters of a set of cameras are retrieved using SfM, with or without the intrinsic parameters known as prior [60]. There exist many open-source software such as COLMAP [38] or AliceVision [61]. Those softwares calibrate the camera and automatically undistort the images.

4.1.2 Depth estimation

Besides parameters estimation, DIBR requires corresponding depth maps for each input view. If they are not acquired with a depth-sensing device, they can be computed using stereo-matching algorithms. Among many algorithms, Depth Estimation

Reference Software (DERS) [62] and Immersive Video Depth Estimation (IVDE) [63, 64] are recognized by the MPEG-I community.

5. Conclusions

In this chapter, an overview of the main steps and frequent problems of view synthesis are described. By starting from sparse input pictures, we showed a DIBR method that renders the parallax effect on a multitude of displays, allowing a user to experience new aspects of multimedia immersion. In the second part, a description of how one can start experimenting with the state-of-the-art RVS software is thoroughly explained to avoid common pitfalls.

As research progresses, novel methods to create view synthesis emerge, such as NeRF, however, recent research results demonstrate that DIBR methods will still reach high-quality performances [16], in real time, that will be highly applicable in immersive applications, for example, in the context of MPEG immersive video.

Acknowledgements

This work was supported in part by the Fonds de la Recherche Scientifique – FNRS, Belgium, under Grant n.33679514, ColibriH; and in part by the HoviTron project that received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement N.951989 (<https://www.hovitron.eu/>).

Conflict of interest

The authors declare no conflict of interest.

Abbreviations

CPU	Central Processing Unit
DERS	Depth Estimation Reference Software
DIBR	Depth image-based rendering
GPU	Graphics Processing Unit
IVDE	Immersive Video Depth Estimation
MIV	MPEG Immersive Video
MPEG-I	Moving Picture Experts Group-Immersive
NeRF	Neural Radiance Field
OMAF	Omnidirectional Media Format
RGB	Red-Green-Blue format
RVS	Reference View Synthesizer
SfM	Structure-from-Motion
VR	Virtual Reality
YUV	Luminance-Chrominance format


Author details

Sarah Fachada[†], Daniele Bonatto[†], Mehrdad Teratani and Gauthier Lafruit*
Université Libre de Bruxelles, Brussels, Belgium

*Address all correspondence to: gauthier.lafruit@ulb.be

[†] These authors contributed equally.

IntechOpen

© 2022 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

References

- [1] Euclid of Alexandria. Optics; 300 BC.
- [2] Al-Haytham HI. Book of Optics. Vol. I-VII circa 1027
- [3] Renner E. Pinhole Photography: From Historic Technique to Digital Application. 4th ed. Amsterdam, Boston: Focal Press; 2009
- [4] Maeda T, Suenaga R, Suzuki K, Panahpour Tehrani M, Takahashi K, Fujii T. Free Viewpoint Video for Sports Events Using Multi-Resolution Visual Hull and Micro-Facet Billboarding. Proc. Intl Workshop on Smart Info-media System in Asia, Ayutthaya, Thailand; 2016
- [5] Suenaga R, Suzuki K, Tezuka TP, Tehrani M, Takahashi K, Fujii T. A practical implementation of free viewpoint video system for soccer games. In: Three-Dimensional Image Processing, Measurement (3DIPM), and Applications. Vol. 9393. International Society for Optics and Photonics, San Francisco, CA, USA; 2015, 2015. p. 93930G
- [6] Hartley R, Zisserman A. Multiple view geometry in computer vision. 2nd ed. Cambridge, UK, New York: Cambridge University Press; 2004
- [7] Agarwal S, Furukawa Y, Snavely N, Simon I, Curless B, Seitz SM, et al. Building Rome in a Day. New York, USA: Communications of the ACM. 2011;54(10):105
- [8] Mildenhall B, Srinivasan PP, Tancik M, Barron JT, Ramamoorthi R, Ng R. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In: Vedaldi A, Bischof H, Brox T, Frahm JM, editors. Computer Vision–ECCV 2020. Cham: Springer International Publishing; 2020. pp. 405-421
- [9] Fehn C. Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV. In: Stereoscopic Displays and Virtual Reality Systems XI. Vol. 5291. International Society for Optics and Photonics, San Jose, CA, USA; 2004. pp. 93-105
- [10] Penner E, Zhang L. Soft 3D reconstruction for view synthesis. ACM Transactions on Graphics. 2017;36(6): 1-11
- [11] Mildenhall B, Srinivasan PP, Ortiz-Cayon R, Kalantari NK, Ramamoorthi R, Ng R, et al. Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines. ACM Transactions on Graphics. 2019;38(4). Art. 29, pp. 1-14
- [12] Vagharshakyan S, Bregovic R, Gotchev A. Light field reconstruction using shearlet transform. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2018;40(1): 133-147
- [13] Wu G, Zhao M, Wang L, Dai Q, Chai T, Liu Y. Light field reconstruction using deep convolutional network on EPI. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, HI: IEEE; 2017. pp. 1638-1646
- [14] Kroon B. Reference View Synthesizer (RVS) Manual [N18068]. ISO/IEC JTC1/SC29/WG11, Ljubljana, Slovenia; 2018
- [15] Fachada S, Bonatto D, Schenkel A, Lafruit G. Depth Image Based View Synthesis With Multiple Reference Views For Virtual Reality. In: IEEE 2018-3DTV-Conference: The True Vision-Capture, Transmission and

Display of 3D Video (3DTV-CON).
Helsinki: IEEE; 2018

[16] Bonatto D, Fachada S, Rogge S, Munteanu A, Lafruit G. Real-Time Depth Video Based Rendering for 6-DoF HMD Navigation and Light Field Displays. *IEEE Access*. 2021;**9**: 146868-146887

[17] Li S, Zhu C, Sun MT. Hole filling with multiple reference views in DIBR view synthesis. *IEEE Transactions on Multimedia*. 2018;**20**(8):1948-1959

[18] Telea A. An image inpainting technique based on the fast marching method. *Journal of Graphics Tools*. 2004;**9**(1):23-34

[19] Huang HY, Huang SY. Fast hole filling for view synthesis in free viewpoint video. *Electronics*. 2020;**9**(6): 906

[20] Tehrani PM, Tezuka T, Suzuki K, Takahashi K, Fujii T. Free-viewpoint image synthesis using superpixel segmentation. *APSIPA Transactions on Signal and Information Processing*. 2017; **6**, e5, pp. 1-12

[21] Senoh T, Tetsutani N, Yasuda H. Depth Estimation and View Synthesis for Immersive Media. 2018 International Conference on 3D Immersion (IC3D), Brussels, Belgium; 2018. pp. 1-8

[22] Reinhard E, Adhikhmin M, Gooch B, Shirley P. Color transfer between images. *IEEE Computer graphics and applications*. 2001;**21**(5):34-41

[23] Fecker U, Barkowsky M, Kaup A. Histogram-based prefiltering for luminance and chrominance compensation of multiview video. *IEEE Transactions on Circuits and Systems for Video Technology*. 2008; **18**(9):1258-1267

[24] Dziembowski A, Domański M. Adaptive Color Correction In Virtual View Synthesis. Stockholm: 2018-3DTV-Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON), IEEE; 2018

[25] Dziembowski A, Mieloch D, Rózek S, Domański M. Color Correction for Immersive Video Applications. *IEEE Access*. 2021;**9**:75626-75640

[26] Vijayanagar KR, Loghman M, Kim J. Refinement of depth maps generated by low-cost depth sensors. In: 2012 International SoC Design Conference (ISOCC), Jeju, Korea, IEEE; 2012. p. 355-358

[27] Sancho J, Sutradhar P, Rosa G, Chavarrías M, Perez-Nuñez A, Salvador R, et al. GoRG: Towards a GPU-Accelerated Multiview Hyperspectral Depth Estimation Tool for Medical Applications. *Sensors*. 2021;**21**(12):4091

[28] Lochmann G, Reinert B, Ritschel T, Müller S, Seidel HP. Real-time Reflective and Refractive Novel-view Synthesis. 19th International Workshop on Vision, Modeling and Visualization (VMV), Darmstadt, Germany. Eurographics Association, 2014. pp. 9-16.

[29] Nieto G, Devernay F, Crowley J. Linearizing the Plenoptic Space. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). Honolulu, HI, USA: IEEE; 2017. pp. 1714-1725

[30] Fachada S, Bonatto D, Teratani M, Lafruit G. Light field rendering for non-Lambertian objects. *Electronic Imaging, Stereoscopic Displays and Applications XXXII*. 2021, pp. 54-1-54-8

[31] Fachada S, Bonatto D, Teratani M, Lafruit G. Polynomial Image-Based Rendering for non-Lambertian Objects.

- Munich, Germany: Visual Communication and Image Processing 2021; 2021. p. 5
- [32] Bonatto D, Fachada S, Lafruit G. RaViS: Real-time accelerated View Synthesizer for immersive video 6DoF VR. Burlingame, USA: Society for Imaging Science and Technology (IS&T) - Electronic Imaging; 2020
- [33] Maeno K, Nagahara H, Shimada A, Taniguchi RI. Light Field Distortion Feature for Transparent Object Recognition. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition. Portland, OR, USA: IEEE; 2013. pp. 2786-2793
- [34] Xu Y, Nagahara H, Shimada A, Ri T. Transcut: Transparent object segmentation from a light-field image. Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile; 2015. pp. 3442-3450
- [35] Bonatto D, Fachada S, Lafruit G. RVS kickstart: example test sequences and config files. Brussels, Belgium: ULB DI-fusion; 2019
- [36] Lafruit G, Bonatto D, Tulvan C, Preda M, Yu L. Understanding MPEG-I Coding Standardization in Immersive VR/AR Applications. SMPTE Motion Imaging Journal. 2019;128(10): 33-39
- [37] Blender Online Community. Blender - A 3D Modelling and Rendering Package. Blender Institute, Amsterdam: Blender Foundation; 2021 Available from: <http://www.blender.org>
- [38] Schonberger JL, Frahm JM. Structure-from-Motion Revisited. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA: IEEE; 2016. pp. 4104-4113
- [39] Kessenich J, Baldwin D, Rost R. The OpenGL® Shading Language, Version 4.60.7, The Khronos Group Inc.; 2019
- [40] Group TKVW. Vulkan® 1.2.202 - A Specification; 2021.12.07.
- [41] Omnidirectional Media Format. MPEG-I; 2021 Available from: <https://mpeg.chiariglione.org/standards/mpeg-i/omnidirectional-media-format>
- [42] Bonatto D, Schenkel A, Lenertz T, Li Y, Lafruit G. ULB High Density 2D/3D Camera Array data set, version 2 [M41083]. ISO/IEC JTC1/SC29/WG11; 2017
- [43] Bonatto D, Fachada S, Lafruit G. ULB ToysTable. Zenodo; 2021. Available from: <https://zenodo.org/record/5055543>.
- [44] Fachada S, Bonatto XY, Teratani M, Lafruit G. Two Non-Lambertian Test Materials and View Synthesis Performance [m56450]. ISO/IEC JTC1/SC29/WG11; 2021
- [45] Fachada S, Bonatto D, Teratani M, Lafruit G. Transparent Magritte Test Sequence. Zenodo; 2021 Available from: <https://zenodo.org/record/4488243>
- [46] Fachada S, Bonatto D, Teratani M, Lafruit G. Mirror Magritte Torus Test Sequence; 2021. Available from: <https://zenodo.org/record/5048262>
- [47] Fachada S, Bonatto D, Teratani M, Lafruit G. Magritte Sphere Test Sequence; 2021. Available from: <https://zenodo.org/record/5048265>
- [48] Fachada S, Bonatto D, Teratani M, Lafruit G. Magritte Sphere Video Test Sequence; 2021. Available from: <https://zenodo.org/record/5048270>

- [49] Fachada S, Xie Y, Bonatto D, Lafruit G, Teratani M. [DLF] Plenoptic 2.0 Multiview Lenslet Dataset and Preliminary Experiments [m56429]. ISO/IEC JTC1/SC29/WG11; 2021
- [50] Fachada S, Yupeng X, Bonatto D, Lafruit G, Teratani M. RabbitStamp Test Sequence; 2021. Available from: <https://zenodo.org/record/5053770>
- [51] Xie Y, Fachada S, Bonatto D, Lafruit G. HoviTronBear Test Sequence; 2021. Available from: <https://zenodo.org/record/5047464>
- [52] Domański M, Dziembowski A, Grzelka A, Mieloch D, Stankiewicz O, Wegner K. Multiview test video sequences for free navigation exploration obtained using pairs of cameras [M38247]. ISO/IEC JTC1/SC29/WG11; 2016
- [53] Doyen D, Langlois T, Vandame B, Babon F, Boisson G, Sabater N, et al. Light field content from 16-camera rig [M40010]. ISO/IEC JTC1/SC29/WG11; 2017
- [54] Doré R, Briand G, Tapie T. Technicolor 3DoFPlus Test Materials [M42349]. ISO/IEC JTC1/SC29/WG11; 2018
- [55] Kroon B. 3DoF+ test sequence ClassroomVideo [M42415]. ISO/IEC JTC1/SC29/WG11; 2018
- [56] Fachada S, Bonatto D, Lafruit G. High-quality holographic stereogram generation using four RGBD images. *Applied Optics*. 2021;60(4):A250–A259. Publisher: Optical Society of America.
- [57] Bonatto D, Hirt G, Kvasov A, Fachada S, Lafruit G. MPEG Immersive Video tools for Light-Field Head Mounted Displays. Munich, Germany: IEEE International Conference on Visual Communications and Image Processing; 2021. p. 2
- [58] Furgale P, Rehder J, Siegart R. Unified temporal and spatial calibration for multi-sensor systems. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. Tokyo: IEEE; 2013. pp. 1280-1286
- [59] Bradski G. The open CV library. *Dr Dobb's Journal: Software Tools for the Professional Programmer*. 2000;25(11): 120–123;Publisher: Miller Freeman Inc.
- [60] Quan L, Lan Z. Linear N-point camera pose determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1999;21(8): 774-780
- [61] Griwodz C, Gasparini S, Calvet L, Gurdjos P, Castan F, Maujean B, et al. AliceVision Meshroom: An open-source 3D reconstruction pipeline. *Proceedings of the 12th ACM Multimedia Systems Conference - MMSys '21*. ACM Press; 2021
- [62] Rogge S, Bonatto D, Sancho J, Salvador R, Juarez E, Munteanu A, et al. MPEG-I Depth Estimation Reference Software. In: 2019 International Conference on 3D Immersion (IC3D). Brussels, Belgium: IEEE; 2019. pp. 1-6
- [63] Mieloch D, Stankiewicz O, Domański M. Depth Map Estimation for Free-Viewpoint Television and Virtual Navigation. *IEEE Access*, Conference Name: IEEE Access. 2020;8:5760-5776
- [64] Mieloch D, Dziembowski A. Proposal of IVDE 3.0 [m55751]. ISO/IEC JTC1/SC29/WG11; 2020