OpenDIBR: Open Real-Time Depth-Image-Based Renderer of Light Field Videos for VR

Julie Artois[1*], Martijn Courteaux[1], Glenn Van Wallendael[1] and Peter Lambert[1]

[1]IDLab-MEDIA, Ghent University - imec, Ghent, Belgium.

*Corresponding author(s). E-mail(s): julie.artois@ugent.be;

Abstract

In this work, we present a novel light field rendering framework that allows a viewer to walk around a virtual scene reconstructed from a multi-view image/video dataset with visual and depth information. With immersive media applications in mind, the framework is designed to support dynamic scenes through input videos, give the viewer full freedom of movement in a large area, and achieve real-time rendering, even in Virtual Reality (VR). This paper explores how Depth-Image-Based Rendering (DIBR) is one of the few state-of-the-art techniques that achieves all requirements. We therefor implemented OpenDIBR, an Openly available DIBR, as a proof of concept for the framework. It uses Nvidia's Video Codec SDK to rapidly decode the color and depth videos on the GPU. The decoded depth maps and color frames are then warped to the output view in OpenGL. Each input contribution is blended together through a per-pixel weighted average depending on the input and output camera positions. Experiments comparing the visual quality conclude that OpenDIBR is, objectively and subjectively, similar to TMIV and better than NeRF. Performancewise, OpenDIBR runs at 90Hz for up to 4 full HD input videos on desktop, or 2-4 in VR, and there are options to further increase this by lowering the video bitrates, reducing the depth map resolution or dynamically lowering the number of rendered input videos.

Keywords: Light field rendering, View synthesis, Depth-image-based rendering, Real time, Virtual Reality

1 Introduction

For applications like teleconferencing, education, medical treatment and entertainment, immersive technologies such as holographic displays or Virtual Reality (VR) prove to be more effective than watching simple images or videos on regular displays [1, 2]. The multimedia industry is however held back by the limited realism and computational performance that current systems achieve. Especially for VR, the frameworks need to be well optimized to render new views at high refresh rates (e.g. 90Hz) and resolutions for both eyes [3]. This work focuses on light field rendering, which takes camera captures as input and synthesizes new views for a user walking around the scene with six degrees of freedom (6DoF).

For natural content, most state-of-the-art view synthesizers preprocess the image/video dataset to estimate the intrinsics and extrinsics of the cameras. Structure-from-Motion (SfM) generally results in good estimates with low effort if the number of input views is sufficient [4, 5]. This step inherently produces a sparse reconstruction of the scene. Dense reconstructions, in the form of depth maps, point clouds and 3D meshes, can be achieved through Multi-View Stereo (MVS) and surface reconstruction algorithms [6–9].

Instead of estimating an explicit 3D mesh for the scene, Image-Based Rendering (IBR) techniques try to interpolate pixels between views, which is computationally heavier but often leads to more photorealistic imaging [10–13]. For example, a dense light field dataset can approximate the plenoptic function of a scene. However, large dense camera setups are expensive, hard to setup and require a lot of processing power. Recently, Neural Radiance Fields (NeRF) [14–17] have successfully utilized machine learning to model the plenoptic function from sparser camera setups. Still, a large number of input views is necessary before a photorealistic result is achieved. Currently, the computational performance of NeRF-based renderers are insufficient for large scenes, especially in VR.

Geometry-based IBR methods link a 3D position or depth to the image pixels, which helps with occlusions and allows for sparser datasets and more freedom of movement [18, 19]. These techniques suffer when the depth is not easily estimated, e.g. for reflections, transparent objects and textureless areas. However, several techniques have achieved near-photorealistic view synthesis by utilizing machine learning to improve depth estimation [20–22]. Due to these recent breakthroughs, we conclude that for a set of sparse camera captures, high quality depth maps can be estimated.

A subcategory of IBR is Depth-IBR (DIBR) [23], which works by converting each input depth map into a 3D mesh, warping them to the output view, and blending all input contributions together. DIBR can handle any number and placing of cameras, as long as there is enough overlap for depth estimation. An example is TMIV from MPEG-I [24], which synthesizes high quality views for dynamic content and allows for a large area of movement, but is not optimized for real-time rendering.

Several proposals have managed to further increase the visual quality by using machine learning for the blending of the warped input views [25, 26]. These techniques currently only apply to static content. Recently however, several real-time view synthesis frameworks have been built for dynamic content based on Multi-Plane Images (MPI) and Multi-Sphere Images (MSI) [21, 22, 27]. The viewer's movement is limited to a small sphere, though.

With immersive multimedia applications in mind, we are looking for a view synthesis technique that achieves the following requirements. Firstly, it should be able to display dynamic content, i.e. process input videos. This means that NeRF and many other deep learning based techniques are not viable. New views should be rendered in real time, even for the high resolutions and frame rates necessary in VR. This requires highly optimized frameworks. Lastly, the viewer should have unrestricted freedom of movement, so not just a small sphere like in Google's MSI-based framework [27].

In this work, we propose a novel DIBR-based framework, like TMIV, to comply with all these requirements. Unlike TMIV, our framework should be highly optimized for computational performance, in order to accomplish real-time VR experiences. As a proof of concept, we present OpenDIBR, an Openly available Depth-Image-Based Renderer. OpenDIBR takes as input a light field dataset, consisting of one image/video and one depth map per camera used to capture the scene. While the framework is running, the viewer watches a desktop or VR display and moves around physically or through keyboard and mouse. OpenDIBR will synthesize views accordingly at 90 frames per second.

OpenDIBR works by constructing a textured triangle mesh for each input camera The meshes are then rendered from the perspective of the viewer in OpenGL and blended together using a per-pixel weighted average. Section 3 explains the pipeline in more detail. In case of light field videos, each video is decoded in real time on the GPU using NVidia's Video Codec SDK [28]. The smaller the packet size, the faster data can be streamed to the GPU, so in this work we used the H.265/HEVC codec. OpenDIBR offers options to further increase the computational performance, as explained in Section 4.

The visual quality depends on how much of the scene was captured by the dataset and the quality of the depth maps [29, 30]. First, noisy or erroneous depth maps are annoying for the viewer, lowering the Quality of Experience (QoE). Second, disocclusions are parts of a scene which were not captured by the dataset. Typically, inpainting is necessary to fill the holes, but this is often slow and not temporally stable [31, 32]. For the purpose of this paper, no inpainting step was included in

OpenDIBR, but interfaces are in place to do so. Lastly, view-dependent elements change appearance when viewed from different angles, such as reflective or semi-transparent objects. OpenDIBR addresses this in the way it blends input triangle meshes together.

This work includes an analysis of OpenDIBR's visual and computational performance. The framework, as well as all mentioned datasets and instruction on how to replicate the results presented in this paper are available online[1]. The main contributions of this work are available in the OpenDIBR framework:

1. A real-time view synthesizer that accepts light field datasets of images or videos as input. There are no restrictions on the number of inputs and their positioning.
2. Options to improve the computational performance, meaning that large datasets can be processed while staying real time, even in VR.
3. Unrestricted freedom of movement of the viewer. The QoE depends on how well the dataset captured the scene, not on the system.
4. The implementation is openly available, which allows for comparison to other techniques.

2 Related Work

View synthesis from input images or videos is a difficult task. Scenes can get complex, requiring many cameras to capture the scene from different angles to reconstruct view-dependent objects in a plausible way and minimize disocclusions. This leads to an enormous amount of data to be processed in real time. State-of-the-art implementations have to make a trade-off between the visual quality and the performance. In this Section, we compare the properties of existing light field rendering techniques to those of OpenDIBR, as summarized by Table 1. This paper focuses on real-time view synthesis. It is hard to compare visual and computational performance against techniques without openly available implementations.

[1] https://github.com/IDLabMedia/open-dibr

	Openly available	Real-time VR	Input videos	6DoF area
Soft3D [33]			Yes	Free
Deep blending IBR [25]	Yes			Free
Welcome to light fields [20]		Yes		Limited
LLFF [34]	Yes			Limited
MSI [21, 22, 27]	Some	Yes	Some	Limited
NeRF [14–17]	Yes		Some	Free
Intel FVS [26]				Free
ColibriVR [35]	Yes	Yes		Free
Koniaris et al. [36]		Yes	Yes	Free
TMIV [24]	Yes		Yes	Free
Bonatto et al. [37]		Yes	Yes	Free
OpenDIBR (Ours)	Yes	Yes	Yes	Free

Table 1: A summary of the mentioned view synthesis frameworks. The last column indicates whether or not the viewer's movement is limited due to the technique.

2.1 Depth-image-based rendering

IBR generates output views semi-automatically by processing datasets of camera captures from real-world settings [38]. This work focuses on methods that estimate a per-view depth map, since knowing the depth of objects allows to view them from different angles using less input views.

An example of a straightforward DIBR approach is *3D warping*, where each input view contains color and depth information. The input image is then warped to the output view as a 3D point cloud or triangle mesh. All input contributions need to be blended to create the final image. This approach is used by both OpenDIBR and TMIV from MPEG-I [24]. TMIV, which is openly available, is implemented on the CPU only and does not run in real time.

Many DIBR researchers have proposed a wide variety of GPU implementations to speed up the view synthesis. Do et al. [39], and later Yao et al. [40], deliver high quality view synthesis through color correction, depth filtering, inverse warping and inpainting, accelerated through CUDA. They show that there are significant performance benefits to using the GPU, but they do not achieve real-time performance if the decoding of the videos is taken into account. Almost no framework is optimized enough to render video content in real time for a use case a demanding as VR. One of the few is the framework by Bonatto et al. [37, 41]. Just like OpenDIBR, the implementation takes H.265/HEVC videos as input, sends

the demuxed packets to the GPU where they are decoded, 3D warped and blended in OpenGL. They report a frame rate of 90Hz in VR on the Oculus Rift (resolution of 1200 × 1080 per eye) for three 2048 × 1088 input views with a GTX 1080TI GPU. Their implementation is not openly available and is limited to a very low number of input views. OpenDIBR offers several options to improve the computational performance, with as trade-off a lower visual fidelity.

One of the few implementations that is openly available is ColibriVR [35]. This tool helps to capture datasets, estimate camera parameters and depth, and synthesize views using 3D warping and blending similar to OpenDIBR. Because of the integration in Unity, the tool gives a lot of freedom and application potential. It also supports real-time rendering and VR, but it only works for static content.

Koniaris et al. [36] report real-time performance at 90Hz in VR with their highly optimized view synthesis framework. For six 1024×1024 360° input cameras, dynamic visual and depth information is decompressed on the GPU and used to render the new view. At runtime, the number of used inputs can be decreased to reduce frame drops. Due to the low resolution, their reconstruction quality decreases in areas with a lot of fine detailed geometry.

2.2 View-dependent elements

Current depth estimation techniques are a long way from reaching the accuracy needed for DIBR. View-dependent objects, whose appearance changes when viewed from different angles, are also notoriously hard for DIBR techniques. Several researchers have published methods to mitigate these shortcomings.

Overbeck et al. [20] propose a setup of densely placed cameras across the surface of a sphere. The hundreds of input views allow for a plausible reconstruction of the view-dependent elements. Their system cannot support videos, though, and the viewer's movement is restricted to a small sphere. For these reasons, the other frameworks discussed in this Section rely on less densely-packed camera setups.

Soft3D [33] is an end-to-end pipeline, from the depth estimation to the view synthesizer. It uses a representation that models uncertainty about the depth. The depth is iteratively refined through depth estimation passes, leading to occlusion-aware soft depth maps. The view synthesizer has smooth continuity as the viewer moves around, as well as plausible renderings of challenging content. The implementation is not real-time, although the authors remark that the algorithm would benefit from running massively in parallel on a GPU.

The quality of DIBR techniques is highly dependent on the available depth maps. Many of the cited sources incorporate their own depth estimation tool, which leads to their success in visual quality. In this paper, the depth maps of the real-world datasets were generated using DERS [42] as provided by MPEG-I. As with most depth estimation techniques, the resulting depth is inaccurate for texture-less areas and the temporal consistency is lacking. The alternative is to use depth sensors [43], but they cannot generate depth beyond a certain radius.

2.3 Deep learning for light field rendering

Many research groups have advocated for the use of deep learning to improve the quality of light field renderers. For example, Hedman et al. [25] use geometry-based IBR with a novel Deep Blending strategy. Instead of hand-crafted heuristics, deep learning models are trained to blend the input contributions into the output image. The openly available implementation delivers high-quality results but has a low computational performance and does not work for input videos.

Riegler and Kolton [26] propose an encoder-decoder approach. First, a proxy geometry is created through SfM, MVS and a Delaunay-based surface reconstruction. During view synthesis, a set of input views is encoded by a convolutional neural network into features, which are then mapped onto the output view (via the proxy geometry) and blended through a recurrent Neural Network (NN). This technique shows promising results, but it currently only works for static content.

Local Light Field Fusion (LLFF) [34] uses machine learning to turn each input image into a MPI and blend adjacent MPIs to synthesize new views. Their visual results are promising, but this

machine learning approach suffers from generalization, slow performance, and it does not support input videos.

So far, the mentioned systems utilizing deep learning assumed that the NNs were pre-trained on large datasets of different scenes. However, in the case of NeRF [14], a NN is trained from scratch for every scene separately. The NN learns to output a density and view-dependent emitted radiance at any 3D position in a bounded volume. An output view is then rendered by querying the NN along each pixel ray for density and color information. It does not support video inputs and does not render in real time. A lot of research has been done to mitigate this. Dynamic NeRF [15] can render videos, but still runs slow. Meanwhile Hedman et al. [16] deliver real-time rendering but only for static input images. Müller et al. [17] reduced the training time significantly through a GPU-optimized implementation. Overall, NeRF delivers continuous and plausible results even for challenging scenes, but the existing implementations are far from real-time for VR, especially for video inputs, and the quality suffers when there are only a low number of input views.

In the previous systems, the use of NNs slowed down the performance to improve quality, but this does not have to be the case. Zhou et al. [44] first introduced the MPI representation, where a scene is reconstructed as layered image planes to allow for easy view synthesis. The implementation uses a NN that was pre-trained on Youtube videos to construct the MPIs. Later, Broxton et al. [27] and Attal et al. [21] replaced the planes by spherical meshes, resulting in MSI. They also extended the technology to allow for input videos instead of just still frames. MSIs are incredibly efficient to render, reaching high computational performance. The visual quality is also promising, but the movement of the user is limited to a small sphere due to the MSI representation. This is addressed by estimating intermediary MSIs from existing MSIs by Li et al. [22], but the performance in VR is unclear.

In conclusion, OpenDIBR is the only view synthesis framework that delivers all four characteristics:

1. Both images and videos are accepted as input.
2. The system can be tuned to render views in real time, even in VR and for larger datasets.
3. The freedom of the viewer is only limited by how well the dataset captured the scene, not by the system itself.
4. The implementation is openly available, which allows for comparison to other techniques.

3 Proposed DIBR View Synthesis

3.1 Input

The proposed system allows the viewer, through a desktop or through VR, to walk around a scene of choice with 6DoF. Visual and depth information about the scene is fed to the system in the form of camera captures, either PNG images or H.265/HEVC videos. Each input frame has a corresponding depth map frame with the same resolution. Lastly, the system takes a JSON file as input, describing the intrinsics and extrinsics of each camera in the dataset.

OpenDIBR accepts datasets that use the perspective, equirectangular or fisheye equidistant projection. The bit depth (per channel) of the color and depth images/videos should lie between 8 and 16. Formula (1) shows how to convert from depth in meters to a n-bit depth value, with $near$ and far being the user-defined lower and upper bound of the depth in meter. The n-bit depth value is what is stored in the input images/videos.

$$depth_{n\_bit} = (2^n - 1)\frac{\frac{1}{depth_{meters}} - \frac{1}{far}}{\frac{1}{near} - \frac{1}{far}} \quad (1)$$

Note that this mapping is non-linear. Close to $near$, the depth map precision is at its highest but quickly falls off as the depth increases to far.

3.2 3D warping

Say that the input dataset consists of n video captures. At iteration i of the render loop, the i^{th} frame of each video is extracted and *3D warped* to the viewer's perspective. As demonstrated in Fig. 2, each input image pixel is mapped to a 3D point according to the depth map and camera intrinsics and extrinsics. For each of the n inputs, its 3D points are projected onto the output camera image plane, resulting in n images that need to be blended together into the final output image.

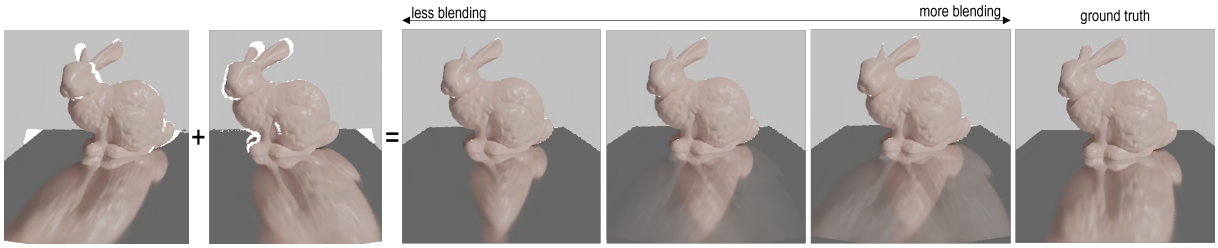

Fig. 1: 3D warping two input images results in the two images on the left. These need to be blended together to create the final output image, of which the ground truth is shown on the right. The middle three images show OpenDIBR's results using different levels of blending. Note that this illustration is an extreme case which can be mitigated by adding intermediate views and/or making the depth maps take the reflection into account.

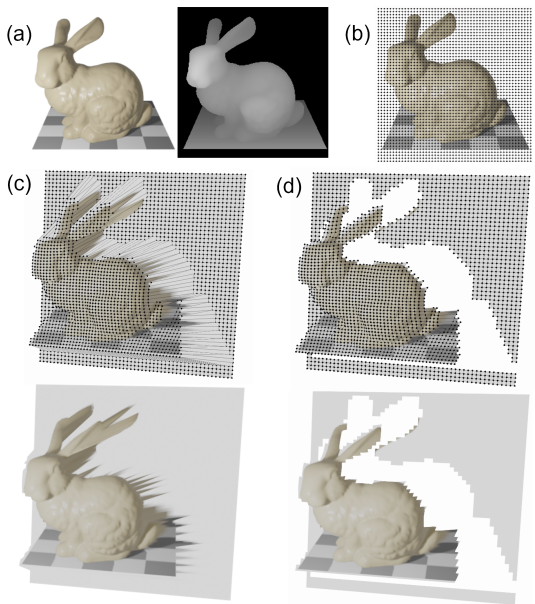

Fig. 2: Example of the 3D warping process. (a) An input image and depth map. (b) A 3D mesh is built by mapping the pixels to vertices in 3D space, using the depth, camera intrinsics and extrinsics. (c) The mesh is turned slightly. (d) The elongated triangles that connect foreground to background are deleted.

However, triangle meshes are used instead of unconnected 3D points. This means that each image pixel is associated with a 3D vertex and neighboring pixels form connected triangles, see Fig. 2(c). This decision was made to prevent scenarios in which the viewer takes a closer look and sees all the empty space between the unconnected points. Having triangles that cover this empty space significantly improves the QoE.

When there is a jump in depth values between adjacent pixels, the corresponding triangles connect foreground to background elements. These elongated triangles are deleted in the geometry shader, as illustrated in Fig. 2(d). The threshold for deletion is based on the maximum difference in depth between the three vertices of the triangles and can be fine-tuned by the user. The threshold becomes less strict at greater depths, since the depth maps lose precision close to the far plane.

3.3 Blending warped inputs

After 3D warping all n input frames to the output camera's perspective, they need to be blended together into the final output image. In OpenDIBR, high-depth pixels are overwritten by lower-depth pixels. If pixels have more or less the same depth, a weighted average is taken. The choice of the weights determines the look of view-dependent elements such as reflections, as illustrated in Fig. 1.

OpenDIBR calculates the weights as follows by default, but this approach can be changed as desired. Say that input camera I has a pixel that was mapped to 3D point P which is projected onto output camera O. The weight increases as the angle $\widehat{IPO}$ decreases. How much this weight depends on angle $\widehat{IPO}$ can be tuned by the user, as illustrated on Fig. 1 through the arrow from less to more blending.

4 Speed optimizations

The larger the light field dataset, the better the QoE conveyed to the viewer. Higher resolutions and bit depths improve the level of detail, while larger fields of view and more input cameras lead to more accurate view-dependent elements, less disocclusions and more freedom of movement. However, the computational performance of a 6DoF system is inversely proportional to the size of the dataset, assuming that we never run out of memory. For example, $n \times 1920 \times 1080$ input images require $> 4n$ million triangles to be drawn and twice the number of texture lookups to be performed every frame. In the case of VR, this needs to happen once per eye at a rate faster than 90 Hertz (Hz). The $2n$ textures also need to be updated 30 times per second. Even high-end systems will not achieve this without serious system optimizations.

As OpenDIBR targets real-time 6DoF experiences from large datasets at high refresh rates and resolutions, the following optimizations were made.

1. The 3D warping of every input pixel onto an output pixel can be executed massively in parallel on the GPU using OpenGL. The 3D warping of n input images takes up n draw calls, plus an additional $n-1$ quick draw calls for the blending. To limit memory use, a pool of framebuffers is reused.
2. The input images and depth maps need to be sent to the VRAM for the GPU to process them. In the case of video content, the VRAM cannot contain all videos entirely, so they are streamed frame by frame. To avoid a bottleneck between disk and VRAM, the input videos were compressed using H.265/HEVC beforehand. While OpenDIBR is running, the CPU demuxes the videos, sending packets to the GPU, where they are decompressed using NVidia's Video Codec SDK. CUDA is used to copy the resulting decompressed frames to the OpenGL texture buffers.
3. A thread pool is used to parallelize the demuxing workload across the CPU's cores, if applicable.

OpenDIBR has additional settings that allow to increase performance, sometimes at the cost of visual quality. This is a trade-off the user has to make.

1. OpenDIBR allows to reduce the number of triangles that are drawn for each input image. For example, by cutting the detail level in half along the width and height, the computational performance will be up to four times higher. For datasets where the triangles are small due to high resolutions and low field of views, the visual difference is negligible. The worst case scenario is shown in Fig. 3, where the input image comes from a 360° camera, which results in noticeably larger triangles around the equator because of the stretched-out equirectangular projection.

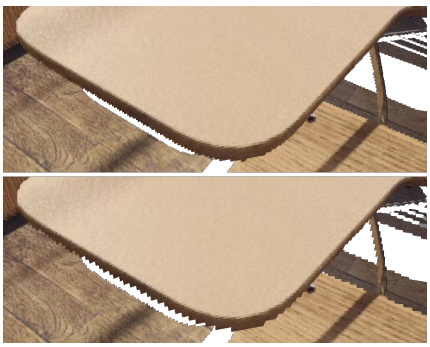

Fig. 3: Two outputs rendered by OpenDIBR from one 4096x2048 360° input camera of Classroom. (top) No speed optimizations are used. (bottom) The number of triangles is reduced to 25% by halving the detail level of the triangle mesh along the width and height. This demonstrates the worst case scenario of 360° input cameras.

2. The user has control over the maximum number of inputs that need to be processed at the same time. In this paper, this feature is denoted as *input culling*, see Fig. 4. Instead of processing all n inputs every render loop, only m inputs ($m < n$) can be used at any moment in time. Alg. 9 explains the heuristic approach used by OpenDIBR to decide which m inputs will result in the best QoE given the current output camera position and rotation. The heuristic prioritizes reducing the need for inpainting first and the correctness of view-dependent elements second.

Input culling is the key to rendering large datasets in real time. It does come at the cost of lowering the QoE, since using less input cameras means more inpainting is necessary and view-dependent

elements are less accurate. It also results in triangle meshes popping in and out of view as the choice of the m inputs changes. Note that in the case of video content, all n inputs still need to be decoded every frame, even though only m of them will be used.

Algorithm 1 Input culling from n to m input cameras

Input: output camera O with field of view (fov_x, fov_y) and model matrix M, far plane d, list L of n input cameras
Output: set F of $m < n$ input cameras

```
Sort L from smallest to largest angle between the looking
  direction of each input camera and the looking direction of O
For the 4 corners of O's image, define a 3D point at depth d
  and store them in list C
  c[0] = (−tan(fov_x/2)d, +tan(fov_y/2)d, −d, 1)
  c[1] = (+tan(fov_x/2)d, −tan(fov_y/2)d, −d, 1)
  c[2] = (+tan(fov_x/2)d, +tan(fov_y/2)d, −d, 1)
  c[3] = (−tan(fov_x/2)d, −tan(fov_y/2)d, −d, 1)
Left-multiply each point in C by M
foreach c ∈ C do
    foreach i ∈ L do
        if i sees c then
            add i to F
while #F < M do
    remove L[0] from L and append to F
```

Fig. 4: Four outputs synthesized by OpenDIBR with (a)(c) and without (b)(d) input culling from four input views. (b) All cameras look in the same direction, resulting in only a few disocclusions. (d) There are more disocclusions, because the input cameras are more spread out and all look in different directions.

5 Visual Quality

5.1 Light field datasets

Throughout this work, several datasets are used as input for OpenDIBR and other view synthesizers. Each dataset contains the video captures of a number of cameras and the corresponding depth map videos. *Fan*, *Kitchen*, *Frog*, *Painter*, *Museum* and *ClassroomVideo* (here referred to as *Classroom*) are provided by the MPEG-I community [45]. Before being fed to OpenDIBR, each video is encoded by H.265/HEVC. *Barbershop_c*, *Zen_Garden_c*, *Barbershop_s* and *Zen_Garden_s* are derived from the datasets of Courteaux et al. [46]. The "_c" and "_s" suffixes indicate the cuboid or spherical arrangement of cameras respectively. To allow for comparisons with the other view synthesis methods, the images of the "_c" datasets are converted from a fisheye to a perspective projection.

Fan, Frog and Painter contain real-world images, from which depth maps were estimated by DERS [42]. The other datasets are computer-generated. The camera intrinsics, extrinsics, file formats, compression, etc. can be found online[1].

5.2 Objective quality measurements

To calculate objective metrics on output images, a ground truth is needed. For several datasets, we selected six images/videos to function as the ground truth output. In other words, OpenDIBR generated the six outputs using only the remaining videos as inputs. Fig. 5 illustrates how the cameras of the datasets were split into inputs and ground truth outputs. The results are summarized in Table 2.

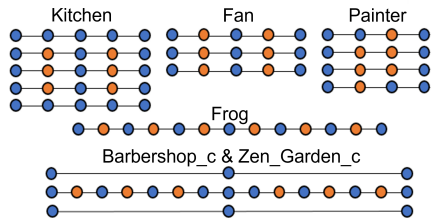

Fig. 5: Illustration of the input (blue) and output (orange) cameras used per dataset during the objective quality assessment.

The Table contains well-known metrics such as PSNR (in this case only for the luma channel), Video Multimethod Assessment Fusion (VMAF) [47] and Structural Similarity (SSIM) [48]. MPEG-I's IV-PSNR [49] makes two alterations to PSNR to be better suited for objective quality comparisons in the context of immersive video applications. LPIPS [50] uses learned deep features to deliver a perceptual similarity metric, with a *lower* score indicating a *higher* quality. We noticed that the LPIPS results in Table 2 more accurately estimate the actual visual quality compared to the other four metrics.

The comparison is made against two methods that each have a relatively high visual quality in their sub-category. The first is TMIV [24], since it is a DIBR framework which uses 3D warping and blending very similar to OpenDIBR. The second is NeRF [17], which is a more general IBR technique that models the plenoptic function. In this way, researchers can extrapolate how OpenDIBR compares to a wider range of DIBRs and plenoptic function modelers.

There are two factors that complicate the comparison between the three techniques. Firstly, OpenDIBR and TMIV depend on the dataset's depth maps, while NeRF does not. This leads to NeRF outputting blurry images, as it struggles to model the depth accurately with the limited number of available inputs. However, in cases where the provided depth maps are incorrect or misleading, NeRF performs better. One example of this is the mirror of Barbershop_c, which has a flat depth but NeRF does not model it this way.

Secondly, OpenDIBR outputs images that still need to be inpainted, as opposed to NeRF and TMIV. The results in Table 2 were calculated by setting the pixels that still need to be inpainted to the ground truth value for all three techniques. Since the impact of the inpainting techniques of NeRF and TMIV on the quality is removed, the comparison is fairer. However, overwriting pixels to ground truth values can influence the metric scores. In this paper that impact is rather small since only a small percentage of the pixels warrants inpainting, as shown in Table 3.

From Table 2 we conclude that TMIV generally scores best. This makes sense since TMIV has a triangle mesh refinement stage that results in cleaner edges when different depth maps disagree. This could also have been included in OpenDIBR, but was omitted to improve the computational performance. Due to blurry results, NeRF generally scores worse than OpenDIBR.

		psnr-y	iv-psnr	vmaf	ssim	lpips
Fan	NeRF	28.02	35.72	43.83	0.9459	0.2292
	TMIV	**30.36**	40.69	62.36	**0.9818**	**0.0524**
	Ours	29.05	**41.26**	**65.90**	0.9804	0.0739
Kitchen	NeRF	31.49	37.35	66.13	0.9795	0.1961
	TMIV	**34.16**	**42.65**	73.40	**0.9901**	**0.0571**
	Ours	32.15	42.45	**75.32**	0.9868	0.0617
Frog	NeRF	28.19	37.88	46.67	0.9430	0.2476
	TMIV	**29.17**	**39.69**	**59.29**	**0.9646**	0.1248
	Ours	27.39	37.73	54.03	0.9537	**0.1200**
Painter	NeRF	34.09	42.56	83.00	0.9821	0.2343
	TMIV	**37.74**	**47.81**	**83.87**	**0.9917**	0.1023
	Ours	36.27	47.23	82.71	0.9883	**0.0715**
Barber-shop_c	NeRF	**31.88**	**39.17**	**43.38**	**0.9769**	0.2735
	TMIV	28.72	35.28	35.71	0.9378	0.1428
	Ours	28.27	35.37	39.50	0.9378	**0.1095**
Zen_ Garden_c	NeRF	31.07	37.39	22.66	0.9661	0.2337
	TMIV	26.78	33.94	36.41	0.9613	0.1101
	Ours	**32.46**	**41.61**	**53.79**	**0.9851**	**0.0739**

Table 2: Objective quality metrics calculated by comparing the ground-truth of six cameras per dataset to the outputs of NeRF, TMIV and OpenDIBR.

Maximum percentage of inpainted pixels	
Fan	1.5%
Kitchen	0.4%
Frog	1.2%
Painter	0.4%
Barbershop_c	0.3%
Zen_Garden_c	0.5%

Table 3: The upper bound of the percentage of pixels that need inpainting per dataset used in Table 2.

5.3 Subjective quality

The datasets mentioned in this paper can be downloaded to run OpenDIBR locally through a display or a VR HMD[1]. The performance optimization mentioned in Section 4 can be toggled to see the effects on the QoE. The conclusions on the subjective quality in this Section are based on informal viewing tests. Fig. 6 contains crops of some of the images used in the qualitative comparison of Table 2.

Fig. 6: For the creaton of Table 2, NeRF, TMIV and OpenDIBR each rendered six views from different light field datasets. Here, one of the six output views is shown (cropped) for datasets Barbershop_c, Frog, Painter, Fan and Kitchen. The bright green pixels are holes that need to be inpainted.

The quality of the DIBR techniques such as OpenDIBR is highly dependent on the available depth maps. For example, a lack of temporal consistency in the depth maps results in flickering geometry. Moreover, view-dependent elements such as reflections and semi-transparent objects have an ambiguous depth. For example, the mirror in Barbershop_c has a flat depth, resulting in overlapping reflections typical for DIBR, as shown in Fig. 6. If the depth maps use the depth of the reflection instead, the results will look more realistic. This is what NeRF does. However, if the user is allowed to move behind the mirror, they will see parts of the reflection floating around, which is not realistic.

TMIV and OpenDIBR do not lose the fine geometry of the Fan sequence like NeRF does. NeRF's output is blurry for the used datasets, mainly because of the low number of input views. The parameters used to encode the input videos also influence the QoE. Lower bitrates improve computational performance because the smaller packet sizes mean that they can be sent to VRAM faster. However, compression artefacts in the videos can be noticeable in the OpenDIBR output, especially in VR.

6 Computational Performance

To evaluate the value of OpenDIBR for certain real-time rendering use cases, different measurements were performed on a Windows desktop with a i9-9900K CPU, 8 cores and a RTX 2080 Ti. As input, a variety of light field datasets is used, with input camera resolutions going from 1920x1080 up to 4096x2048. The bit depth of the color and depth videos are 8 and 16 respectively. The computational performance is expressed in number of frames per second (fps) being rendered. The fps is measured and averaged over a duration of 30 seconds while the viewer moves around the scene. The output is rendered for a 1920x1080 display or a HTC Vive Pro with a per-eye resolution of 2016x2240.

A problem with benchmarking the performance in VR is that SteamVR forces the application to use Vsync at a specific refresh rate, in this case 90Hz. This means that a new frame needs to be ready to be displayed every 11.1111 milliseconds (ms). If rendering a frame takes slightly longer than that, the result is that the actual fps drops down from 90 to 45 immediately.

Accurately reporting the fps for *dynamic* content, where the input videos are played out and looped, also proves challenging. If we assume that every frame should take the same amount of time and that the frame rate of the video itself is 30 fps, we restrict OpenDIBR to run at an fps that is a multiple of 30, e.g. 60, 90 or 120 fps.

6.1 Without speed optimizations

Table 4 and 5 show the average fps of OpenDIBR for static and dynamic content. For static content, only one frame of each input video needs to be decoded and is reused for every output frame. Note that these Tables project the worst case computational performance, i.e. there are no speed optimizations.

	Average frames per second							
	Full-HD display				VR (Vsync 90Hz)			
Nr. inputs	1	2	4	8	1	2	4	8
Fan	833	490	271	139	90	90	90	<90
Kitchen	690	468	252	133	90	90	90	<90
Frog	801	463	250	129	90	90	90	<90
Painter	770	442	241	125	90	90	90	<90
Zen_Garden_s	515	296	155	81	90	90	<90	<90
Barbershop_s	514	284	148	75	90	90	<90	<90
Museum	460	261	138	73	90	90	<90	<90
Classroom	252	133	68	34	90	<90	<90	<90

Table 4: Average frames per second for *static* content.

	Average frames per second							
	Full-HD display				VR (Vsync 90Hz)			
Nr. inputs	1	2	4	8	1	2	4	8
Fan	>840	360	180	60	90	90	90	<90
Kitchen	>840	360	150	60	90	90	<90	<90
Frog	>840	360	150	<30	90	90	<90	<90
Painter	>840	360	150	<30	90	90	<90	<90
Museum	240	240	90	<30	90	90	<90	<90
Classroom	240	120	60	<30	90	<90	<90	<90

Table 5: Average frames per second for *dynamic* content.

6.2 With speed optimizations

Table 4 and 5 only go up to 8 input cameras, while all these datasets actually contain tens or even hundreds of input cameras. There are several ways to improve the performance so that more inputs can be processed while maintaining a high fps. Table 6 illustrates the effect of these improvements on the Painter dataset, which holds 16 2048×1088 input views.

Average frames per second (16 input views, 2048x1088 output, no VSync)	Default	Input culling	Half triangle mesh dimensions
Static	56	110	136
Lossless depth (190Mbps)	13	13	13
Lossy depth (3Mbps)	36	41	41

Table 6: Average frames per second for Painter, with 16 2048×1088 input views. The first row denotes static content, the others are (dynamic) video content. For input culling, OpenDIBR dynamically selects 8 out of 16 input views to render each output frame.

Firstly, switching from lossless to lossy compression for the depth videos boosts the fps from 13 to 36. The smaller video packet sizes mean that the streaming of packets from RAM to VRAM can happen at a much higher rate. The impact on the visual difference is negligible for datasets like Painter that have a noisy depth map. Note that if Table 4 and 5 used lossy compression for the depth maps, the reported fps would lie much higher.

Secondly, the speed optimizations at the end of Section 4 can be applied to reduce the number of triangles that need to be processed in the render pipeline. By input culling from 16 to 8 inputs or by halving the triangle mesh dimensions, the fps further increases from 36 to 41. However, the largest jump in performance happens for static content, from 56 to 110 and 136 fps. As mentioned before, these speed optimizations do not noticeably degrade the visual quality for datasets like Painter, where the cameras have a low field of view and all look in the same direction.

6.3 Comparison to other techniques

In the same conditions as Table 6, i.e. 16 input views from Painter to 2048x1088 output images, TMIV [24] renders at 13.5 seconds per frame and NeRF [17] at 24 seconds per frame. Note that for NeRF, the default settings were applied and that the time to train the model was not included. As mentioned in Section 2.1, the framework by Bonatto et al. [37] is the closest to OpenDIBR in terms of implementation. They report a frame rate of 90Hz in VR on the Oculus Rift (resolution of 1200×1080 per eye) for three 2048×1088 input views with a GTX 1080TI GPU. This is similar to the worst case performance of OpenDIBR, but detailed testing could not be done since their framework is not openly available.

7 Conclusion and future work

In this work, we presented OpenDIBR, an openly available view synthesizer that accepts images/videos light field datasets plus depth as input (any number and arrangement of input cameras is allowed) and renders output views in real time without restrictions on the movement of the viewer. By default, there is no inpainting, but since the framework is openly available, the desired inpainting technique can be added if necessary. The best QoE is achieved for datasets that capture a large portion of the scene to minimize disocclusions.

To assess the visual quality and computational performance, a comparison was made to NeRF by Müller et al. [17] and TMIV [24]. OpenDIBR is highly optimized in order to deliver real-time performance even in performance-heavy cases such as VR, as measured and summarized in Tables 4, 5 and 6. OpenDIBR greatly outperforms TMIV and NeRF, and performs similarly to the framework by Bonatto et al. [37]. Table 6 shows how input culling and reducing the detail level of the triangle mesh can further improve the performance.

In most cases, the visual quality of OpenDIBR is slightly lower than that of TMIV, since the latter adds a step to improve the geometry of the triangle meshes. However, TMIV is significantly slower than OpenDIBR. The quality of NeRF is often lower than OpenDIBR because the datasets used in this paper have a low number of input views, resulting in a blurry reconstruction of the scene. However, in the case of incorrect/ambiguous depth maps, for example for objects that change appearance when viewed from different angles, NeRF delivers more accurate output images than DIBR-based techniques. OpenDIBR can mitigate this to some extend by processing more input views.

In future work, to allow for larger datasets to run at 90Hz in VR, two performance bottlenecks can be addressed. Currently, two triangles are used for each pixel of each depth map. This is unnecessary for parts of the scene with little geometrical complexity. In the future, we would replace the depth maps with pre-defined, dynamic triangle meshes, which are decompressed on the

GPU in real time. This will result in less triangles that need to be rendered, as well as making the deletion of stretched triangles in the geometry shader obsolete.

Another benefit of this approach would be to pre-generate an additional triangle mesh to perform the function of inpainting, i.e. to fill the parts of the screen for which none of the input views holds information. This approach to inpainting would be low-cost in terms of performance and would prove beneficial towards the QoE.

Declarations

Funding

This work was funded in part by the Research Foundation – Flanders (FWO) under Grant 1SD8221N, in part by IDLab (Ghent University – imec), Flanders Innovation and Entrepreneurship (VLAIO), and the European Union.

Data

The datasets generated during and/or analysed during the current study are available at https://cloud.ilabt.imec.be/index.php/s/z4bm23cy2ineApS. Instructions and more information can be found at https://github.com/IDLabMedia/open-dibr/wiki.

Code availability

The OpenDIBR framework is available at https://github.com/IDLabMedia/open-dibr.

References

[1] Biocca F, Delaney B (1995) Immersive virtual reality technology. Communication in the age of virtual reality 15(32):10–5555

[2] Baños RM, Botella C, Alcañiz M, et al (2004) Immersion and emotion: their impact on the sense of presence. Cyberpsychology & behavior 7(6):734–741

[3] Geršak G, Lu H, Guna J (2020) Effect of vr technology matureness on vr sickness. Multimedia Tools Appl 79(21–22):14,491—14,507

[4] Bolles RC, Baker HH, Marimont DH (1987) Epipolar-plane image analysis: an approach to determining structure from motion. Internat Journal Comput Vis 1:7–55

[5] Schönberger JL, Frahm JM (2016) Structure-from-motion revisited. In: IEEE Conf. Comput. Vis. Pattern Recognition, pp 4104–4113

[6] Seitz SM, Curless B, Diebel J, et al (2006) A comparison and evaluation of multi-view stereo reconstruction algorithms. In: IEEE/CVF Conf. Comput. Vis. Pattern Recogn. (CVPR), pp 519—-528

[7] Schönberger JL, Zheng E, Frahm JM, et al (2016) Pixelwise view selection for unstructured multi-view stereo. In: 16th Europ. Conf. Comput. Vis. (ECCV). Springer, Cham, pp 501–518

[8] Kazhdan M, Bolitho M, Hoppe H (2006) Poisson surface reconstruction. In: Proc. fourth Eurograph. Sympos. Geometry Processing, vol 7. Eurographics Association, Goslar, DEU, p 61–70

[9] Field DA (1988) Laplacian smoothing and delaunay triangulations. Communications in Applied Numerical Methods 4(6):709–712

[10] Chen SE, Williams L (1993) View interpolation for image synthesis. In: 20th Annu. Conf. Comp. Graph. Interact. Techn. ACM, New York, NY, USA, pp 279–288

[11] Levoy M, Hanrahan P (1996) Light field rendering. In: 23rd Annu. Conf. Comp. Graph. Interact. Techn. ACM, New York, NY, USA, pp 31—42

[12] Gortler SJ, Grzeszczuk R, Szeliski R, et al (1996) The lumigraph. In: 23rd Annu. Conf. Comp. Graph. Interact. Techn. (SIGGRAPH). ACM, New York, NY, USA, pp 43—54

[13] Chan SC (2021) Image-Based Rendering, Springer, New York, NY, USA, pp 656–664

[14] Mildenhall B, Srinivasan PP, Tancik M, et al (2020) Nerf: Representing scenes as neural radiance fields for view synthesis. In: Eur. Conf. Comput. Vis. (ECCV)

[15] Pumarola A, Corona E, Pons-Moll G, et al (2021) D-nerf: neural radiance fields for dynamic scenes. In: IEEE/CVF Conf. Comput. Vis. Pattern. Recogn. (CVPR), pp 10,318–10,327

[16] Hedman P, Srinivasan PP, Mildenhall B, et al (2021) Baking neural radiance fields for real-time view synthesis. In: IEEE/CVF Internat. Conf. Comput. Vis. (ICCV), pp 5855–5864

[17] Müller T, Evans A, Schied C, et al (2022) Instant neural graphics primitives with a multiresolution hash encoding. ACM Trans Graph 41(4):102:1–102:15

[18] Debevec PE, Taylor CJ, Malik J (1996) Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In: Proc. 23rd Annu. Conf. Comp. Graph. Interact. Techn. (SIGGRAPH). ACM, New York, NY, USA, pp 11—-20

[19] Buehler C, Bosse M, McMillan L, et al (2001) Unstructured lumigraph rendering. In: Proc. 28th Annu. Conf. Comp. Graph. Interact. Techn. (SIGGRAPH '01). ACM, New York, NY, USA, pp 425—432

[20] Overbeck RS, Erickson D, Evangelakos D, et al (2018) Welcome to light fields. In: ACM SIGGRAPH Virtual, Augmented, Mixed Reality. ACM, New York, NY, USA

[21] Attal B, Ling S, Gokaslan A, et al (2020) Matryodshka: Real-time 6dof video view synthesis using multi-sphere images. In: Europ. Conf. Comput. Vis. (ECCV). Springer, Cham, pp 441–459

[22] Li J, He Y, Jiao J, et al (2021) Extending 6-dof vr experience via multi-sphere images interpolation. In: 29th ACM Internat. Conf. Multimedia. ACM, New York, NY, USA, pp 4632—4640

[23] Fehn C (2004) Depth-image-based rendering (dibr), compression and transmission for a new approach on 3d-tv. Proc SPIE 5291:93—-105

[24] MPEG-I Visual (2022) Test model of mpeg immersive video (tmiv). URL https://gitlab.com/mpeg-i-visual/tmiv

[25] Hedman P, Philip J, Price T, et al (2018) Deep blending for free-viewpoint image-based rendering. ACM Trans Graph 37(6):1–15

[26] Riegler G, Koltun V (2020) Free view synthesis. In: 16th Europ. Conf. Comput. Vis. (ECCV). Springer, Cham, pp 623–640

[27] Broxton M, Flynn J, Overbeck R, et al (2020) Immersive light field video with a layered mesh representation. In: ACM Trans. Graph. (SIGGRAPH), vol 39. ACM, New York, NY, USA, pp 1–15

[28] NVidia (2022) Nvidia video codec sdk. URL https://developer.nvidia.com/nvidia-video-codec-sdk

[29] Kertész G, Vámossy Z (2015) Current challenges in multi-view computer vision. In: 2015 IEEE 10th Jubil. Internat. Sympos. Appl. Computat. Intell. Informat., pp 237–241

[30] Sun W, Xu L, Au OC, et al (2010) An overview of free view-point depth-image-based rendering (dibr). In: APSIPA Annual Summit Conf., pp 1023–1030

[31] Oh KJ, Yea S, Ho YS (2009) Hole filling method using depth based in-painting for view synthesis in free viewpoint television and 3-d video. In: 2009 Picture Coding Symposium, IEEE, pp 1–4

[32] Chen X, Liang H, Xu H, et al (2021) Disocclusion-type aware hole filling method for view synthesis. Multimedia Tools Appl 80:11,557—11,581

[33] Penner E, Zhang L (2017) Soft 3d reconstruction for view synthesis. ACM Trans Graph (SIGGRAPH Asia) 36(6):1–11

[34] Mildenhall B, Srinivasan PP, Ortiz-Cayon R, et al (2019) Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. ACM Trans Graph (TOG) 38:1–14

[35] Dinechin GDd, Paljic A (2020) From real to virtual: An image-based rendering toolkit to help bring the world around us into virtual reality. In: 2020 IEEE Conf. Virtual Reality 3D User Interfaces Abstracts and Workshops (VRW), pp 348–353

[36] Koniaris B, Kosek M, Sinclair D, et al (2017) Real-time rendering with compressed animated light fields. In: Proc. 43rd Graph. Interface Conf., vol 2. Canadian Human-Computer Communications Society, Waterloo, CAN, p 33–40

[37] Bonatto D, Fachada S, Rogge S, et al (2021) Real-time depth video-based rendering for 6-dof hmd navigation and light field displays. IEEE Access 9:146,868–146,887

[38] Zhang C, Chen T (2004) A survey on image-based rendering—representation, sampling and compression. Signal Processing: Image Communication 19(1):1–28

[39] Do L, Bravo G, Zinger S, et al (2012) Gpu-accelerated real-time free-viewpoint dibr for 3dtv. IEEE Trans Consumer Electr 58(2):633–640

[40] Yao L, Han Y, Li X (2019) Fast and high-quality virtual view synthesis from multi-view plus depth videos. Multimedia Tools Appl 78:19,325—-19,340

[41] Bonatto D, Fachada S, Lafruit G (2020) Ravis: Real-time accelerated view synthesizer for immersive video 6dof vr. Electronic Imaging 2020:382–1

[42] Stankiewicz O, Wegner K, Tanimoto M, et al (2013) Enhanced Depth Estimation Reference Software (DERS) for Free-viewpoint Television [M31518]. document ISO/IEC JTC1/SC29/WG11

[43] Xie Y, Souto AL, Fachada S, et al (2021) Performance analysis of dibr-based view synthesis with kinect azure. In: 2021 Internat. Conf. 3D Immersion (IC3D), pp 1–6

[44] Zhou T, Tucker R, Flynn J, et al (2018) Stereo magnification: Learning view synthesis using multiplane images. In: ACM Trans. Graph., vol 37. ACM, New York, NY, USA, pp 1–12

[45] Jung J, Kroon B (2022) Common Test Conditions for MPEG Immersive Video [N0232]. document ISO/IEC JTC1/SC29/WG04

[46] Courteaux M, Artois J, De Pauw S, et al (2022) Silvr: a synthetic immersive large-volume plenoptic dataset. In: 13th ACM Multimedia Systems Conf. ACM, New York, NY, USA, pp 221—226

[47] Netflix Technology Blog (2018) Vmaf: the journey continues. URL https://netflixtechblog.com/vmaf-the-journey-continues-44b51ee9ed12

[48] Wang Z, Bovik A, Sheikh H, et al (2004) Image quality assessment: from error visibility to structural similarity. IEEE Trans Image Processing 13(4):600–612

[49] Dziembowski A (2020) Software Manual of IV-PSNR for Immersive Video [N19495]. document ISO/IEC JTC1/SC29/WG11

[50] Zhang R, Isola P, Efros AA, et al (2018) The unreasonable effectiveness of deep features as a perceptual metric. In: IEEE/CVF Conf. Comput. Vis. Pattern Recogn. (CVPR), pp 586–595