HAL
open science

From Real to Virtual: An Image-Based Rendering Toolkit to Help Bring the World Around Us Into Virtual Reality

Grégoire Dupont de Dinechin, Alexis Paljic

To cite this version:

Grégoire Dupont de Dinechin, Alexis Paljic. From Real to Virtual: An Image-Based Rendering Toolkit to Help Bring the World Around Us Into Virtual Reality. 2020 IEEE 6th Workshop on Everyday Virtual Reality (WEVR), Mar 2020, Atlanta, Georgia, United States. hal-02492896

HAL Id: hal-02492896

https://hal.science/hal-02492896v1

Submitted on 27 Feb 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

From Real to Virtual: An Image-Based Rendering Toolkit to Help Bring the World Around Us Into Virtual Reality

Grégoire Dupont de Dinechin* Alexis Paljic†

Centre for Robotics, MINES ParisTech, PSL University - Paris, France

Abstract

The release of consumer-grade head-mounted displays has helped bring virtual reality (VR) to our homes, cultural sites, and workplaces, increasingly making it a part of our everyday lives. In response, many content creators have expressed renewed interest in bringing the people, objects, and places of our daily lives into VR, helping push the boundaries of our ability to transform photographs of everyday real-world scenes into convincing VR assets. In this paper, we present an open-source solution we developed in the Unity game engine as a way to make this image-based approach to virtual reality simple and accessible to all, to encourage content creators of all kinds to capture and render the world around them in VR. We start by presenting the use cases of image-based virtual reality, from which we discuss the motivations that led us to work on our solution. We then provide details on the development of the toolkit, specifically discussing our implementation of several image-based rendering (IBR) methods. Finally, we present the results of a preliminary user study focused on interface usability and rendering quality, and discuss paths for future work.

Index Terms: Computing Methodologies—Computer Graphics—Graphics Systems and Interfaces—Virtual Reality; Computing Methodologies—Computer Graphics—Image Manipulation—Image-Based Rendering;

1 Introduction

Image-based virtual reality is relevant to a wide variety of use cases, from visiting digital reproductions of cultural heritage sites to sharing homemade 360° videos in VR with family and friends. Many of these use cases have been explored by expert researchers but also casual developers and content creators, making use of low-cost camera rigs and 3D reconstruction tools to develop small-scale VR experiences in which parts of the real world are rendered virtually [26]. As hospitals start using 360° images for interactive neurosurgery training [19] and local museums investigate visitors' expectations for the integration of VR exhibits [16], it thus seems safe to say that virtual reality has never been more present in our everyday lives.

However, there appears to currently be no easy way for small-scale content creation teams to bring the real world into VR with levels of visual accuracy similar to what is typically demonstrated by the specialized research community. Several works in the field of computer graphics have indeed demonstrated advanced methods for blending photographs together to create seamless, high-resolution virtual environments [8, 15, 17, 23], a notable example being the publicly-released *Welcome to Light Fields* VR experience [17]. Unfortunately, the rendering solutions used to create such demonstrations are seldom released for public use, preventing non-expert content creators from applying these methods to their own photograph datasets. The transfer of knowledge between the two groups is also further complicated by the fact that they use different development tools: while graphics researchers most often work from the ground up with low-level OpenGL and C++, focusing on the quality and speed of their implementation, most content creators typically prefer to rely on game engines such as Unity, Godot, or Unreal, which provide convenient graphical user interfaces (GUIs) for the rapid creation of interactive content.

*e-mail: gregoire.dupont_de_dinechin@mines-paristech.fr
†e-mail: alexis.paljic@mines-paristech.fr

©2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Figure 1: From unstructured sets of photographs to interactive VR scenes providing motion parallax and view-dependent highlights. We present an open toolkit we developed to help casual content creators render real-world scenes in virtual reality.

Therefore, to help casual content creators learn more about and apply image-based rendering for the creation of interactive virtual reality experiences, we discuss in this paper our development of an open-source IBR-for-VR toolkit. The toolkit aims to provide a simple, accessible interface, able to handle a wide variety of input data, that brings together multiple different tools for transforming sets of photographs into virtual environments that are visually accurate and can be viewed comfortably in VR. In this paper, we thus aim to give a clear overview of the methods and applications of image-based virtual reality, by analyzing target use cases and sharing details on core components of our toolset, which we illustrate on several online image datasets both in the paper and the complementary video. Additionally, we provide insight into our design choices and on the lessons learned during the development of the toolkit, to encourage readers to extend and improve on our approach. Finally, we present a first evaluation of the toolkit, in the form of a user study we led in collaboration with a mineralogy museum to investigate the usability of the toolset by non-expert users.

2 Designing the toolkit

Before delving into the implementation, we start by explaining the motivations and design choices that helped guide our approach.

2.1 Motivations

2.1.1 A growing interest in image-based VR: use cases

Many VR experiences fundamentally rely on - or can be enhanced by - the use of virtual environments and assets created from sets of photographs. This field of research and applications is commonly referred to using the names *cinematic virtual reality* and *image-based virtual reality*. A sizable portion of the related research is dedicated to studying and demonstrating low-cost alternatives for creating

high-quality image-based VR experiences from casually-captured photographs and videos [1,6,12,26], to provide local content creators easy access to the corresponding knowledge and tools.

One notable group of use cases is related to education and training. For instance, photographs can help create immersive virtual tours, e.g. of cultural heritage sites [9, 16], apartments, and industrial facilities. Additionally, image-based assets can be used in VR documentaries, to inform about and create empathy for a given circumstance [7]: for example, volumetric capture technologies can be used to place immersed viewers face-to-face with real-world people, telling their account of living through a specific situation. Furthermore, photographs and videos can be used to create compelling virtual task training environments, that accurately depict real-world situations such as high-risk surgical operations [19] and firefighter interventions in dangerous conditions. Such experiences, in which multiple generations of trainees can be immersed for no additional cost and with no exposure to danger, are indeed particularly relevant when the corresponding live training is costly or dangerous.

Another important field of application is entertainment. Concerts, theater performances, and sports events can be recorded for live or delayed viewing in VR, to enable users that could not attend in person to participate in an immersive and interactive way. Additionally, virtual assets created from photographs of moments and places that one feels close to can be used for personal reminiscence and shared for family and friends to explore. Finally, volumetric captures and video recordings are also commonly used for game development and interactive VR films [5].

2.1.2 The untapped potential of image-based rendering

Methods for processing and rendering sets of photographs to display visual information for multiple viewpoints are commonly grouped under the names *novel view synthesis*, *free-viewpoint rendering*, and *image-based rendering*. We map out the general shape of image-based rendering methods in Fig. 2. The defining function of IBR is to generate novel views from an input set of photographs so that the captured scene is accurately depicted within a range of potential viewpoints. The direct application for VR is the creation of *6-DoF* VR experiences, in which viewers can not only look in all directions but also move around in the scene, i.e. that provide coherent visual information even as users move with the six degrees-of-freedom corresponding to translations and rotations.

There are two important ways in which image-based rendering research has the potential to enhance the rendering solutions commonly used by VR developers. First, recent works have demonstrated methods that enhance the rendering of 360° images to provide *motion parallax* [1, 6, 12, 23], i.e. the sensation that, during head movement, closer objects move faster than ones further away. Indeed, 360° images are most usually rendered on a virtual sphere surrounding the immersed viewer, and therefore do not provide this sense of parallax. In contrast, such works describe methods to restore a sense of parallax in a small range of head motion, thereby improving viewers' sense of presence and comfort [6, 23]. Second, research on view-dependent rendering has shed light on ways to render *specular highlights* from the input photographs, i.e. ways to accurately display, based on the viewer's current viewpoint, the bright reflections of light bouncing off the facets of the captured object [2, 4, 8, 17]. This is a notable improvement over rendering a 3D-reconstructed object solely with a diffuse texture map, which is the simplest solution accessible to most content creators, yet is unadapted to rendering objects with specular surfaces and complex lighting properties. Therefore, having access to these methods could enable casual developers to provide more comfortable and visually accurate virtual environments, both of which may help strengthen the application's impact: for instance, these methods could be used to provide VR museums with increasingly faithful virtual replicas, thereby enhancing visitors' learning experience [9].

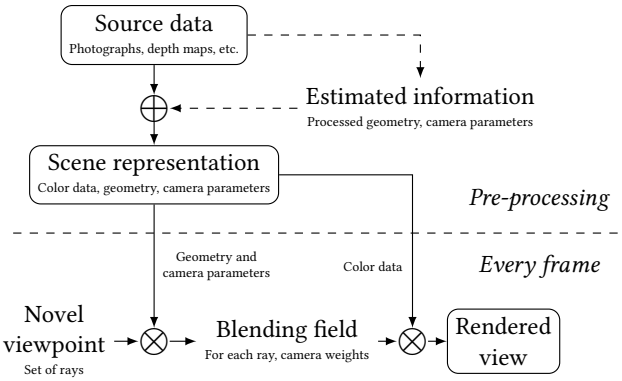

Figure 2: The standard shape of an image-based rendering method.

Unfortunately, the potential of image-based rendering seems to be largely untapped in everyday contexts. Based on discussions with several other research teams, we believe that this is mainly due to the lack of an easy-to-use interface encouraging content creators to use and learn more about these methods: no such interface currently seems to be accessible for public use[1]. By giving access to simple tools for rendering specular highlights and providing motion parallax from sets of photographs, we thus hope to encourage broader use of these methods for image-based VR content creation.

2.2 Design choices

2.2.1 Game engine implementation

We made the choice of implementing our toolset as a package for a popular game engine. It is important to acknowledge the limits of this decision: the usability of the toolkit is conditioned by that of the game engine itself, and there may be issues of efficiency (e.g. rendering speed, memory usage) that could have been avoided had the toolset been built from scratch as standalone software. Nonetheless, we believe that implementing our toolset within a game engine remains the best choice both to efficiently achieve our development goals and to reach our target audience. Indeed, game engines have a wide user base, and, in particular, are used by a great majority of VR content creators. Moreover, they easily enable creating responsive experiences with interactive 3D objects and provide built-in support for a wide array of color and geometry data formats. Finally, their flexible GUIs and shader programming pipelines make them a good fit for developing our user-oriented IBR interface. Note that our approach could be implemented equivalently within any commonly-used game engine. In our case, the main criteria for selecting Unity were its cross-platform support for many VR head-mounted displays and its wide adoption by casual content creators.

2.2.2 Extensions to build upon existing tools

By design, our interface naturally complements 3D reconstruction toolkits: indeed, it enables leveraging the dense geometry (e.g. 3D meshes and depth maps) estimated by these toolkits to generate accurate novel views from the original set of photographs. Therefore, in order to provide a seamless pipeline from input photographs to rendered views, we decided to make several functionalities of widely-used 3D reconstruction and mesh processing tools accessible from within our package's interface in Unity. Specifically, we provide a simple GUI to enable end-users to perform sparse and dense reconstruction from images using Schönberger et al.'s [20, 21]

[1] One discussion did point us towards two European Union projects, *CR-PLAY* and *EMOTIVE*, that aimed to integrate IBR in Unity, in the contexts respectively of game design and cultural heritage; however, to our knowledge, the resulting toolsets currently remains unavailable to the general public.

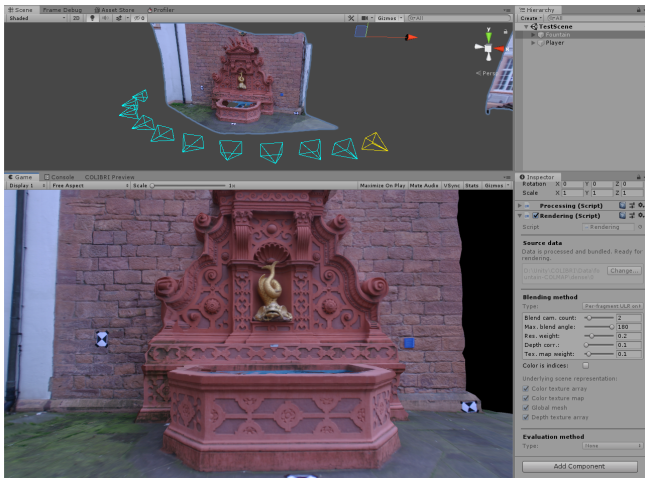

Figure 3: The toolkit's rendering interface, used to render source data interactively in VR. On the right, a GUI helps users select a rendering solution. On the left, the cameras are displayed in the scene, and the main window displays what is seen by the viewer. This scene is rendered using the 11 photographs of the fountain-P11 [24] dataset.

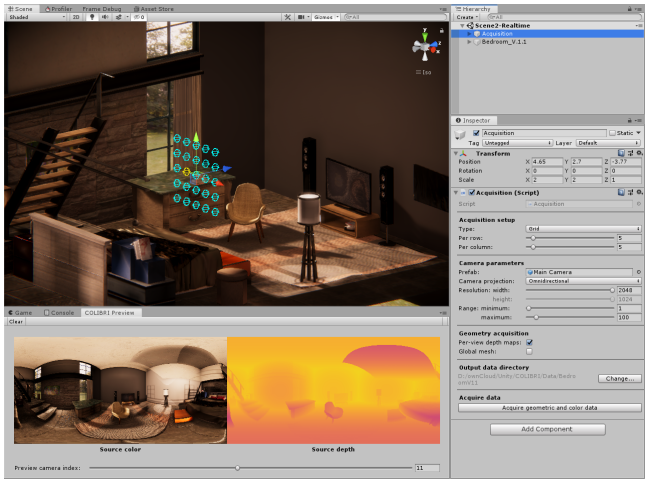

Figure 4: The toolkit's acquisition interface, used to capture data from virtual scenes. On the right, a GUI helps users set the parameters of the capture setup, here a set of 360° cameras arranged on a regular grid. On the left, the setup is represented, and a window displays the color image and depth map for the previewed camera.

COLMAP toolkit. Additionally, we include scripts that extend the toolkit's functionalities with the mesh processing and retopology methods provided by both Blender and Jakob et al.'s [13] Instant Meshes. This choice of external tools was motivated not only by the functionalities that they provide but also because all of them are free and provide downloadable executable files, ensuring that casual developers can access them with no added difficulty.

Note that these added functionalities are extensions of our toolkit, not dependencies: our rendering interface can be used to complement any preferred choice of reconstruction tools, so long as the generated geometry is in a supported format. This is also notably true for the camera models estimated for each source image, which ultimately have to be converted to match the toolkit's file structure, but can have been obtained by any preferred means.

2.2.3 Input data and camera model

Given the diversity of use cases and types of source data, we wanted the toolkit to be able to handle a variety of capture setups, including geometric and color information of different shapes and sizes. Building within Unity helps in this regard, as we thus ensure support for many common 3D mesh and image formats. On top of this, we define a custom camera model in the toolkit, to be able to adapt to input photographs of different resolutions, fields-of-view, and projection types (perspective but also omnidirectional, since many VR authors work with 360° images). This model intentionally does not include radial distortion parameters: because a sparse reconstruction step is required to estimate the cameras' positions and orientations from the input images, we expect this preliminary step to also generate undistorted images (as is done for example in COLMAP). Any real camera can still be used to capture the images, as long as the reconstruction tool can correctly estimate its parameters.

2.2.4 Rendering and acquisition tools

The main part of our interface is the rendering tool (see Fig. 3). There are two steps to this tool. First, one specifies the directory containing the source data (e.g. photographs and depth maps), and launches processing methods to transform this data into the desired scene representation (e.g. perform sparse reconstruction to estimate the respective positions and orientations of the cameras, and transform the depth maps into 3D meshes). This is an offline step, resulting in the scene representation being stored on disk as a set of assets. Stored assets include color data (typically stored as a texture array, interpreted by the GPU at render time as a single object), geometric proxies (stored as 3D meshes, instantiated at render time), and camera setup parameters (stored in text files). Second, one simply has to choose a blending method from the GUI and launch Unity's play mode: this will instantiate the stored representation into the scene, and render it in real-time using the specified blending method.

Another part of our interface is the acquisition tool (see Fig. 4), which we designed to help users capture images and geometry from synthetic 3D scenes. This component can be used to create test datasets on which to try out rendering methods before capturing the real source data. To acquire data from the virtual scene, one simply sets up the cameras in the scene and specifies their parameters in the GUI, before launching the capture process. Modifiable variables include the number of acquisition cameras, their parameters, and the setup's size along each axis, as well as the type of setup: structured grid, inside-out sphere, and outside-in sphere.

2.2.5 Scope of the project

We named our package *COLIBRI VR*[1], the Core Open Lab on Image-Based Rendering Innovation for Virtual Reality. The current version of the project only targets still photographs: video is out of its scope for now, and will be left for future work. Additionally, we currently only examine rendering solutions based on explicit geometry: methods based on optical flow [1] and depth-based image warping [3] have not yet been investigated. The contents of the project are open-source, and it can freely be accessed online. The code and documentation can currently be found at the address `https://caor-mines-paristech.github.io/colibri-vr`.

3 Details on our implementation

We now provide details on two of the rendering solutions that we implemented in COLIBRI VR. The first takes as input a global mesh, the second a set of per-view depth maps. Both aim to provide motion parallax and render captured specular highlights.

[1] Many languages use the word {*k-c*}*olibri* [koh-lee-bree] to mean *hummingbird*. We found this to be an appropriate acronym for our IBR-for-VR toolkit, as it aims to resemble hummingbirds in many ways: fast (i.e. must render in real-time), beautiful (i.e. must generate high-quality views) and compact (i.e. must efficiently handle data to avoid memory stress).

Figure 5: Views rendered by applying our implementation of unstructured lumigraph rendering on the 42 photographs of the Terrains [22] dataset and the 19 photographs of the Aquarium [3] dataset. Moving to different viewpoints reveals specular highlights.

Figure 6: Meshes created from the 360° color image and depth map of the Snow [6] dataset. Top: the added geometry provides motion parallax, but creates stretching artifacts when moving away from the initial viewpoint. Bottom: our toolkit can create depth-based meshes with smaller triangle counts (middle) than full pixel meshing (left), and remove stretched triangles at disocclusion boundaries (right).

3.1 View-dependent rendering of a global mesh

3.1.1 Our implementation

A good baseline for image-based rendering is the unstructured lumigraph rendering (ULR) algorithm [2]: it efficiently demonstrates the potential of view-dependent rendering algorithms and has consistently been used by recent works as a reference point for comparison [1, 10, 11, 15]. Therefore, we decided to implement our own version of ULR using vertex/fragment shaders. Note that the focus here is not on motion parallax, which is provided by default when the game engine renders the 3D mesh, but on convincingly rendering the captured specular highlights.

Our implementation (illustrated in Fig. 5) essentially applies the ULR vertex and fragment operations described by Buehler et al. [2]. In the vertex stage, we compute blending weights for each source camera, and store the weights and indices of the n most relevant cameras. Then, in the fragment stage, we compute the output color in each pixel by blending the colors of the n source images most relevant for this pixel. Relevance is here defined based on the considerations outlined in the original ULR publication: we give more weight to cameras closer to the given vertex (to select higher-resolution information) and in which the vertex is seen from an angle similar to that of the current viewpoint, while discarding those in which the vertex is occluded by another object. Because high framerates are commonly recommended to ensure comfortable VR viewing [12], we also modify the original algorithm to better maintain rendering at interactive speeds. Our method thus spreads the computational load over several frames by updating the blending weights only for a subset of the mesh's vertices every frame, using a sliding window to iterate over the entire set in a small number of frames, and rendering non-updated vertices with the stored weights from the last iteration. Therefore, it takes at most m frames to update the appearance of the entire mesh, where m can be specified by the user in the GUI (it should remain relatively small for the difference to hardly be noticeable). The actual number of vertices to process each frame is dynamically adjusted based on the current framerate.

3.1.2 Obstacles and limitations

We faced several obstacles when implementing this method. We already described how we overcame the issue of maintaining high framerates. Another issue was to find a way to efficiently interpolate the camera blending weights between the vertex and fragment stages, the built-in interpolators being too few in number to store all of them when there are large numbers of input images. We overcame this obstacle by adding an intermediate geometry stage, in which, for each of the mesh's triangles, only the n most relevant images are selected to provide color in the fragment stage: only a small subset of weights thus have to be passed in the interpolators. Finally, unstructured lumigraph rendering typically suffers from noticeable ghosting artifacts [15, 23], linked to inconsistencies between the image data and the reconstructed camera setup and 3D mesh. This remains a limitation in our current implementation.

3.2 Rendering per-view meshes from depth maps

3.2.1 Our implementation

Like global geometry, per-view geometry can be captured or estimated from sets of cameras with overlapping fields-of-view. This geometric information typically comes in the shape of depth maps, usually instantiated as 3D meshes during rendering [11, 17, 23]. The goal here is therefore to implement a processing method that generates 3D meshes from an input set of depth maps, in order to easily provide motion parallax [14, 23]. Additionally, for scenes created from multiple images, we want to render the generated set of per-view meshes in a way that displays specular highlights [17].

To transform input sets of depth maps into per-view meshes, we implemented a parallel GPU algorithm as a custom compute shader, inspired by the quadtree-based approach presented by Lee et al. [14]. The objective here is to create a compact mesh from an input depth map (see Fig. 6). To do so, the method examines each zone created by consecutive quadtree division of the depth map and evaluates the geometric error that would result from approximating this zone as a set of four triangles. We thus successively add to the mesh's triangles, adding larger triangles in zones for which this creates little error and smaller triangles where more detail is needed. Extending the original algorithm, we add the orthogonality and triangle size tests presented by Pajarola et al. [18] to give users the possibility to exclude triangles that would create visual artifacts (a stretching effect [6, 23], sometimes referred to as *rubber sheets* [18]) at disocclusion boundaries.

To render this representation, it is best not to use ULR, which was not designed for rendering per-view meshes and would be inefficient in doing so. Instead, we implemented a method inspired by the work of Overbeck et al. [17]: we render each per-view proxy one after the other, thereby consecutively adding to the rendered image's color. Each added color value is scaled by a blending weight, computed based on considerations similar to those used to compute the weights of unstructured lumigraph rendering: this blending weight is what will help render captured specular highlights during head movement. The total weight of an output pixel is stored in the texture's alpha channel, so that the output color values can be normalized after all per-view proxies have been drawn.

3.2.2 Obstacles and limitations

Correctly blending overlapping per-view meshes takes care. Depth testing is an issue, because such meshes are expected to closely overlap in 3D space. The standard z-test cannot be kept, as it may prematurely discard potentially relevant color values. Disabling the z-test would also be problematic: it may lead to inaccurate self-occlusions, because per-view meshes are potentially non-convex. To solve this, we implemented a soft z-test within the shader, using a ping-pong buffer system to read and write depth. We also scale the proxies' depth values [17] to ensure that those rendered later appear on top of those rendered first. We unscale these values when finally writing to the depth buffer, so as not to incorrectly write over objects that have been drawn before nor prevent future objects from being drawn correctly. We also chose to implement this method using command buffers, to have more control over the rendering process: indeed, command buffers enable us to know the proxies' rendering order for the scaling operation, and make it simple to normalize the output color as a final step after rendering every proxy.

4 Evaluation and Discussion

We conclude this paper by providing an initial evaluation of the toolkit. We also discuss paths for future work.

4.1 Evaluating rendering quality

The visual quality of an IBR method implementation can be objectively evaluated by the process of virtual rephotography [25]: value for a specified evaluation metric is obtained by comparing, for multiple image datasets, each source image in the set with the corresponding view rendered by the method when observing the scene from the source camera's point of view (the source image itself being excluded before rendering). We currently implement one such evaluation metric in the toolkit, that helps visualize the $C_b + C_r$ error [25] when comparing source image and rendered view. We also provide a list of useful test datasets in the toolkit's online documentation, some of which are referred to in this paper's illustrations [3, 6, 22, 24]. We have not yet used these methods to evaluate our toolkit, however, as there do not yet seem to exist [25] online rendering benchmarks with which to compare our implementation.

4.2 Usability by non-experts: rendering minerals in VR

To better assess the usability of our toolkit by casual users, we conducted a user study in the form of a short practice session.

4.2.1 Protocol

The study consisted in guiding each participant in an end-to-end practice session on the toolkit, from capturing photographs to viewing the scene in a head-mounted display. To do so, users were tasked first with capturing a specific object using a photo camera, then with following the toolkit's documentation to render this object in VR. They were told to ask the experimenter for guidance only if there was something they did not understand. After having successfully rendered their data, participants observed the rendered object in VR for several dozens of seconds using a HTC Vive head-mounted display. They then answered a post-test questionnaire consisting of open-ended and 7-point Likert-type questions related to ease of use, time spent processing the source data, and visual quality of the results. Participants also provided oral feedback by interacting with the experimenters after completion of the protocol.

As a practical backdrop for this study, we partnered with a mineralogy museum, interested in displaying mineral collections in VR. Participants were thus tasked with capturing and rendering a mineral of their choosing in the museum's collection (see Fig. 7), with an emphasis on recreating the mineral's visual properties (semi-transparence, highlights, etc.). As a guideline, users were told to take between 15 and 25 photographs of the selected mineral.

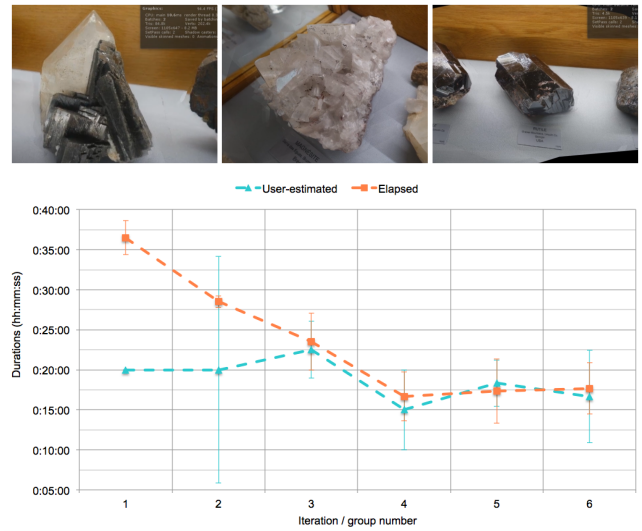

Figure 7: Top: rendered views of minerals captured by participants during the study. Participants took several photographs of a mineral of their choice, then followed the toolkit's documentation to render it and view it in VR. Bottom: we recorded the duration of the protocol (from capturing the first photograph to viewing the scene in VR), and asked participants for an estimation of how long it took them to complete it. Results are averaged per group, with later groups testing the protocol on improved iterations of the interface and documentation.

We designed the study to test incremental versions of the toolkit, in order to obtain diverse suggestions for improvement. Groups of 2 to 3 participants thus performed the protocol and provided feedback, based on which we improved the interface and documentation, thereby creating a new version for the next group to test.

4.2.2 Results and discussion

15 volunteers (3 female, 12 male) aged 24 to 61 ($M = 31.40$, $SD = 10.20$) took part in the study, testing 6 iterations of the project. Most were either museum staff or university-level students and researchers. None worked in fields directly related to IBR.

Overall, a majority of suggestions concerned how to improve the documentation, in its content, layout and illustrations, although it was generally found to already be quite clear (*Documentation*: $Mdn = 6$, $M = 5.73$, $SD = 0.96$). As for the interface, it was generally considered easy to use, although Unity's layout was perceived by several users as being initially quite intimidating (*Interface*: $Mdn = 6$, $M = 5.53$, $SD = 1.06$). Time-efficiency was consistently rated quite high and specified as one of the toolkit's strong points (*Time-efficiency*: $Mdn = 6$, $M = 6.07$, $SD = 0.88$). Rendering quality, on the other hand, was a more divisive matter: multiple users were very enthusiastic to see their photographs rendered as an interactive object in VR, but several others were disappointed by the presence of visual artifacts (*Rendering*: $Mdn = 6$, $M = 5.47$, $SD = 1.25$). The total time it took participants to process and render their data using COLIBRI VR drastically fell after the first series of improvements, before stabilizing at around 18 minutes, which we believe is a satisfying result for novice users (see Fig. 7). Users' estimations of elapsed time were also generally inferior or close to the ground truth, which is consistent with the ratings of time-efficiency.

4.3 Paths for future work

Several paths for future work remain. Volumetric video seems to remain a challenge even for many graphics researchers [17]: our next steps are therefore likely to focus instead on improving our current

implementation to better handle static scenes, e.g. to achieve higher levels of performance. A first step towards this goal could be to replace our current way of storing color data on the GPU (as a static texture array) with some form of fast dynamic texture streaming [17]. Secondly, to remove stretching artifacts without creating holes and jagged edges, one notable improvement could be to implement methods described recently for creating depth-based multi-layered mesh representations using background inpainting and soft handling of disocclusions [23]. To go beyond ULR, a future improvement path could also be to examine learning-based approaches, which seem to produce compelling results [8, 10]. Finally, we also hope to soon include a greater number of evaluation metrics [25], lead additional studies to further investigate the toolkit's usability and usefulness, and further demonstrate how the toolset can be used to create image-based content for a variety of everyday VR use cases.

5 Conclusion

In conclusion, we presented in this paper an open-source toolkit aimed at enabling casual content creators to render real-world scenes in virtual reality with motion parallax and specular highlights. We described the design choices that define the toolkit, detailed core components of our implementation, and presented preliminary evaluation results. Paths for future work include improving the toolset's technical functionalities, as well as further demonstrating and evaluating it in a variety of typical contexts.

References

[1] T. Bertel, N. D. F. Campbell, and C. Richardt. MegaParallax: Casual 360° panoramas with motion parallax. *IEEE Transactions on Visualization and Computer Graphics*, 25(5):1828–1835, May 2019. doi: 10.1109/tvcg.2019.2898799

[2] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pp. 425–432. ACM, New York, NY, USA, Aug. 2001. doi: 10.1145/383259.383309

[3] G. Chaurasia, O. Sorkine, and G. Drettakis. Silhouette-aware warping for image-based rendering. *Computer Graphics Forum*, 30(4):1223–1232, July 2011. doi: 10.1111/j.1467-8659.2011.01981.x

[4] C.-F. Chen and E. S. Rosenberg. Dynamic omnidirectional texture synthesis for photorealistic virtual content creation. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 85–90, Oct. 2018. doi: 10.1109/ismar-adjunct.2018.00040

[5] J. Cho, A. Kothari, Z. Ding, Y. Won, S. Fawaz, and X. Cheng. Injustice: Interactive live action virtual reality experience. In *ACM SIGGRAPH 2016 VR Village*, SIGGRAPH '16, pp. 9:1–9:2. ACM, New York, NY, USA, 2016. doi: 10.1145/2929490.2929493

[6] G. D. de Dinechin and A. Paljic. Cinematic virtual reality with motion parallax from a single monoscopic omnidirectional image. In *2018 3rd Digital Heritage International Congress (DigitalHERITAGE) held jointly with 2018 24th International Conference on Virtual Systems & Multimedia (VSMM 2018)*, pp. 1–8, Oct. 2018. doi: 10.1109/digitalheritage.2018.8810116

[7] N. de la Peña, P. Weil, J. Llobera, E. Giannopoulos, A. Pomés, B. Spanlang, D. Friedman, M. V. Sanchez-Vives, and M. Slater. Immersive journalism: Immersive virtual reality for the first-person experience of news. *Presence: Teleoperators and Virtual Environments*, 19(4):291–301, Aug. 2010. doi: 10.1162/pres_a_00005

[8] J. Flynn, M. Broxton, P. Debevec, M. DuVall, G. Fyffe, R. Overbeck, N. Snavely, and R. Tucker. DeepView: View synthesis with learned gradient descent. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2362–2371, June 2019. doi: 10.1109/cvpr.2019.00247

[9] D. A. Guttentag. Virtual reality: Applications and implications for tourism. *Tourism Management*, 31(5):637–651, Oct. 2010. doi: 10.1016/j.tourman.2009.07.003

[10] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (TOG)*, 37(6):257:1–257:15, Dec. 2018. doi: 10.1145/3272127.3275084

[11] P. Hedman, T. Ritschel, G. Drettakis, and G. Brostow. Scalable inside-out image-based rendering. *ACM Transactions on Graphics (TOG)*, 35(6):231:1–231:11, Nov. 2016. doi: 10.1145/2980179.2982420

[12] J. Huang, Z. Chen, D. Ceylan, and H. Jin. 6-DOF VR videos with a single 360-camera. In *2017 IEEE Virtual Reality (VR)*, pp. 37–44, Mar. 2017. doi: 10.1109/vr.2017.7892229

[13] W. Jakob, M. Tarini, D. Panozzo, and O. Sorkine-Hornung. Instant field-aligned meshes. *ACM Transactions on Graphics (TOG)*, 34(6):189:1–189:15, Oct. 2015. doi: 10.1145/2816795.2818078

[14] E.-S. Lee, J.-H. Lee, and B.-S. Shin. Bimodal vertex splitting: Acceleration of quadtree triangulation for terrain rendering. *IEICE Transactions on Information and Systems*, E97.D(6):1624–1633, June 2014. doi: 10.1587/transinf.e97.d.1624

[15] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar. Local light field fusion: practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):29:1–29:14, July 2019. doi: 10.1145/3306346.3322980

[16] E. M. Namara, S. O. Ciardhuáin, and D. M. Evoy. Visitor perceptions of the potential for virtual reality in a small museum. In *Proceedings of the 32nd International BCS Human Computer Interaction Conference*, HCI '18, pp. 1–5. BCS Learning & Development Ltd., Belfast, United Kingdom, July 2018. doi: 10.14236/ewic/hci2018.180

[17] R. S. Overbeck, D. Erickson, D. Evangelakos, M. Pharr, and P. Debevec. A system for acquiring, processing, and rendering panoramic light field stills for virtual reality. *ACM Transactions on Graphics (TOG)*, 37(6):197:1–197:15, Dec. 2018. doi: 10.1145/3272127.3275031

[18] R. Pajarola, M. Sainz, and Y. Meng. DMesh: Fast depth-image meshing and warping. *International Journal of Image and Graphics*, 04(04):653–681, Oct. 2004. doi: 10.1142/S0219467804001580

[19] M. Salmimaa, J. Kimmel, T. Jokela, P. Eskolin, T. Järvenpää, P. Piippo, K. Müller, and J. Satopää. Live delivery of neurosurgical operating theater experience in virtual reality. *Journal of the Society for Information Display*, 26(2):98–104, 2018. doi: 10.1002/jsid.636

[20] J. L. Schönberger and J.-M. Frahm. Structure-from-motion revisited. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4104–4113, June 2016. doi: 10.1109/cvpr.2016.445

[21] J. L. Schönberger, E. Zheng, J.-M. Frahm, and M. Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, pp. 501–518. Springer, Cham, Oct. 2016. doi: 10.1007/978-3-319-46487-9_31

[22] T. Schöps, J. L. Schönberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys, and A. Geiger. A multi-view stereo benchmark with high-resolution images and multi-camera videos. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2538–2547, July 2017. doi: 10.1109/cvpr.2017.272

[23] A. Serrano, I. Kim, Z. Chen, S. DiVerdi, D. Gutierrez, A. Hertzmann, and B. Masia. Motion parallax for 360° RGBD video. *IEEE Transactions on Visualization and Computer Graphics*, 25(5):1817–1827, May 2019. doi: 10.1109/tvcg.2019.2898757

[24] C. Strecha, W. von Hansen, L. Van Gool, P. Fua, and U. Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, June 2008. doi: 10.1109/cvpr.2008.4587706

[25] M. Waechter, M. Beljan, S. Fuhrmann, N. Moehrle, J. Kopf, and M. Goesele. Virtual rephotography: Novel view prediction error for 3D reconstruction. *ACM Transactions on Graphics (TOG)*, 36(1):8:1–8:11, Jan. 2017. doi: 10.1145/2999533

[26] J. O. Wallgrün, A. Masrur, J. Zhao, A. Taylor, E. Knapp, J. Chang, and A. Klippel. Low-cost VR applications to experience real world places anytime, anywhere, and with anyone. In *2019 IEEE 5th Workshop on Everyday Virtual Reality (WEVR)*, pp. 1–6, Mar. 2019. doi: 10.1109/wevr.2019.8809593