Presenting COLIBRI VR, an Open-Source Toolkit to Render Real-World Scenes in Virtual Reality

Grégoire Dupont de Dinechin, Alexis Paljic

Presenting COLIBRI VR, an Open-Source Toolkit to Render Real-World Scenes in Virtual Reality

Grégoire Dupont de Dinechin* Alexis Paljic†

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

ABSTRACT

From image-based virtual tours of apartments to digital museum exhibits, transforming photographs of real-world scenes into visually faithful virtual environments has many applications. In this paper, we present our development of a toolkit that places recent advances in the field of image-based rendering (IBR) into the hands of virtual reality (VR) researchers and content creators. We map out how these advances can improve the way we usually render virtual scenes from photographs. We then provide insight into the toolkit's design as a package for the Unity game engine and share details on core elements of our implementation.

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

1 MOTIVATIONS AND APPROACH

1.1 Use cases and current methods

Image-based VR experiences, in which the virtual environment essentially consists of assets created from real-world photographs, often serve two complementary purposes: education and entertainment. For instance, in the tourism sector, image-based virtual reality is typically used for the promotion and preservation of heritage sites, via the creation of visually accurate digital replicas of cultural objects [3]. Tools such as photogrammetry and 360° image capture also have industrial applications, e.g. to create virtual reproductions of industrial facilities for the training of operators, and are used in medical fields, e.g. to immerse viewers in digital replicas of specific locations in the context of therapy. Many everyday, general public applications, such as capturing personal memories to re-experience them and share them with family and friends, are also made possible by the development of new methods that enable 3D virtual environments to be created from photographs acquired casually using low-cost, consumer-grade cameras [4].

To create these experiences, VR developers commonly use two methods. The first is to render the scene using 360° photographs [9]: the image data is typically acquired using consumer omnidirectional cameras, and displayed on the inside of an inward-facing sphere surrounding the immersed viewer. The second is to render 3D-reconstructed meshes [8]: the image data can be acquired using any standard camera, and is then usually processed using structure-from-motion and multi-view stereo algorithms before being rendered as a texture-mapped 3D mesh. Both of these methods are quite practical, but also have weaknesses. 360° images alone notably fail to provide viewers with *motion parallax*, i.e. the depth cue that helps understand the distance of objects based on how they move in one's field-of-view during head motion. This may cause discomfort for viewers if they attempt to move around [2]. As for 3D meshes rendered only from a diffuse texture map, they fail to reproduce the *specular highlights* that are captured when taking photographs of reflective surfaces, and are therefore unable to faithfully render the visual appearance of many objects of our daily lives.

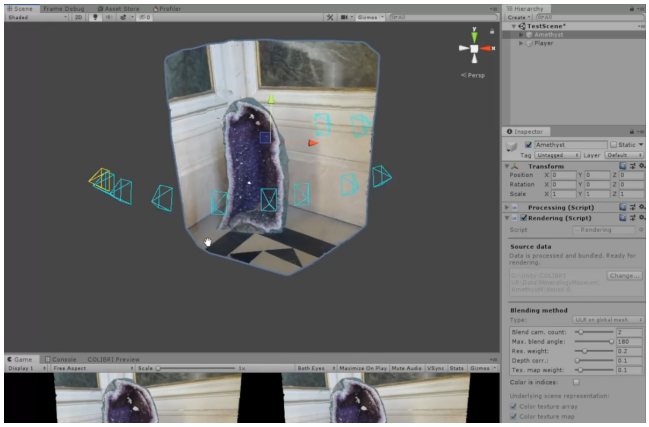

Figure 1: Rendering an amethyst in VR with motion parallax and specular highlights on the mineral's facets. This virtual object was processed and rendered from multiple photographs of the original mineral using our toolkit's graphical user interface, illustrated here.

1.2 Enhancing these experiences with IBR: designing an open-source toolset

Research in the field of image-based rendering has demonstrated potential for enhancing both the comfort and visual accuracy of these standard approaches to image-based VR. Specifically, many recent works have focused on creating *6-DoF VR* experiences from 360° photographs [2,4,9], demonstrating rendering solutions able to provide motion parallax from color images enhanced with depth information. There have also been advances in the field of view-dependent rendering, enabling the efficient rendering of captured specular highlights [1,6]. Unfortunately, it seems that no solution has yet been developed to make these methods easily accessible to the broader audience of VR researchers and content creators.

To tackle this issue, we therefore seek to provide a simple, openly available graphical user interface (GUI) by which developers can learn about and apply image-based rendering for VR. We name our open-source toolkit the Core Open Lab on Image-Based Rendering Innovation for Virtual Reality, COLIBRI VR[1]. We implement this toolset as a package for the Unity game engine, in order to reach a wide audience of VR developers and provide cross-platform support for a wide selection of popular VR head-mounted displays. The main functionality of our interface is its rendering tool, that enables processing the input data and rendering it using a selection of blending methods. We also provide a virtual acquisition tool, used to generate test datasets on which to try out different rendering

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

[1] https://caor-mines-paristech.github.io/colibri-vr

solutions. Finally, because most image-based rendering methods make use of underlying geometric proxies, our interface also includes extensions that provide easy access to 3D reconstruction and retopology tools from within Unity. Our GUI can thus notably be used to leverage the reconstruction capabilities of Schönberger et al.'s [8] open-source COLMAP toolkit, which is practical both for estimating the input cameras' relative positions and orientations and for recovering dense geometric information for the scene.

2 Implementation details

We now provide details on our implementation of several rendering solutions. We specifically consider the cases of two different types of input data: global geometry and per-view depth.

2.1 View-dependent rendering from a global proxy

A common way of storing the scene's geometry is as a single, global 3D mesh, obtained for instance by 3D reconstruction. To accurately render the scene's visual appearance from the captured photographs, we implement a modified version of a standard view-dependent rendering method, the Unstructured Lumigraph Rendering (ULR) algorithm presented by Buehler et al. [1]. This method updates the mesh's color values in real-time based on the user's current viewpoint, as a weighted combination of values from the n most relevant input images. To display view-dependent effects in each rendered 3D point, more weight is given to images that saw the point from an angle similar to that of the current viewpoint and with a high resolution, while images that saw the point from too different a perspective or in which this point is occluded are discarded.

We implement this method using the vertex and fragment shader operations described by the original authors, with several modifications. A notable challenge was implementing ULR efficiently enough to enable the high framerates commonly recommended for comfortable VR viewing [4]. To solve this issue, we provide options in the GUI to spread the computation of the camera blending weights in each vertex over several frames: we apply a rolling-window approach to iterate over all of the mesh's vertices, updating the weights for only a subset of vertices per frame, and rendering the others with the stored weights from the last iteration. This helps reduce the computational load of the method, with little cost in terms of visual quality as long as the process is spread over a small number of frames. A second challenge arose when we decided to move the computations from the fragment to the vertex stage [1], for similar reasons of improving framerates. Specifically, we found that the restricted number of built-in interpolators prevented us from easily interpolating the entire blending field between the two stages when considering large image datasets. To solve this, we added an intermediate geometry stage in which each of the mesh's triangles is matched with only a few images (those most relevant for this triangle): only the weights for these images are then carried to the fragment stage to create the output color.

2.2 View-dependent rendering from per-view depth

Another common approach is to render the input images from a set of corresponding depth maps, to provide motion parallax e.g. from 360° images [9] or large sets of perspective photographs [6]. To display this representation, we first generate 3D meshes from the input depth data, then render them using a view-dependent rendering solution able to display captured specular highlights.

To create per-view meshes from the input set of depth maps, we implement the quadtree-based parallel GPU algorithm introduced by Lee et al. [5] as a custom compute shader. This enables us to rapidly create a compact mesh that accurately fits the geometric data contained in the depth map, with larger triangles being added in zones for which this creates little error, and smaller triangles being added in zones where more details are required. We also include the orthogonality and triangle size tests described by Pajarola et al. [7], which enable automatically excluding triangles that would otherwise create a stretching effect at disocclusion boundaries [9].

To render the generated set of per-view meshes, we implement the disk-based blending solution described by Overbeck et al. [6] using vertex/fragment shaders. The output view is generated by combining the color values displayed by each per-view proxy: blending weights are computed based on considerations similar to those described for the ULR algorithm, and are similarly updated in real-time to adapt to the user's current viewpoint. To correctly implement this method in Unity, we had to rely on command buffers. This is because the method requires knowing the rendering order (when scaling the proxies' depth values), and because a final normalization operation is performed to divide the color in each pixel of the output view by the total weight stored in the alpha channel [6]. The main obstacle here was correctly handling depth testing: a z-test is required to prevent non-convex proxies from displaying inaccurate self-occlusions, yet it must not be implemented in a way that discards the information from meshes appearing only slightly behind what has already been drawn, because per-view meshes are expected to closely overlap in 3D space. We solved this by implementing a soft z-test, which enables us to blend the color values only from points at a small depth range around the final stored depth. We use a ping-pong buffer system to read and write depth during the process.

3 Conclusion

In conclusion, we briefly presented our development of a toolkit to help VR researchers and content creators more easily apply image-based rendering techniques. Paths for future work notably include extending our approach to video data, as well as implementing improved methods for preventing the emergence of noticeable visual artifacts.

References

[1] 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

[2] 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

[3] 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

[4] 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

[5] 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

[6] 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

[7] 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

[8] 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

[9] 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