Virtual Reality & Intelligent Hardware

• Article •

April 2025 Vol. 7 No. 2: 155–172
10.1016/j.vrih.2023.06.007

Multidimensional image morphing-fast image-based rendering of open 3D and VR environments

Simon SEIBT[1*], Bastian KUTH[2], Bartosz von Rymon LIPINSKI[1], Thomas CHANG[1], Marc Erich LATOSCHIK[3]

1. *Game Tech Laboratory, Faculty of Computer Science, Nuremberg Institute of Technology, Nuremberg* 90489*, Germany;*
2. *Department of Electrical Engineering and Computer Science, Coburg University of Applied Sciences and Arts, Coburg* 96450*, Germany;*
3. *Human-Computer Interaction Group, Institute of Computer Science, University of Wuerzburg, Wuerzburg* 97074*, Germany*

Received 19 May 2023; **Revised** 24 October 2023; **Accepted** 26 February 2024

Abstract: Background In recent years, the demand for interactive photorealistic three-dimensional (3D) environments has increased in various fields, including architecture, engineering, and entertainment. However, achieving a balance between the quality and efficiency of high-performance 3D applications and virtual reality (VR) remains challenging. **Methods** This study addresses this issue by revisiting and extending view interpolation for image-based rendering (IBR), which enables the exploration of spacious open environments in 3D and VR. Therefore, we introduce multimorphing, a novel rendering method based on the spatial data structure of 2D image patches, called the image graph. Using this approach, novel views can be rendered with up to six degrees of freedom using only a sparse set of views. The rendering process does not require 3D reconstruction of the geometry or per-pixel depth information, and all relevant data for the output are extracted from the local morphing cells of the image graph. The detection of parallax image regions during preprocessing reduces rendering artifacts by extrapolating image patches from adjacent cells in real-time. In addition, a GPU-based solution was presented to resolve exposure inconsistencies within a dataset, enabling seamless transitions of brightness when moving between areas with varying light intensities. **Results** Experiments on multiple real-world and synthetic scenes demonstrate that the presented method achieves high "VR-compatible" frame rates, even on mid-range and legacy hardware, respectively. While achieving adequate visual quality even for sparse datasets, it outperforms other IBR and current neural rendering approaches. **Conclusions** Using the correspondence-based decomposition of input images into morphing cells of 2D image patches, multidimensional image morphing provides high-performance novel view generation, supporting open 3D and VR environments. Nevertheless, the handling of morphing artifacts in the parallax image regions remains a topic for future research.

Keywords: Computer graphics; 3D real-time rendering; Computer vision; Image morphing; Virtual reality

Supported by the Bavarian Academic Forum (BayWISS), as a part of the joint academic partnership digitalization program.

Citation: Simon SEIBT, Bastian KUTH, Bartosz von Rymon LIPINSKI, Thomas CHANG, Marc Erich LATOSCHIK. Multidimensional image morphing-fast image-based rendering of open 3D and VR environments. Virtual Reality & Intelligent Hardware, 2025, 7(2): 155–172.

[*]**Corresponding author**, simon.seibt@th-nuernberg.de

Peer review under the responsibility of Beijing Zhongke Journal Publishing Co. Ltd.

1 Introduction

Over the last few years, there has been significant demand for photorealistic digital 3D environments, with applications ranging from virtual walkthroughs and digital twins to high-end gaming. In addition to memory efficiency, a major technical issue is balancing visual quality and performance, particularly to enable the feasibility of VR devices[1]. Motivated by this challenge, image-based modeling (IBM) and rendering (IBR) have been proposed as alternative approaches to conventional, i.e., geometry-based 3D computer graphics[2].

IBM is used to generate photorealistic representations by approximating a geometrical 3D model of an object or scene and then rendering it using a standard graphics pipeline[3]. However, this approach also poses significant challenges, such as hole-filling issues and visual artifacts, owing to the inaccurate reconstruction of the underlying 3D point cloud or surface mesh triangulation, respectively. Furthermore, the corresponding techniques often struggle to reconstruct surfaces that lack detectable features or have reflective and refractive properties.

In contrast, traditional IBR approaches focus on rendering by direct or at least implicit sampling of a given image set, typically including warping and compositing operations, to synthesize novel views[4]. Typically, the visual results depend strictly on the quality of the underlying detection of image feature correspondences between adjacent views, which may be impaired by parallaxes. In addition, these approaches typically produce intermediate virtual views with adequate visual quality within a limited spatial range.

Recently, neural-based IBR approaches, i.e., neural rendering and, specifically, neural radiance fields (NeRFs), have gained much popularity[5–9]. These methods often utilize a large set of input images to generate abstract scene representations using deep learning, which synthesizes novel views. Despite impressive visual results, they require an excessive amount of computational GPU power and memory capacity during preprocessing and rendering to achieve adequate performance and rendering quality, particularly for larger 3D scenes with free camera control.

In this paper, a new rendering method called multimorphing (multidimensional image morphing) is presented by revisiting and extending traditional image morphing approaches[10]. The proposed method utilizes a spatial 3D data structure, an image graph, to store image patches that are directly extracted from the original image set. It enables the rendering of photorealistic environments in 3D and VR with up to six degrees of freedom, using only a relatively sparse set of input images. By detecting and filtering parallax areas during preprocessing and extrapolating patches from adjacent image graph nodes, rendering artifacts are minimized. Furthermore, exposure inconsistencies were addressed to provide seamless brightness transitions between areas with varying illumination intensities. Multimorphing is designed to handle spacious, open environments that present performance and visual quality challenges, even for modern IBR solutions. The presented technique also handles complex and repeating visual details, such as tree leaves and stony grounds, which basic geometry-based approaches may struggle with. Its high performance makes it suitable for rendering photorealistic scenes even on legacy and mid-range GPUs, including portable VR devices.

The main contributions of this work are as follows. First, a preprocessing pipeline for generation of the image graph, a depth-free image-based rendering data structure for multidimensional image morphing, is presented. An interactive visualization with an adequate quality of open and static three-dimensional (3D) environments is the focus of this study. Second, a highly efficient IBR-rendering algorithm that supports interaction with six degrees of freedom (i.e., free camera movement and rotation) was proposed. Third, an on-the-fly extrapolation technique is presented for hole filling with missing image patches during rendering, covering feature mismatches or parallaxes detected during preprocessing. Therefore, pixel patches from

adjacent image graph cells were incorporated to fill empty image regions. Finally, a GPU implementation of real-time color mapping was proposed to support image datasets with varying light intensities. Thus, the eyes' ability to adjust to different lighting conditions was imitated.

2 Related work

Our work was partially inspired by traditional IBR approaches, as presented by Chen and Williams[11]. They described the interpolation between views using standard image morphing as an alternative to 3D rendering. Beier and Neely initially introduced standard image morphing, which is defined as a process of transforming, or warping, an image's pixel elements into another, typically including simultaneous blending of color values to approximate in-between views[10]. Further work by Chen comprised an extension of the aforementioned image-based concepts to be used in walkthroughs of virtual environments[12], which use arrangements of panoramic views in a grid data structure. Snapping the virtual camera at the closest grid points results in a stepwise virtual movement. The process of looking around is implemented simply by the rotational transformation of the panoramic views. However, the use of cylindrical panoramas and grid-based movements offers only a limited interactive experience. In this study, these basic concepts were extended to provide a highly efficient stereoscopic IBR method for 3D and VR with up to six degrees of freedom for camera movement.

McMillan and Bishop pioneered the connection between image-based rendering and plenoptic functions[13]. Gortler et al. and Levoy et al. utilized this approach for the development of plenoptic data structures, targeting image-based representations of 3D scenes[14,15]. Seitz and Dyer adapted morphing to handle 3D projective cameras and scene transformations between image pairs[16]. Debevec et al. presented a hybrid approach, including IBR and structure-from-motion, for the reconstruction of approximating meshes by combining it with projective texture mapping and hole filling[17]. Chaurasia et al. utilized superpixel warping based on depth approximation[4]. Lipski et al. introduced another hybrid approach that combines image morphing and depth-image-based rendering[18], utilizing both dense pixel correspondences between image pairs and a view-dependent geometrical model. Bertl et al. presented a technique based on novel-view synthesis with optical flow-based blending for the reconstruction of parallax-capable 360° panoramas within a limited spatial area, which can also be viewed in VR[19,20]. Ogniewski and Bonatto et al. utilized depth information acquired from RGB-D cameras and implicit depth reconstruction, respectively, to synthesize novel views for six-degrees-of-freedom HMD navigation and light-field displays[21,22]. Penner and Zhang proposed a soft 3D model of scene geometry to improve view synthesis quality while using depth information[23].

In recent years, image-based approaches based on neural rendering have exhibited impressive developments. Hedman et al. presented a deep learning approach for blending; that is, a convolutional neural network was trained to learn blending weights. These were used to combine the input image contributions and synthesize new images[24]. Zhou et al., Mildenhall et al., and Flynn et al. used neural networks to dissect input images into multiple planes[25–27]. Their solutions enabled more accurate transformation of image elements for view synthesis. Furthermore, Mildenhall et al. presented "neural radiance fields" (NeRFs), using deep neural networks to represent radiance fields for IBR[28]. Various NeRF extensions have been developed, and Liu et al. utilized a voxel data structure to accelerate neural rendering[5]. Rückert et al. used a hybrid approach, in which they reconstructed a point cloud from a given image dataset and augmented it with learned feature vectors[6]. For rendering, they rasterized the point cloud using one-pixel splats and used a deep neural network to fill the remaining gaps between the point splats. Their approach also involved a physically based photometric camera model for handling input images with

varying exposure and white balance. Attal et al. proposed optimization techniques to improve the performance and quality of NeRF rendering by predicting the sample points for each sight ray[1]. However, the small training sample sizes may have led to blurry results. Additionally, low frame rates (e.g., four FPS on an RTX 3090 GPU) limit the applicability of highly interactive 3D and VR applications. To address this low-performance issue, Garbin et al. extended NeRF rendering using caching, which, at the cost of higher memory demand, significantly improved the performance (e.g., to more than 200 FPS on an RTX 3090 GPU and an output resolution of 800×800)[7]. Müller et al. significantly reduced the training cost by using a smaller network and introducing versatile new input encoding to preserve quality[9]. For handling larger NeRF scenes, Tancik et al. proposed a scalable neural rendering approach[8]: spacious scenes were illustrated using multiple NeRF blocks, which can be trained individually. Tancik's et al. other modular software solution made it possible to balance between speed and quality by combining multiple NeRF approaches[29]. Nevertheless, even with modern neural rendering techniques, high performance and visual quality, along with memory efficiency and independence from costly high-end GPUs, remain challenges for IBR, particularly when visualizing larger photorealistic 3D and VR environments.

3 Image graph

This section presents a preprocessing pipeline for generating the image graph rendering data structure, which constructs a spatial data structure for IBR by partitioning the input images into morphing cells of four adjacent views (or three views for 2D scenes) with similar rotations. Each cell contains 2D image patches that are used for intracellular image warping and blending with respect to the current virtual camera location. Using dense image patches only from directly adjacent cells facilitates hole filling, fast processing, and less blurry image morphing. A high-level illustration of the pipeline is shown in Figure 1. The corresponding techniques for acquiring IBR input image sets are described in the previous subsection.

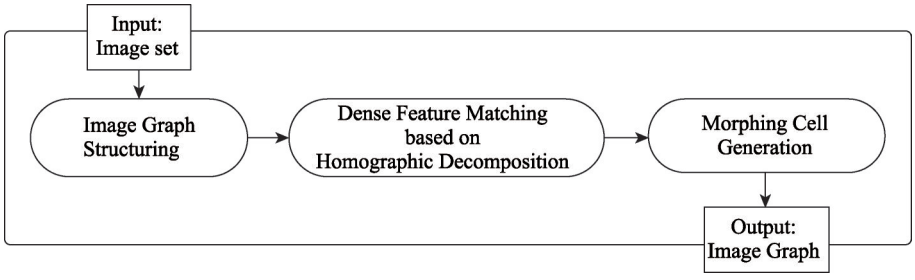


Figure 1 UML-based activity diagram of the preprocessing pipeline.

3.1 Basic structure

In the first stage of the preprocessing pipeline, the image graph data structure is constructed as follows: Initially, structure-from-motion is performed to estimate the camera poses, which then represent the graph nodes[30]. Graph edges are generated by the Delaunay triangulation of nodes with similar pitch and yaw rotations, thus creating simplexes that represent morphing cells for rendering, as shown in Figure 2. Depending on the desired degrees of freedom for translation, the simplexes are triangles for 2D or 3D tetrahedrons. For a 1D translation, no triangulation is necessary. In this case, only edges are generated between the nearest adjacent nodes, representing "rails" for composite camera movement paths.

3.2 Dense feature matching

In the next stage, feature matching is executed for each image pair defined by the edges in the image graph.

Because the resulting features are used to create triangular rendering patches for warping and blending, dense and precise feature correspondences are crucial for reducing visual distortions and ghosting artifacts in intermediate views (see Subsection 4.1). Therefore, the dense feature matching (DFM) method by Seibt et al. is used[31]. This feature-matching method targets images of real scenes with high depth complexity. The DFM extends "conventional feature matching" to an iterative rematching process by repeatedly searching for new feature matches, each with an individually estimated homography. The resulting homographic decomposition of the image space provides several advantages regarding feature-based IBR in terms of visual quality: (a) Matching features are positionally refined by taking advantage of multiple local homography candidates; (b) Through combinatorial analysis of the homographic decomposition, critical image areas, such occlusions and parallaxes, are detected and handled during rendering; and (c) Local feature extrapolation is used to identify additional feature points, especially in peripheral zones of critical image areas. Figure 3(c) and (d) show examples of Delaunay triangulations of a dense feature-matching set resulting from the DFM.

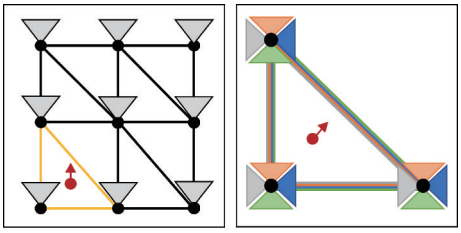


Figure 2 (a) 2D example of an image graph with nine graph nodes, each with the same viewing direction. Nodes were triangulated into eight morphing cells. The red dot represents the virtual camera; the current cell is colored orange. (b) Another example with three nodes and four camera rotations each. Camera poses with similar yaw and pitch angles were triangulated, resulting in four overlaying morphing cells.

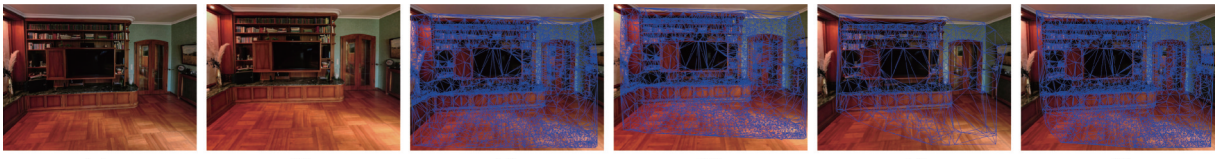


Figure 3 (a), (b) Input image pair; (c) Triangulation of DFM source image feature points; (d) Corresponding DFM mesh, mapped to matches in target image; (e) Initial feature triangulation after intra-cell matching and consolidation within a morphing cell; (f) Densified morphing cell triangulation due to intra-cell matching extrapolation.

3.3 Morphing cell construction

The last stage involves the construction of the rendered image patch structure of each morphing cell by Delaunay triangulation of the DFM's matching feature point sets per image graph node (i.e., per associated image). To ensure image morphing consistency across all nodes per morphing cell, we performed intra-cell matching consolidation; that is, the computation of one consistent intersection set from all included matched features (which implies discarding inconsistent feature points per node). However, this significantly reduced the matching density, as shown in the example in Figure 3(e).

To solve the aforementioned issue, the results from the DFM homographic decomposition are utilized, and the fact that each matched feature point is associated with a locally estimated homography is exploited. This information is also integrated into the morphing cell structure during preprocessing (i.e., each Delaunay mesh feature point vertex gets a different associated homography), depending on the respective image graph edge, or morphing target connection. To densify the consolidated feature set, the following intra-cell matching extrapolation algorithm is executed for each morphing cell: (1) Choose an arbitrary node of the morphing cell as the root node n_r and select an incident edge e of the cell. (2) Choose the remaining DFM feature match (p, p') corresponding to e that is not yet covered by the aforementioned intersection set. Then, select the enclosing Delaunay triangle t, which contains feature point p. (3) Transform p to the images corresponding to the other edges of n_r, using the local homographies at the vertices of t. (4) If a matching

distance between the transformed point and p is smaller than the minimal matching distance of the corresponding local vertex matches of t (i.e., those based on the homographies associated with the vertices that span t), then these "extrapolated" feature matches are considered in the morphing cell structure, including the aforementioned intersection set. (5) Steps 2 to 4 are repeated by successively selecting another edge e incident on n_r until all associated edges have been processed once.

Finally, the initial Delaunay feature point triangulations were repeated to account for additional extrapolated feature points. An example of the triangulation results after intra-cell matching extrapolation is shown in Figure 3(f). Furthermore, all Delaunay triangles classified as inhomogeneous by the DFM were discarded. As described by Seibt et al., the inhomogeneity of a triangle refers to an inconsistent configuration of homographic associations with respect to the corresponding vertices[30]. Typically, this refers to critical image areas, i.e., those containing visual occlusions and parallaxes, respectively. The resulting morphing cell structure, including the updated feature point meshes, now constitutes the final image patches and the associated morphing information required for rendering.

3.4 Image acquisition

The proposed preprocessing pipeline supports two types of RGB image datasets: unordered 2D photography and panoramic shots. To achieve adequate visual results, the input images should be captured with sufficient visual overlap (horizontally and vertically) because this directly affects the estimation of the feature correspondences described in subsection 3.2. For 2D photography, multiple shots at a fixed camera position with varying yaw and pitch are required to achieve satisfactory rotational coverage. The number of photographs required depends on the focal length of the camera and the desired field of view for rendering.

The proposed graphics pipeline also supports virtual 2D photography shots, that is, synthetic image datasets generated in 3D modeling software (such as Blender) using an integrated rendering engine. Multimorphing enables the visualization of geometrically complex 3D environments pre-rendered in high quality (e.g., offline path tracing using physically based shading) at high frame rates in 3D and VR, even on weaker graphics hardware. In the case of panoramic image datasets (360° equirectangular projections), all operations related to image feature matching were performed on multiple reprojected pinhole images. After the feature correspondence estimation, the results were projected back onto the panoramic sphere. Spherical Delaunay triangulation was used to generate the final image patches for rendering, as proposed by Caroli et al. [32]. In 2D photography, the resulting image graph consists of separate partially overlapping morphing cells, each in a distinct rotational image direction. In the panorama case, the morphing cell nodes are directly associated with the panoramic images, simultaneously covering all rotation dimensions.

4 GPU rendering

First, the rendering stages of a single morphing cell are outlined. The rendering process for multiple cells is then described, followed by real-time color mapping and VR rendering. An overview of the rendering pipeline is shown in Figure 4.

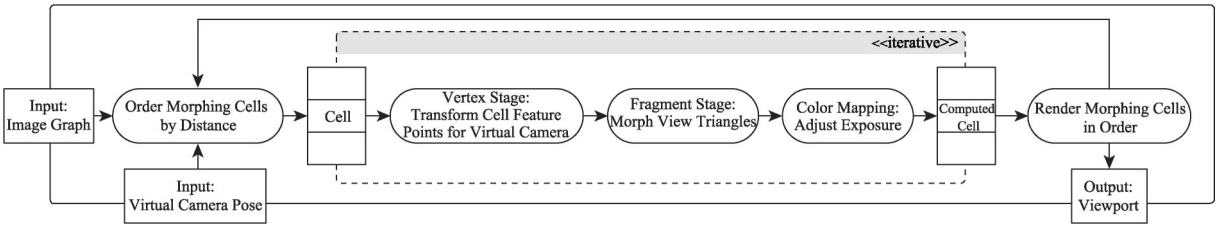


Figure 4 UML-based activity diagram of the presented multimorphing rendering pipeline.

4.1 Single cell morphing

To render a single image graph-morphing cell, its barycentric coordinates, relative to the current 3D position of the virtual camera, were first calculated. For simplexes with dimensions lower than the camera's three degrees of translation freedom, the projected barycentric coordinates that correspond to the position closest to the virtual camera position are determined.

4.1.1 Vertex stage

In the GPU's vertex stage, the morphing cell's feature points are processed as follows: the input attributes for each vertex comprise the 2D positions of the feature points in each cell's image. First, each 2D position was extended to 3D using the image focal length as a pseudo-depth component. After normalization, the vertex shader combines these pseudo-positions into a single vector using a rotational camera-to-world transformation and barycentric weights, each associated with the considered cell image. This vector represents the direction of the source light at the corresponding feature points. It is assumed that the input images adhere to a pinhole model and, hence, do not contain lens distortions. The resulting pseudo-position was then transformed into a world space before being transformed into a virtual camera space. This position can then be used to obtain the vertex position in the clip space of the graphics API. The vertex stage can be summarized using the following equation:

$$E \cdot V \cdot \sum_{i=0}^{d-1} \left(\lambda_i \cdot C_i \cdot \frac{\vec{p}}{\| \vec{p} \|_2} \right). \tag{1}$$

Let $x_i \in \mathbb{R}$ and $y_i \in \mathbb{R}$ be the coordinates of the feature point in the cell image i. w and h represent the image width, and height, respectively. The position (x_i, y_i) is in camera coordinates with: $-\frac{w}{2} \leqslant x_i \leqslant \frac{w}{2}$ and $-\frac{w}{2} \leqslant y_i \leqslant \frac{w}{2}$. Let $f \in \mathbb{R}$ be the focal length of the cell image i in pixels. After normalization, it acts as a pseudo-depth for a feature point and is negated because, by convention, the camera is looking in the negative z-direction. $\vec{p} = \left[x_i y_i - f_i \right]^T$ is calculated from the feature point position in cell image i and its focal length. After normalization, this vector represents the pseudo-pseudo-position of a feature. C_i is the rotational transformation from a vector in the camera coordinates of the cell image i to the world coordinates deduced from the reconstructed camera pose. $\lambda_i \in \mathbb{R}$ is the barycentric coordinate of cell image i. $d \in \{2, 3, 4\}$ is a dataset-dependent morphing cell dimensionality, i.e., the cell's spanning vertex count, which also determines the possible degree of translation. V is a virtual camera matrix that converts a vector from world coordinates to virtual camera coordinates. E is a matrix that transforms the camera coordinates into the clip space of the graphics API. If the pseudo-depths fit into the bounds set by the near and far planes of the clip space, then the same projection matrix can be used as in conventional 3D rendering. For input images that are not based on the pinhole camera model, the computation of the pseudo-position must be adjusted accordingly. For images based on equirectangular projections, $\vec{p}$ is determined by

$$\vec{p} = \left[\cos(\chi_i) \sin(\psi_i) \sin(\chi_i) \sin(\psi_i) - \cos(\psi_i) \right]^T, \tag{2}$$

where $0 \leqslant \chi_i < 2\pi$ is the feature point longitude angle in cell image i (thus the horizontal coordinate in an equirectangular image), and $0 \leqslant \psi_i < \pi$ is the latitude (thus the vertical image coordinate).

In addition, the 2D feature point positions are passed to the rasterizer as texture coordinates. For panoramas, a triangle t_0 can have vertices on opposite sides of the image (e.g., vertex a has a longitude of $\chi_a = 0 + \varepsilon$, and vertex b has $\chi_b = 2\pi - \varepsilon$). In such cases, the rasterizer would incorrectly interpolate the spherical coordinates for a fragment located in t_0 in the wrong spherical direction. Therefore, for equirectangular panoramas, the texture coordinates are passed to the fragmentation stage in a 3D Cartesian

representation. These were then renormalized, and the actual texture coordinates were recalculated (visualization shown in Figure 5).

(a) Equirectangular (b) Spherical (c) Cartesian

Figure 5 Panorama texture coordinate interpolation. (a) Input panorama image; (b) Incorrect texture sampling for spherical coordinates; (c) Correct interpolation using 3D cartesian coordinates.

4.1.2 Fragment stage

In the fragmentation stage, the colors of each cell image pixel were combined using texture sampling and blending operations. As introduced earlier, the use of "pseudo depths" based on the focal length allows the rasterizer to handle perspective-correct attribute interpolations, required for texture mapping. To prevent blending artifacts from being extrapolated to the border regions of the image graph, the barycentric coordinates of the camera are clamped to the closest position within the cell simplex during rendering. This is not the same as clamping each individual barycentric weight to $[0, 1]$: instead, these values must be clipped, depending on d – against the simplex's planes, edges, and corner points.

For virtual camera positions at the morphing cell centers, any preprocessing inaccuracies in the underlying morphing mesh structure may become most noticeable, leading to blurry rendering results. Therefore, a smoothstep function can be applied to "modulate" the clamped barycentric coordinates[33]. This function is used to bias the coordinates towards blending with the dominant contribution of only one cell image, resulting in potentially sharper results. Although smoothstep functions of a higher order can be constructed, we suggest using $f(x)=6x^5-15x^4+10x^3$, as it provides a reasonable balance between sharpness and smooth color transitions during camera movement within a morphing cell[34]. The smoothstep function is applied to the normalized barycentric coordinates as follows. First, it is performed on each individual barycentric weight. The weights were then renormalised. Now, let $\hat{\lambda}_i$ be the clamped and smoothstepped barycentric coordinate of cell image i. Moreover, let c_i be the corresponding source color. Subsequently, the final blended color was determined using

$$c_{\text{blend}}=\sum_{i=0}^{d-1}\left(\hat{\lambda}_i\cdot c_i\right). \tag{3}$$

4.2 Multiple cell morphing

Our image-based renderer attempts to fill in the viewport as much as possible by considering multiple overlaying and adjacent morphing cells. First, for each morphing cell in the image graph, the normalized barycentric coordinates for the current 3D camera pose are computed. Next, the cells are sorted based on two criteria: (a) Extrapolation distance, i. e., the translational distance that corresponds to extrapolation length required to maintain the camera position. If the virtual camera pose lies within a cell, then this distance is zero; (b) Rotational distance, which is the angle (yaw and pitch) between the virtual camera orientation and the originally recorded orientation of the included cell images. The extrapolation distance was defined as the primary sorting criterion. The rotational distance was used as a secondary criterion; that is, it was applied only to morphing cells with an equal extrapolation distance. This is the case for positionally overlapping cells with only varying rotations. For panorama image datasets, rotations are

handled directly because of the panoramic-image setup (see Subsection 3.4). Only the first sorting criterion is considered.

Finally, the rendering order for multiple morphing cells was determined using inverse cell sorting. Thus, the morphing cell that currently contains the virtual camera position (zero extrapolation distance) and is rotationally closest to its orientation was rendered last. This results in the overdrawing of distant morphing cells with potentially less accurate visual contributions. For optimization, the morphing cells outside the camera-view frustum were culled. Furthermore, the extrapolation can be controlled by setting a user-defined maximum allowable distance. Figure 6 demonstrates that multiple cell morphing can fill empty image regions that were filtered out during preprocessing, for example, owing to parallaxes (cf. inhomogeneous Delaunay triangles in Subsection 3.3). However, this approach worked satisfactorily for relatively small extrapolated distances. The alignment of distant cells may degenerate owing to numerical and geometric inaccuracies caused by extensive extrapolation. Figure 7 illustrates the overdrawing effect of the extrapolated morphing cells in the viewport.

(a) Single cell (b) Multiple cells

Figure 6 Multi-cell extrapolation. (a) Rendering of a single morphing cell, with visible "holes" primarily due to parallaxes; (b) Filling these gaps by taking advantage of extrapolation, which utilizes image patches from neighboring morphing cells.

(a) Multiple morphing cells (b) Wireframe visualization

Figure 7 Multi-cell overdrawing. (a) Six morphing cells, each consisting of three images, are morphed for one camera position (each cell is highlighted by exposure inconsistency); (b) Associated wireframe visualization with "heat color" per cell mesh (increasing extrapolation order from red to yellow).

4.3 Color mapping

As demonstrated in the previous section, rendering multiple morphing cells to fill the viewport can result in visible transitions between image patch meshes due to color or exposure inconsistencies. It is possible to equalize the colors of all input images "globally" during preprocessing. Nevertheless, real-time adjustment may be desired, particularly for scenes with highly dynamic lighting conditions, such as when the virtual camera moves in a dark room towards a bright window[35]. This effect is typically observed in modern video-game engines[36]. The main contribution presented in this section is a real-time GPU color mapping method specifically designed for fast IBR using multimorphing. This method is based on grey-level histogram matching by Gonzales et al.[37]. The method consists of the following three processing steps per morphing cell, where the primary morphing cell (i.e., the one containing the camera) is used as a color scheme reference for all subsequent rendered cells.

4.3.1 Histogram construction

The first step is to compute two RGB histograms of the overlapping region between two projected morphing cell meshes More complex color models, such as HSV or CMYK, are not considered, because additional conversions would result in lower performance. The reference histogram denotes the histogram containing the colors of the overlapping region that is currently in the framebuffer. The modification histogram refers to the colors of the overlapping region defined by the succeeding morphing cell. These histograms were computed concurrently during the fragmentation stage (see Section 4.1). To improve the performance, only 50% of an overlapping region's subset was sampled, defined by a blue noise texture.

4.3.2 Histogram mapping computations

Second, a color mapping lookup table $f(i)$ was computed for each color channel. $f(i)$ maps a single-color channel from the modification to the reference color space. Let $d \in \mathbb{N}$ be the color depth, and $G = \{ g \in \mathbb{N} \mid 0 \leqslant g < d \}$ be the set of valid indices or color intensity levels for mapping $f(i)$. Let r and m be the reference and modification histograms for a single channel, and $\hat{r}$ and $\hat{m}$ be the corresponding cumulative histograms[37]. For color mapping, each intensity level $i \in G$ must be mapped to a level $j \in G$, for which $\hat{m}_i = \hat{r}_j$. Because this equality is not always achievable owing to the different color distributions, it can be approximated per color channel using

$$f_{\text{naïve}}(i) = \min\left(\left\{ j \in G \mid \widehat{m}_i \leqslant \hat{r}_j \right\}\right). \tag{4}$$

Naïve mapping has limitations, in particular undesired color shifting if the range of colors observed in the overlapping region is smaller than the colors of the region to be mapped (for example, if the overlapping region represents only green grass and trees, then there is no color mapping information for a blue sky). This problem is addressed as follows. Let the observed range $\left[i_{\text{bot}},\ i_{\text{top}} \right]$ of the modification histogram be defined using $i_{\text{bot}} = \min\left(\left\{ g \in G \mid \widehat{m}_g \neq 0 \right\}\right)$ and $i_{top} = \min\left(\left\{ g \in G \mid \hat{m}_g = \max\ \widehat{m} \right\}\right)$. $f_{\text{naïve}}(i)$ searches for the minimum index j where $\hat{m}_i \leqslant \hat{r}_j$, which is given as follows:

$$\begin{aligned} \forall i \in \left\{ g \in G \mid g < i_{\text{bot}} \right\} &: f_{\text{naïve}}(i) = 0, \\ \forall i \in \left\{ g \in G \mid i_{\text{top}} \leqslant g \right\} &: f_{\text{naïve}}(i) = f_{\text{naïve}}\left(i_{\text{top}}\right). \end{aligned} \tag{5}$$

To address the flawed color mapping outside of $\left[i_{\text{bot}},\ i_{\text{top}} \right]$, strict mapping is introduced: For $f_{\text{strict}}(i)$, it is assumed that $f_{\text{naïve}}(i)$ only shifts each intensity value by a constant value. This is the case if the color difference is only a shift in the image brightness. In this scenario, the mapping function within the interval $\left[i_{\text{bot}},\ i_{\text{top}} \right]$ is a line. So, for $f_{\text{strict}}(i)$, we "strictly" extrapolate that line in the "unobserved range": Let $m = \dfrac{f_{\text{naïve}}(i_{\text{top}}) - f_{\text{naïve}}(i_{\text{bot}})}{i_{\text{top}} - i_{\text{bot}}}$ and $\text{clamp}(x,\ a,\ b) = \min\left(\max(x,\ a),\ b\right)$, then

$$f_{\text{strict}}(i) = \begin{cases} f_{\text{naïve}}(i), & \text{if } i_{\text{bot}} \leqslant i \leqslant i_{\text{top}}, \\ \text{clamp}\left(\left[mi - mi_{\text{bot}} + f_{\text{naïve}}(i_{\text{bot}}) \right],\ 0,\ d-1\right), & \text{otherwise.} \end{cases} \tag{6}$$

Despite correct extrapolation from the observed to an unknown color range, this approach still has a drawback: clamping causes a reduction in color detail, which can result in overly bright colors.

Finally, to preserve color gradients in the unobserved color range, preserving mapping is proposed: Therefore, a linear interpolation between the nearest known mapping and the endpoints' identity mapping is applied. Let $\text{mix}(a,\ b,\ t) = (1-t)\ a + t \cdot b$. Then, $f_{\text{preserving}}(i)$ is defined as

$$\begin{cases} \max\left(f_{\text{strict}}(i),\ \left[\text{mix}\left(0,\ f_{\text{naïve}}(i_{\text{bot}}),\ \dfrac{i}{i_{\text{bot}}} \right) \right] \right), & \text{if } 0 \leqslant i < i_{\text{bot}}, \\ f_{naïve}(i), & \text{if } i_{\text{bot}} \leqslant i \leqslant i_{\text{tot}}, \\ \min\left(f_{\text{strict}}(i),\ \left[\text{mix}\left(f_{\text{naïve}}(i_{\text{top}}),\ d-1,\ \dfrac{i - i_{\text{top}}}{d - 1 - i_{\text{top}}} \right) \right] \right), & \text{if } i_{\text{top}} < i < d. \end{cases} \tag{7}$$

The use of preserving mapping generally results in a more uniform distribution of color values. Some morphed cells may have exhibited minimal overlap. In such cases, based on a user-defined pixel area threshold, there is a fallback to identity mapping (i.e., $f(i) = i$). Color mapping was performed in parallel for each color channel. This GPU is implemented using a computer shader for fast processing.

4.3.3 Histogram mapping application

Pixels requiring color adjustment, masked by utilizing the alpha channel during histogram construction, are color mapped during rendering. If the morphing cell order changes (see Subsection 4.2), a smooth transition is applied between the previous and current color maps based on a user-defined effect duration. In areas with varying brightness levels, this effect simulates the human eye's adaptation to different light conditions. Figure 8 shows an example of the color mapping result.

(a) Mapping disabled (b) Mapping enabled

Figure 8 Visual comparison regarding real-time color mapping.

4.4 Stereoscopic and hybrid rendering

Stereoscopic IBR for VR applications is implemented using the multimorphing steps described in subsections 4.1 and 4.3. However, cell transitions in the virtual camera may result in visual discontinuities with respect to transformations and colors (e.g., inaccuracies during image acquisition and pre-processing). Therefore, the center position between the two virtual eyes was used to determine one morphing-cell order for rendering instead of using a separate order per eye. For the final multimorphing computations, the barycentric coordinates were calculated separately for each stereoscopic view. For optimization, the color mapping of the first rendered view was reused for the second view (Figure 9).

Figure 9 Stereo (VR) rendering using multimorphing.

Multimorphing for the background and polygon-based 3D rendering for foreground objects can be combined in a straightforward manner, given that the IBR scene is first rendered with disabled z-buffering. Next, the same camera and clip space transformations are used to transform the foreground polygon mesh vertices into the clip space of the frame buffer with enabled z-buffering. Figure 10 shows an example of a hybrid IBR.

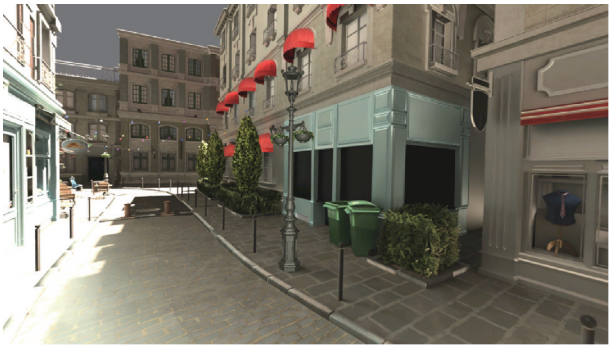


Figure 10 Hybrid IBR (with a polygonal streetlamp).

5 Experiments and discussion

Our software prototype was developed in C++ using the OpenGL software. Preprocessing was performed using OpenCV 4. All benchmarks were performed on a mid-range and legacy hardware setup: (a) an Intel i9-9900K CPU with 32 GB of RAM and an NVIDIA GeForce RTX 2080 Super GPU; and (b) an AMD Ryzen 7 3700X CPU with 32 GB of RAM and an NVIDIA GeForce GTX 1050 Ti GPU. The screen resolution was 1920×1080. For our VR headset (HTC Vive Cosmos), the combined resolution was 2880×1700 pixels at a refresh rate of 90 Hz.

Rendering results are evaluated for five image datasets: Farmland, Nature Walk (outdoor scenes),

Livingroom (indoor scene), Cave and Bistro (synthetic scenes, the first from the Unreal ICVFX project① and the second from NVIDIA ORCA②). For Nature Walk, Livingroom, and Cave, image graphs of different morphing cell dimensions were generated. All real-life scenes were self-recorded using a DJI Mavic Air 2 drone. Nature Walk and Livingroom were captured at eye level with a translation step-size of 1.0 to 1.5 m. The synthetic image datasets were generated using the physically based path tracer Cycles in Blender and the Scene Capture Cube actor in Unreal Engine 5, respectively. In both cases, we used a cubemap-based virtual camera setup for rotation with an extended overlapping field of view in the range of $100°-110°$. Table 1 provides detailed information regarding the image datasets used in the evaluation. The selected multimorphing rendering results for each scene are shown in Figure 11.

Table 1 Image datasets: Real photographs, reprojected pinhole images of 360° panoramas and synthetic scenes

Scene	Image Count	Image Source	Source Type	Degrees[a] $d-1$	Data Resolution $w \times h$
Farmland	63	real	photography	2	5472×3648
Nature Walk	35	real	photography	2	5472×3648
	378			3	5472×3648
Livingroom	20	real	panorama	2	4096×4096
	40			3	4096×4096
Cave	48	synthetic	panorama	1	4096×4096
	48			2	4096×4096
	48			3	4096×4096
Bistro	71	synthetic	panorama	1	2048×2048

[a] The dimension d of a morphing cell determines its shape and degrees of freedom with respect to translation: lines (1), (2), and (3).

Figure 11 IBR results for each test scene: Farmland, Nature Walk, Livingroom, Cave and Bistro.

5.1 Performance

To evaluate the multimorphing performance, the median rendering time of a virtual view was measured based on 500 samples for each dataset. For the 2D photo-based scenes, measurements were performed with and without color mapping. The number of morphing cells required to fill the viewport during IBR varies depending on the dataset's field of view. As shown in Table 2, all measurements significantly exceeded the minimum recommended FPS rate of 60 Hz for 3D real-time rendering. The results depend on the number of

① https://www.unrealengine.com/marketplace/production-test
② https://developer.nvidia.com/orca/amazon-lumberyard-bistro

input images and morphing cells used to fill the viewport. The presented preserving color mapping approach has a significant impact on the performance, multiplying the rendering times approximately by a factor of three. For stereoscopic (VR) rendering, multimorphing achieves the maximum frame rate of our VR headset (90 FPS) for every dataset, except for Nature Walk, with color mapping enabled on the legacy GPU (NVIDIA GeForce GTX 1050 Ti).

Table 2 Performance measurements: Times for rendering $|C|$ morphing cells with $|T|$ triangles and $|V|$ vertices

Scene	Degrees $d-1$	Color mapping	NVIDIA GeForce RTX 2080 Super			NVIDIA GeForce GTX 1050 Ti	#Cells $\lvert C\rvert$	#Triangles $\lvert T\rvert$	#Vertices $\lvert V\rvert$
			3D frame time (FPS)	VR frame time (FPS)	3D frame time (FPS)	VR frame time (FPS)			
Farmland	2	No	832 (1201)	11060 (90)	1300 (769)	11060 (90)	11	138138	70357
	2	Yes	2471 (404)	11060 (90)	4650 (215)	11060 (90)			
Nature Walk	2	No	1121 (892)	11060 (90)	2560 (391)	11060 (90)	17	503809	260738
	2	Yes	2673 (374)	11060 (90)	10550 (95)	11060 (90)			
	3	No	1838 (544)	11060 (90)	4460 (224)	11060 (90)	23	133218	67675
	3	Yes	3476 (288)	11060 (90)	13180 (76)	13900 (72)			
Livingroom	2	No	671 (1490)	11060 (90)	1050 (952)	11060 (90)	4	75491	39416
	3	No	752 (1329)	11060 (90)	1450 (689)	11060 (90)	4	9118	14025
Cave	1	No	709 (1410)	11060 (90)	1070 (934)	11060 (90)	5	81068	41511
	2	No	745 (1342)	11060 (90)	1100 (909)	11060 (90)	4	61499	31886
	3	No	1045 (957)	11060 (90)	1381 (724)	11060 (90)	10	223108	115526
Bistro	1	No	570 (1754)	11060 (90)	1090 (917)	11060 (90)	10	122038	61301

5.2 Memory

The total memory requirement of a morphing cell with $d-1$ degrees of freedom is determined by the following quantities: (a) **Triangles**, where each of the T triangles of a morphing mesh consists of three position indices, each represented by a 4-B integer. (b) **Positions**: Each of the P positions of a morphing mesh is represented by 2D floating-point coordinates. Moreover, there are d images per cell. (c) **Textures**: The textures were stored on a GPU using BC1 (formerly DXT1) texture compression[38]. In our case, this technique uses 4×4 RGB pixel blocks ($3\cdot8$ bits per pixel) and compresses each block to 64 bits. This results in a compression ratio of $\frac{(4\cdot4\cdot3\cdot8)}{64}=\frac{6}{1}$. In our GPU implementation, we used mip-mapping to reduce aliasing artifacts. Since each mipmap level is $\frac{1}{4}$ of the size of the previous level, all mipmap levels (including the first one) contribute to the memory consumption by an approximate factor of $\log_{n\to\infty}\sum_i^n(4^{-i})=\frac{4}{3}$. Assuming that all morphing cell images have the same width w and height h, the total memory size in bytes of a morphing cell is given by

$$\underbrace{d\cdot|P|\cdot2\cdot4B}_{positions}+\underbrace{|T|\cdot3\cdot4B}_{triangles}+\underbrace{d\cdot\frac{4}{3}\cdot\frac{w\cdot h\cdot3B}{6}}_{textures}. \tag{8}$$

The memory evaluation results shown in Table 3 confirm the plausibility of texture size being the most common significant factor for IBR memory consumption.

Table 3 Memory consumption of each image graph, separated by positions, triangles, and textures

Scene	Degrees $d-1$	Positions (%) $d \cdot \lvert P \rvert \cdot 2 \cdot 4B$	Triangles (%) $\lvert T \rvert \cdot 3 \cdot 4B$	Textures (%) $\lvert C \rvert \cdot d \cdot \frac{4}{3} \cdot \frac{2 \cdot h \cdot 3B}{6}$	Sum (100%) $\sum$
Farmland	2	1.69 MB (0.067%)	1.66 MB (0.066%)	2.52 GB (99.9%)	2.52 GB
Nature Walk	2	6.26 MB (0.444%)	6.05 MB (0.429%)	1.40 GB (99.1%)	1.41 GB
	3	2.17 MB (0.011%)	1.60 MB (0.008%)	20.1 GB (100%)	20.1 GB
Livingroom	2	946 KB (0.141%)	906 KB (0.135%)	671 MB (99.7%)	673 MB
	3	4.49 MB (0.025%)	109 KB (0.006%)	1.79 GB (100%)	1.80 GB
Cave	1	664 KB (0.025%)	973 KB (0.090%)	1.07 GB (99.8%)	1.08 GB
	2	765 KB (0.04%)	738 KB (0.046%)	1.61 GB (99.9%)	1.61 GB
	3	3.70 MB (0.172%)	2.68 MB (0.124%)	2.15 GB (99,7%)	2.15 GB
Bistro	1	981 KB (0.246%)	1.46 MB (0.367%)	397 MB (99.4%)	400 MB

5.3 Image quality

IBR quality measurements were performed using the structural similarity index measure (SSIM) and peak signal-to-noise ratio (PSNR). For this purpose, additional ground-truth images were generated for the synthetic scene Cave. For Nature Walk, real images were removed from the image graph and used as ground-truth references. Additional ground-truth images were captured. The results in Table 4 show that our image quality measures are comparable to those of other modern IBR approaches, such as Soft3D[23]. Quality losses may be primarily attributed to morphing artifacts in the parallax image areas.

Table 4 Image quality measurements (SSIM and PSNR)

Scene	Ground Truth Images	Average SSIM	Average PSNR
Nature Walk 2D	7	0.90	30.07
Nature Walk 3D	21	0.86	27.85
Livingroom 2D	4	0.91	31.77
Livingroom 3D	8	0.90	30.51
Cave 3D	8	0.93	33.01

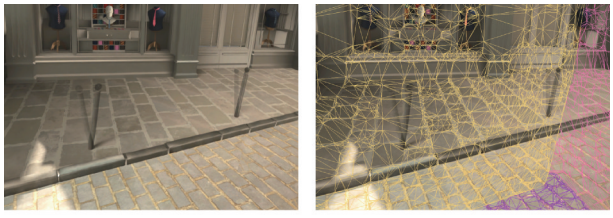
5.4 Comparison

For visual comparison with other methods, novel views were synthesized using the presented datasets. An evaluation was performed including the IBR methods Spixelwarp[4], Soft3D[23], neural rendering with Instant Neural Graphics Primitives (Instant NGP)[9], and Nerfstudio's default model Nerfact[29]. For each method, the same input images are used to render a representative novel view, assuming the best possible parameter settings for each scene. As shown in Table 5, the proposed method outperformed the other approaches in terms of image sharpness and viewport coverage for the relatively small and sparsely sampled image datasets used in this study. All IBR methods, including multimorphing, reveal warping artifacts such as ghosting and visual distortion, particularly in parallax image regions. In addition, Soft3D exhibited significant reconstruction errors, which resulted in visual gaps. SpixelWarp, Soft3D, constant NGP, and Nerfacto produced the most blurry results. This is particularly noticeable in neural rendering, especially in sparsely captured or visually distant image regions. Depending on the dataset size, the preprocessing time for Soft3D and multimorphing ranged between 1 and 2 h, whereas SpixelWarp required less than 3 min. For both machine-learning-based methods, the training for visual convergence took between 15 min and 1 h. All the rendering performance measurements were estimated after preprocessing and training at a resolution of 1920×1080 pixels using an RTX 2080 Super GPU (See FPS values in Table 5).

Table 5 Visual comparisons of novel view generation for scenes: Farmland, Nature Walk, Livingroom, and Cave

Spixelwarp[4] (27 – 61 FPS)	Soft3D[23] (--- FPS[a)])	Instant NGP[9] (0.3 – 5 FPS)	Nerfacto[29] (0.2 – 0.3 FPS)	Proposed (300 – 1400 FPS)
				

[a)] Novel views could not be captured during rendering, but could only be generated for predefined camera poses (https://sibr.gitlabpages.inria.fr/).



(a) Ghosting artifacts (b) Morphing cell meshes

Figure 12 Ghosting artifacts (at the street pillars) due to limited handling of parallaxes during preprocessing.

Figure 13 Morphing cell misalignment due to imprecise camera calibration and pose reconstruction, respectively. The right image is a close-up view of the left red rectangle.

5.5 Discussion

Our proposed IBR solution still has some limitations in terms of rendering quality. For instance, if preprocessing fails to filter out image patches that contribute to visual parallaxes, multimorphing may produce ghosting artifacts, as illustrated in Figure 12. The quality of our multicell rendering technique is also highly dependent on the underlying structure-from-motion precision, including the reconstruction of extrinsic and intrinsic camera parameters. Figure 13 illustrates how suboptimal camera calibration can affect rendering quality. Preprocessing inaccuracies may also affect the color quality because our color-mapping technique depends on correctly aligned morphing cell meshes. Finally, the missing image parts due to extensive or complex parallaxes (cf. Figure 6) can only be partially filled by the presented morphing cell extrapolation. This may result in image distortions or even visible gaps in the rendered image, particularly for low extrapolation distance settings.

6 Conclusion

In this study, a novel image-based rendering approach called multimorphing was presented. This approach

uses a set of calibrated and densely feature-matched images as the input. The underlying rendered data structure image graph supports the representation of photorealistic 3D and VR scenes from 2D photographs and panoramic-image datasets with various degrees of freedom. The processing of multiple overlaying morphing cells in the proposed rendering pipeline makes it possible to handle visual parallaxes using cell extrapolation. The proposed real-time GPU color-mapping technique reduced the visible color discontinuities during rendering. The proposed IBR solution achieves superior 3D and VR performances, even on legacy graphics processors.

Future work will include rendering quality improvements, particularly the preprocessing of images with significant occlusions and parallaxes. One possible approach is to segment the foreground and background image regions and then use the corresponding pixel patches for gap filling via real-time inpainting[39]. Furthermore, we plan to improve the proposed hybrid-rendering approach through additional depth reconstruction during preprocessing.

Declaration of competing interest

We declare that there are no competing interests.

CRediT authorship contributions statement

Simon Seibt: Conceptualization, Data curation, Funding acquisition, Investigation, Methodology, Software, Validation, Visualization, Writing-original draft, Writing-review & editing. **Bastian Kuth:** Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Visualization, Writing-original draft. **Bartosz von Rymon Lipinski:** Conceptualization, Funding acquisition, Methodology, Project administration, Supervision, Validation, Writing-review & editing. **Thomas Chang:** Data curation, Investigation, Writing-review & editing. **Marc Erich Latoschik:** Supervision, Validation, Writing-review & editing.

References

1 Attal B, Huang J B, Richardt C, Zollhoefer M, Kopf J, O'Toole M, Kim C. HyperReel: high-fidelity 6-DoF video with ray-conditioned sampling. 2023, arXiv: 2301.02238
https://arxiv.org/abs/2301.02238

2 Chan S C. Image-based rendering. Computer Vision, 2021. 656–664
DOI: 10.1007/978-3-030-63416-2_4

3 Agarwal S, Furukawa Y, Snavely N, Simon I, Curless B, Seitz S M, Szeliski R. Building rome in a day. Communications of the ACM, 2011, 54(10): 105–112
DOI: 10.1145/2001269.2001293

4 Chaurasia G, Duchene S, Sorkine-Hornung O, Drettakis G. Depth synthesis and local warps for plausible image-based navigation. ACM Transactions on Graphics, 2013, 32(3): 1–12
DOI: 10.1145/2487228.2487238

5 Liu L J, Gu J T, Lin Zaw K, Chua T S, Theobalt C. Neural Sparse Voxel Fields. 2020, arXiv: 2007.11571
https://arxiv.org/abs/2007.11571

6 Rückert D, Franke L, Stamminger M. ADOP: approximate differentiable one-pixel point rendering. ACM Transactions on Graphics, 2022, 41(4):1–14
DOI: 10.1145/3528223.3530122

7 Garbin S J, Kowalski M, Johnson M, Shotton J, Valentin J. FastNeRF: high-fidelity neural rendering at 200FPS. In: 2021 International Conference on Computer Vision (ICCV). Montreal, QC, Canada, 2021. 14326–14335
DOI: 10.48550/arXiv.2103.10380

8 Tancik M, Casser V, Yan X C, Pradhan S, Mildenhall B P, Srinivasan P, Barron J T, Kretzschmar H. Block-NeRF: scalable large scene neural view synthesis. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). New Orleans, LA, USA, 2022. 8238–8248
DOI: 10.1109/cvpr52688.2022.00807

9 Müller T, Evans A, Schied C, Keller A. Instant neural graphics primitives with a multiresolution hash encoding. ACM Transactions on Graphics, 2022, 41(4): 102

DOI: 10.1145/3528223.3530127

10 Beier T, Neely S. Feature-based image metamorphosis. In: Proceedings of the 19th annual conference on Computer graphics and interactive techniques. New York, NY, USA, 1992, 8: 35–42
DOI: 10.1145/133994.134003

11 Chen S E, Williams L, Chen S E, Williams L. View interpolation for image synthesis. In: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques. Anaheim, CA. ACM, 1993. 279–288
DOI: 10.1145/166117.166153

12 Chen S E. QuickTime VR. In: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. New York, NY, USA, 1995. 10–28
DOI: 10.1145/218380.218395

13 McMillan L, Bishop G. Plenoptic modeling. In: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. New York, NY, USA, 1995. 39–46
DOI: 10.1145/218380.218398

14 Gortler S J, Grzeszczuk R, Szeliski R, Cohen M F. The lumigraph. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. New York, NY, USA, 1996. 43–54
DOI: 10.1145/237170.237200

15 Levoy M, Hanrahan P. Light field rendering. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. New York, NY, USA, 1996. 31–42
DOI: 10.1145/237170.237199

16 Seitz S M, Dyer C R. View morphing. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. New York, NY, USA, 1996. 21–30
DOI: 10.1145/237170.237196

17 Debevec P, Yu Y Z, Borshukov G. Efficient view-dependent image-based rendering with projective texture-mapping. In: Rendering Techniques'98. Vienna: Springer Vienna, 1998. 105–116
DOI: 10.1007/978-3-7091-6453-2_10

18 Lipski C, Klose F, Magnor M. Correspondence and depth-image based rendering a hybrid approach for Free-Viewpoint video. IEEE Transactions on Circuits and Systems for Video Technology, 2014, 24(6): 942–951
DOI: 10.1109/TCSVT.2014.2302379

19 Bertel T, Richardt C. MegaParallax: 360° panoramas with motion parallax. In: ACM SIGGRAPH 2018 Posters, 2018. Article No. 38
DOI: 10.1145/3230744.3230793

20 Bertel T, Campbell N DF, Richardt C. MegaParallax: Casual 360° Panoramas with Motion Parallax. In: IEEE Transactions on Visualization and Computer Graphics. 2019, 25(5): 1828–1835
DOI: 10.1109/TVCG.2019.2898799

21 Ogniewski J. High-quality real-time depth-image-based-rendering. The Swedish Chapter of Eurographics, 2017. 1–8

22 Bonatto D, Fachada S, Rogge S, Munteanu A, Lafruit G. Real-time depth video-based rendering for 6-DoF HMD navigation and light field displays. IEEE Access, 2021, 9: 146868–146887
DOI: 10.1109/access.2021.3123529

23 Penner E, Zhang L. Soft 3D reconstruction for view synthesis. ACM Transactions on Graphics, 2017, 36(6): 1–11
DOI: 10.1145/3130800.3130855

24 Hedman P, Philip J, Price T, Frahm J M, Drettakis G, Brostow G. Deep blending for free-viewpoint image-based rendering. ACM Transactions on Graphics. 2018, 37(6): 1–15
DOI: 10.1145/3272127.32750

25 Zhou T, Tucker R, Flynn J, et al. Zhou T H, Tucker R, Flynn J, Fyffe G. Snavely N. Stereo magnification. ACM Transactions on Graphics. 2018, 37(4): 1–12
DOI: 10.1145/3197517.320132

26 Mildenhall B, Srinivasan P P, Ortiz-Cayon R, Kalantari N, Ramamoorthi R, Ng R, Kar A. Local light field fusion. ACM Transactions on Graphics. 2019, 38: 1–14
DOI: 10.48550/arXiv.1905.00889

27 Flynn J, Neulander I, Philbin J, Snavely N. Deep stereo: learning to predict new views from the world's imagery. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA, IEEE, 2016. 5515–5524
DOI: 10.1109/cvpr.2016.595

28 Mildenhall B, Srinivasan P P, Tancik M, Barron J T, Ramamoorthi R. NeRF: representing scenes as neural radiance fields for view synthesis. Communications of the ACM, 2021, 65(1): 99–106
DOI: 10.1145/3503250

29 Tancik M, Weber E, Ng E, Li R L, Yi B, Wang T, Kristoffersen A, Austin J, Salahi K, Ahuja A, Mcallister D, Kerr J, Kanazawa A. Nerfstudio: A Modular Framework for Neural Radiance Field Development. In: Proceedings of ACM SIGGRAPH 2023 Conference

Proceedings. New York, NY, USA, 2023. Article No. 72
DOI: 10.1145/3588432.3591516

30 Schönberger J L, Frahm J M. Structure-from-motion revisited. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA, IEEE, 2016. 4104–4113
DOI: 10.1109/cvpr.2016.445

31 Seibt S, Von Rymon Lipinski B, Latoschik M E. Dense feature matching based on homographic decomposition. IEEE Access, 2022, 10: 21236–21249
DOI: 10.1109/access.2022.3152539

32 Caroli M, de Castro P M M, Loriot S, Rouiller O, Teillaud M, Wormser C. Robust and efficient delaunay triangulations of points on or close to a sphere. In: Experimental Algorithms. Berlin, Heidelberg. 2010. 462–473
DOI: 10.1007/978-3-642-13193-6_39

33 Movania M M. OpenGL: build high performance graphics: assimilate the ideas shared in the course to utilize the power of OpenGL for performing a wide variety of tasks. Birmingham: Packt Publishing. 2017

34 Perlin K. Improving noise. In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques. New York, NY, USA, 2002. 681–682
DOI: 10.1145/566570.566636

35 Tian Q C, Cohen L. Histogram-Based Color Transfer for Image Stitching. Journal of Imaging, 2017, 3(3): 38
DOI: 10.3390/jimaging3030038

36 Roda C. Real Time Visual Effects for the Technical Artist. Boca Raton: CRC Press, 2022

37 Gonzales R C, Fittes B A. Gray-level transformations for interactive image enhancement. Mechanism and Machine Theory, 1977, 12(1): 111–122
DOI: 10.1016/0094-114x(77)90062-3

38 Banterle F, Artusi A, Debattista K, Chalmers A. Advanced high dynamic range imaging: theory and practice. New York: CRC Press, 2011

39 Herling J, Broll W. High-quality real-time video inpaintingwith PixMix. IEEE Transactions on Visualization and Computer Graphics, 2014, 20(6): 866–879
DOI: 10.1109/tvcg.2014.2298016