Panoramic Ray Tracing for Interactive Mixed Reality Rendering Based on 360° RGBD Video

Jian Wu and Lili Wang, *State Key Laboratory of Virtual Reality Technology and Systems, School of Computer Science and Engineering, Beihang University, Beijing, 100191, China*

This article presents an interactive panoramic ray tracing method for rendering real-time realistic lighting and shadow effects when virtual objects are inserted in 360° RGBD videos. First, we approximate the geometry of the real scene. We propose a sparse sampling ray generation method to speed up the tracing process by reducing the number of rays that need to be emitted in ray tracing. After that, an irradiance estimation channel is introduced to generate noisy Monte Carlo images. Finally, the final result is smoothed and synthesized by interpolation, temporal filtering, and differential rendering. We tested our method in a number of natural and synthesized scenes and compared our method with results from ground truth and image-based illumination methods. The results show that our method can generate visually realistic frames for dynamic virtual objects in 360° RGBD videos in real time, making the rendering results more natural and believable.

In mixed-reality (MR) applications, photorealistically inserting virtual objects into natural environments is a challenge.[1] One of the problems is that it is difficult to capture the lighting, geometric, and material information of the whole environment. The lack of information about the natural environment often leads to inconsistencies in the final image due to some artificial lighting and geometric assumptions. Another problem is that when modeling the interaction between virtual objects and natural environment lighting, some approximations are often performed to meet MR applications' high frame rate requirements. However, many details are lost, such as surface shadows, spatially varying lighting, etc.

In a typical head-mounted augmented reality device, the real environment information (including illumination and geometry) is captured within the narrow field of view of the front camera. Due to the lack of environmental information outside the field of view, many researchers have focused on simulating light and shadow interactions between virtual objects and captured parts of the natural environment, assuming out-of-view directional lighting conditions. As a result, the interaction between the virtual object and the geometry of the out-of-view environment is missing, leading to many unacceptable artifacts. Recently, some researchers have chosen to estimate the outer-view environment information in order to obtain the complete real-world environment lighting and geometric information. However, there are several problems with this. For one thing, the real world may be very different from the estimated outlook environment. Another problem is that some artifacts, such as color stretching and missing reflections, may occur when virtual objects interact with depth gaps in RGBD images.

Unlike normal video obtained from a typical head-mounted AR device, 360° video captures omnidirectional illumination of the environment around the center camera but without geometric information. In recent years, researchers have begun inserting virtual objects into 360° videos and trying to make these objects appear to blend in with the natural environment in the video. Assuming that the ambient illumination in 360° videos comes from infinity, the virtual objects can be consistently illuminated by image-based lighting (IBL). However, this assumption of infinity would make the virtual objects appear to float in the

0272-1716 © 2023 IEEE
Digital Object Identifier 10.1109/MCG.2023.3327383
Date of publication 25 October 2023; date of current version 25 January 2024.

real scene. Many interactions between virtual and real objects cannot be modeled without geometric information. This makes it difficult to create realistic merging effects because many important effects, such as spatial change effects, are missing. Many methods have been developed to solve the problem of missing geometry in panoramas. In recent years, many researchers have started to study the RGBD panorama in MR applications. A real-time 3-D panorama sensor cluster system was developed to support instant 3-D reconstruction and real-time updating.[2] A high-density, high-resolution 360° RGBD dataset was constructed at the same time.[3] After performing offline estimation or acquisition of panoramic depth data using the two methods mentioned above, we conducted postprocessing on the sparse raw data. This postprocessing involved data completion and filtering to meet the requirements of our method for dense depth input data. Using the depth and lighting information of the real-world environment, virtual objects can be realistically inserted into the real-world environment.

This article introduces an interactive panoramic ray tracing (IPRT) method to facilitate real-time realistic virtual object insertion and rendering for 360° RGBD videos in MR. We first approximate the geometry of a real scene with a panoramic and a screen space depth buffer. Then, we propose a sparse tracing acceleration method to speed up the tracing process by reducing the number of rays emitted during ray tracing. After that, an irradiance estimation pass is introduced to generate a noisy Monte Carlo image. Finally, the result is smoothed and synthesized by interpolation, spatial–temporal filtering, and differential rendering. Our method supports spatially varying global illumination for multiple virtual objects materials, including diffuse, glossy, and specular, allowing dynamic virtual objects and changing real-world geometry. We tested our method in natural and synthetic scenes and compared the result of our method with the ground truth and that of the IBL method. The results show that our method can generate visually photorealistic frames for inserting virtual objects into 360° RGBD videos in real time. Figure 1 shows the result of our method compared with that of the IBL method in the *Chamber* scene.[3] In contrast, our method exhibits better geometry consistency, for example, the reflections of leaves and the virtual gift box on the vase surface, the shadows on the table between two gift boxes, and the wall surface near the hanger branch. Our method runs at over 100 fps, which, as far as we know, goes beyond the state-of-the-art methods. In summary, the contributions of this article are as follows:

- An IPRT pipeline, with which dynamic virtual objects can be inserted and rendered photorealistically into 360° RGBD videos in real time.
- A backface aware screen space ray tracing (SSRT) algorithm considers the depth gaps and enables more reasonable and realistic virtual and real objects illumination results.
- A sparse tracing acceleration method consisting of a sparse tracing mask generation and interpolation processes to further accelerate the ray tracing process and maintain acceptable quality.

The code of our method is available at https://github.com/nuo-wuliao/PanoramaAR.

RELATED WORK

We refer the reader to well-organized review works[1] for a detailed discussion of MR rendering. Here, we provide a brief overview of some recent work on MR rendering and discuss some of the relevant techniques from our paper.

Rendering in MR

It is crucial to conduct realistic light interactions between real and virtual objects to acquire a photorealistic MR image.

Single RGBD Image-Based MR: Some researchers use the radiance transfer method to realistically insert virtual objects into a single RGBD image. Kan et al. introduced the differential irradiance caching method to simulate the light transport between virtual and real objects and simulate some global illumination effects.[4] Knecht et al. proposed a differential instant radiosity framework for reflective and refractive virtual objects inserting.[5] Gruber et al. presented an improved radiance transfer sampling approach, which supports dynamic changing scene geometry.[6,7] By capturing an environment map of the natural world and extracting the surrounding 2-D texture region, Alhakamy et al. inserted virtual objects into an AR application.[8] Besides, Mehta et al. proposed an axis-aligned filtering scheme to denoise a sparsely sampled image shaded by a physically correct Monte Carlo ray tracing.[9] These works considered only the lighting and geometries inside the single RGBD image. Environmental lighting is ignored or represented with a simple directional light source. Besides, the frame rates are not enough for most interactive MR applications.

Estimated Environment Illumination: Since the illumination and geometry information that a single RGBD image offers is insufficient, some researchers are trying to recover the lost illumination/geometry of

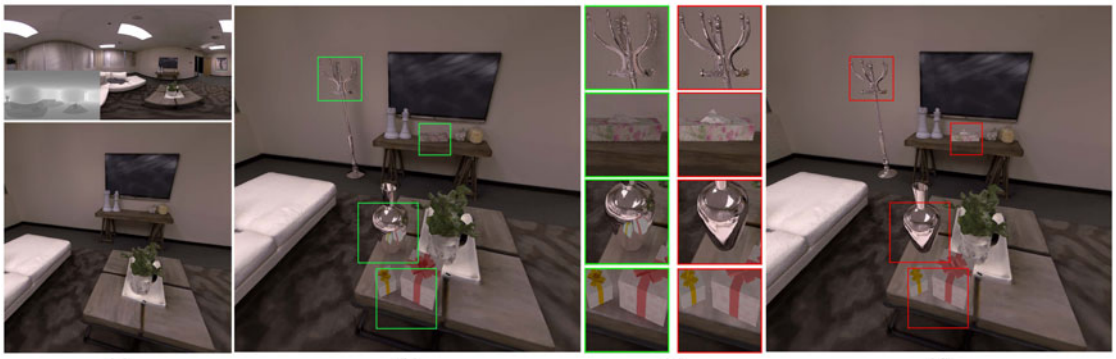

FIGURE 1. (a) Starting with a frame of a 360° RGBD video of the *Chamber* scene and a user view as input, we insert a specular virtual vase, two diffuse gift boxes, a diffuse tissue box, and a glossy coat hanger into the scene. (b) Our method calculates the environment illumination using less than 1 sample per pixel image space Monte Carlo (MC) path tracing followed by spatial–temporal filtering. (c) Zoom-in comp. (d) We compare our result with the image-based lighting (IBL) method, which ignores the geometry relations between virtual and real objects and shows incorrect appearance. Our approach runs at 103.4 fps on average.

the environment. One approach supposes that the indoor environment is a box with six faces and recovers the illumination distribution according to the single RGBD image. Then, virtual objects can be inserted into the recovered scene.[10] Park et al. estimated the HDR environment illumination using an RGB image of a reference object with a deep learning-based method, and the virtual object can be inserted into the environment at the reference position.[11] Another approach estimates a multiscale volumetric scene representation with a 3-D CNN network according to the known RGBD image, and the virtual object is inserted into the scene with a real-time spherical volume tracing.[12] With the rapid development of deep learning, many researchers have started to construct datasets to train for environment illumination estimation.[13,14] A deep inverse rendering framework was conducted to insert virtual objects into single RGB images by estimating the complete reconstruction of the scene.[15] However, the estimated environment illumination/geometry information does not always help improve MR rendering quality. Sometimes, it is not coherent with the existing geometry and introduces some artifacts to the result images. Recently, a GAN-based approach decomposes a single panorama of an empty indoor environment into appearance components to enable applications like virtual furniture insertion.[16] However, the method is designed for empty indoor rooms, and the lighting interactions only exist between virtual objects and the room itself.

Panorama-Based MR: Panoramic video, which offers an omnidirectional illumination of the environment, arouses the interest of researchers in MR. Rhee et al.[17] proposed MR360, which extracts a natural world light source and projects the virtual object shadow onto an assumed 3-D plane ground. Recently, Wang et al.[18] conducted a bidirectional shadow rendering method for 360° video, which can render the interactive shadows between virtual and real objects. Tarko et al.[19] used omnidirectional structure from the motion method to construct a spatially distributed local environment map and insert virtual objects with IBL and image-based shadowing for moving 360° videos. The added shadows and the specialized environment map can improve the realistic effect of the mixed scene. However, these methods may not work well when the real geometries are complex and close to the inserted virtual object. The spatially varying reflection effect plays an important role in this situation. The problem is that the lack of geometry information in panoramic video limits the image quality of the MR application. As we mentioned above, real-time updating 360° RGBD images is already accessible.[2,3] With these approaches, we aim to provide a photorealistic MR rendering application.

Related Technique

Screen Space Ray Tracing: In prior work, depth is used to re-render an image from a nearby viewpoint.[20] This idea is extended to SSRT and is widely used for local specular reflections.[21,22] While some researchers focus on how to improve the accuracy of SSRT,[23] others are trying to accelerate tracing speed by parallel computing on GPU[24] or hierarchical depth layer structure.[25]

We start from SSRT and extend it into panorama image space. Instead of running a general SSRT algorithm in a cube map to find intersections, we directly trace rays in the panoramic image space since the input panoramic data format.

Denoising Monte Carlo Images: Image denoising filters are efficient and straightforward approaches to removing noise in Monte Carlo images and reconstructing the final result. Many accurate reconstruction methods take seconds to minutes to get a smooth image.[26] Although the development of modern hardware reduced the time they cost, it is still hard for them to apply in real-time applications.[9,27] With temporal accumulation, many real-time denoising methods have emerged. The spatiotemporal variance-guided filtering method analyzes both temporal and spatial variance to generate a stable sequence of images from global illumination.[28] We use the joint bilateral filter[29] as a spatial filter, along with a temporal accumulation[30] to reconstruct the virtual-real mixed images.

INTERACTIVE PANORAMIC RAY TRACING

We aim to propose a real-time rendering method to insert dynamic virtual objects into the natural world environment represented by 360° RGBD videos while keeping the appearance of lighting, reflections, and shadows consistent. Depending on the depth of the environmental panorama, we can know precisely where the light emanates from. Then, assuming all materials in the natural environment are diffusely reflective, any position in real-world space can be considered as light by a panoramic pixel since diffusely reflective material objects will reflect light uniformly in any direction. By calculating the appearance of virtual objects based on the position of each pixel on their surface and the ambient light, we can simulate the result of a spatially varying fusion of reality and illusion.

Inspired by the ideas of fast spatial–temporal filtering algorithm[28] and the image space ray tracing approach,[24] we propose an IPRT method for mixed 360° RGBD videos. Our method renders frames in real time, where virtual objects naturally blend with real-world scenes with realistic lighting, shadows, and reflections.

Figure 2 outlines our RGBD panorama frame IPRT pipeline based on the differential rendering framework.[31] The pipeline consists of six passes. The first pass is the preprocessing pass. In this pass, we generate G-buffers for the merged scene that will be used in subsequent processes. We take as input the 360° RGBD video, user view information, virtual objects, and their locations. The outputs are two G-buffers in screen space and panorama space. Each buffer consists of color, normal, depth, material, etc. In addition, we generate mipmaps of the panoramic depth map for the irradiance estimation pass.

The second sparse sampling mask generation pass and the fourth interpolation pass are designed to control the sampling rate of the light tracked from the user's viewpoint to meet the high frame rate requirements of many MR applications. They are packaged as acceleration options, which means the system can often run without them.

The third pass is irradiance estimation. In this pass, we first calculate only the light reaching the real surface and update the color of the real surface. Then, we compute the light reaching the surface of the virtual object. We already know that the irradiance of a pixel E can be computed by Monte Carlo estimation with the following equation:

$$E = \frac{1}{N}\sum_{i=1}^{N} L_{\omega_i} \cdot f_r \cdot \cos\theta / p_{f_r}(\omega_i) \tag{1}$$

where L_{ω_i} is the radiance of the sampled ray from direction ω_i, f_r is the BRDF, $\cos\theta$ is the cosine term, and $p_{f_r}(\omega_i)$ is the probability density function (PDF) of the sampled ray distributed with BRDF. Usually, we need as many samples as possible to reach a better estimation. However, in order to run the system in real time, we only sample at most one Monte Carlo ray for each pixel in a frame ($N = 1$). In each frame, we get the noisy irradiance image. In this process, we compute the irradiance of each pixel using panoramic ray tracing (PRT) and back-face aware screen space ray tracing (BASSRT) algorithms.

After the above passes, we insert a filtering pass to reconstruct the noisy Monte Carlo irradiance images accurately. The reconstruction performs two main steps. In the first step, we use the screen space G-buffer to drive a 65×65 joint bilateral spatial filter[29] to obtain an initial irradiance estimate. This filter is implemented in a 5-level $\grave{a}$-trous manner for efficiency.[28] In the second step, we temporally filter the resulting image from the spatial filtering process using motion vectors. In practice, we set the temporal fading rate to 0.1, which means that the current frame contributes only 10% to the filtering result. We also perform a clamping operation to minimize ghosting during temporal accumulation.[32]

In the final differential rendering pass, we calculate the final result of the merging scene. As illustrated in

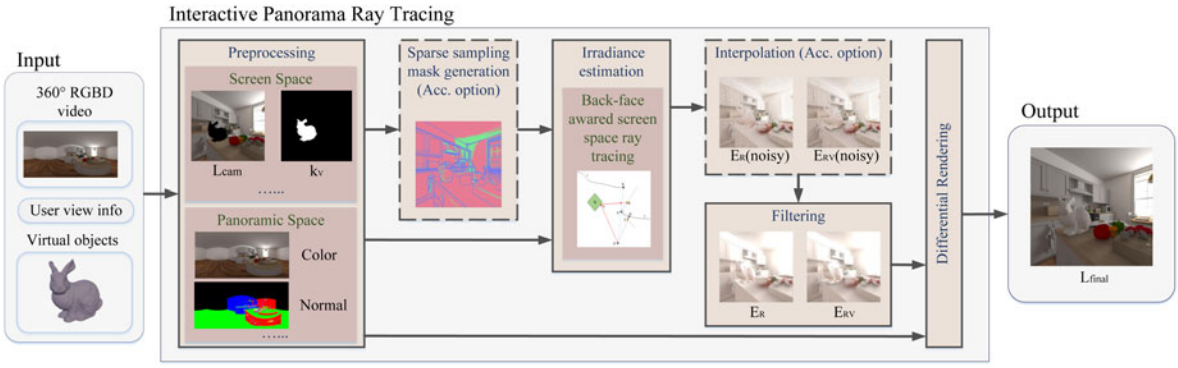

FIGURE 2. Our method merges virtual objects into a 360° RGBD frame and keeps the lighting, reflections, and shadows consistent. Taking the 360° RGBD video, virtual objects, and user's view as input, we first calculate the screen space and panoramic space geometry buffer of the merged scene, including color, normal, depth, etc., in the preprocessing pass. Then, an irradiance estimation and filtering pass are conducted to calculate the irradiance map (E_R and E_{RV}) of the merged scene inside the user's view. Finally, we can get the final result of virtual objects smoothly inserted into the real-world scene with a differential rendering pass. The sparse sampling mask generation and interpolation passes are acceleration options.

the differential rendering framework,[9] the final rendering image of the virtual-real hybrid scene can be calculated as follow:

$$L_{\text{final}} = (1 - M) \cdot k_V E_{RV} + M \cdot \left(\frac{E_{RV}}{E_R} L_{\text{cam}} \right) \tag{2}$$

where L_{final} is the final rendered image of the hybrid scene, M is the fraction of the pixel covered by the real-world environment, k_V is the virtual objects albedo texture, E_{RV} is the pixel's irradiance considering both real and virtual objects, E_R is the pixel's irradiance considering only real objects and L_{cam} is the original real-world G-buffer radiance image (color image). We can easily compute k_V, L_{cam}, and M in the preprocessing pass. When the filtering process is finished, we can get the two types of irradiance images of the scene (E_{RV} and E_R).

With the equation mentioned above (2), we can finally synthesize the merging result image (L_{final}) of the virtual objects and the real world 360° RGBD frame. Our approach is generic and can easily add glossy and specular virtual objects to our framework by representing the irradiance of the virtual object in the same buffer E_{RV} and the glossy BRDF separately using a material buffer.

The following sections introduce the system framework's irradiance estimation and sparse tracing acceleration components. In detail, we also introduce the backside-aware SSRT algorithm and the sparse tracing acceleration algorithm related.

IRRADIANCE ESTIMATION

Framework

In order to render an MR scene realistically, we must physically and correctly model the light interaction between the virtual objects and the real-world 360° RGBD frame. In our case, the 360° RGBD frame image is treated as an omnidirectional area light source; all visible pixels and virtual objects are illuminated by it. We start tracing light rays in the hybrid scene for each pixel. The sparse sampling mask from the last frame passes, and the rasterized screen and panorama space G-buffer (including position and normal) are used to decide whether the ray should be generated. With a virtual object, the original panoramic light source changes. In particular, the surface illumination near the virtual object changed, as the virtual object blocked many of the original light sources. If we simultaneously generate rays for virtual and real objects, the influences by inserting the virtual objects are missing. Instead, we separate the tracing into two steps. In the first step, we generate and trace rays for pixels on the real surfaces, then calculate and update the color of the pixels of the real objects. In the second step, we generate and trace rays into the virtual objects' pixels and use the updated image to calculate the virtual objects' appearance. The tracing process is illustrated in Algorithm 1. So there exist two types of rays in our tracing pass, one for real surface area and another for virtual objects area.

Algorithm 1. Irradiance Estimation Framework

Input: candidate tracing ray r, screen space G-buffer G_s, panoramic space G-buffer G_p
Output: irradiance E_{RV} and E_R

1: **if** $r.type \in RealObject$ **then**
2: E_R = PRT(r, G_p)
3: **if** i_v = IntersectionWithVirtual(r, G_s) **then**
4: r' = Reflection(r, i_v, G_s)
5: **if** !(E_{RV} = BASSRT(r', G_s)) **then**
6: E_{RV} = PRT(r', G_p)
7: **else**
8: $E_{RV} = E_R$
9: UpdateGbuffer(E_{RV}, E_R, G_s)
10: **if** $r.type \in virutalObject$ **then**
11: **if** i_v = IntersectionWithVirtual(r, G_s) **then**
12: r' = Reflection(r, i_v, G_s)
13: **if** !(E_{RV} = BASSRT(r', G_s)) **then**
14: E_{RV} = PRT(r', G_p)
15: **else**
16: E_{RV} = PRT(r, G_p)
17: **return** E_{RV}, E_R

For a candidate tracing ray r, whose ray type is represented as $r.type$, taking the screen space G-buffer G_s and the panoramic space G-buffer G_p as input, Algorithm 1 traces r to get the irradiance E_{RV} and E_R. In the first step, we check if the ray originated from a real object's surface. If so, we calculate E_R by tracing r in 360° RGBD frame image (lines 1-2). Then, we trace r in screen space. If it intersects with virtual objects (line 3), we calculate a randomized possible reflection ray r' according to r and the BRDFs of the virtual surface intersection i_v (line 4). Next, the new ray r' is traced in screen space to calculate E_{RV}. If there is no intersection in screen space, we trace r' in 360° RGBD frame image to get E_{RV} (lines 5-6). If r does not intersect with virtual objects (line 7), we set E_{RV} as E_R (line 8). When finished tracing rays in the real object's surfaces, we update the screen space g-buffers according to E_{RV} and E_R on real surfaces (line 9). Then in the second step, similar to above, we start calculating E_{RV} on the virtual object's surface (lines 10-16). Although we put their pseudocode together, we implement the two steps separately in two passes in practice.

Our PRT algorithm is based on the SSRT approach[24] and is running at a relatively high frame rate. For each ray, we first trace in screen space to find interactions within the screen. If not, we trace in panoramic image space to find interactions with the surface of the real-world environment. Our algorithm only considers interactions with virtual objects in screen space, which is efficient and simple. However, this can lead to abrupt changes in the viewing area when virtual objects move outside the field of view. To address this issue, we extend the size of the downstream frame buffer, which means that the actual frame buffer size is larger than the screen size. When a virtual object moves outside the field of view, it still affects the viewing area in its neighborhood. As shown in Figure 3, by extending the undergoing frame buffer size, when rotating the view direction from (a) to (b), outside gift boxes are still reflected on the surface of the bottle. Otherwise, the reflected image of the gift boxes disappears (c). In practice, we extend the undergoing frame buffer size 110% larger. The extending ratio cannot be set too large. Otherwise, the resulting image may suffer perceivable distortion.

FIGURE 3. In (a), a specular virtual bottle and two diffuse virtual gift boxes are inserted on the desk in the *Houseroom* scene.[33] The gift boxes are reflected on the surface of the bottle. (b) The gift boxes disappeared from the view frustum when the view direction rotated. By extending undergoing frame buffer size, the reflected boxes image is maintained. (c) Otherwise, the reflected boxes image disappears.

Algorithm 1 gives the whole framework of our irradiance estimation process. In terms of the PRT algorithm, we first implement the panoramic image space tracing[24] and then accelerate the algorithm by dynamically changing the marching step size using a precomputed mipmapped panoramic depth map.[25] After that, we propose the BASSRT method to alleviate the artifacts caused by the depth gaps in the RGBD panorama.

Back-Face Aware Screen Space Ray Tracing

As we mentioned before, since 360° RGBD frames store only a single layer of the real-world environment, some artifacts may appear on the surface of the inserted virtual object, especially glossy and specular materials. This is because the intersection point sometimes lies on a depth gap when tracing light in image space. The geometry behind the object suddenly disappears, often leading to the virtual object's color jumping or stretching. We introduce a BASSRT method

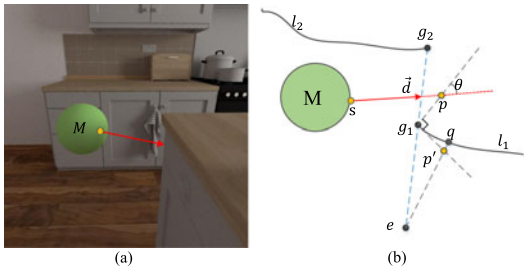

FIGURE 4. In (a), a tracing ray (red line) is marching to the right and goes into the gap between the desk and the cupboard. It causes a back-face missing problem that the back of the desk is unknown. (b) Shows the abstracted top view of the scene in (a). The tracing ray $s + t \cdot \vec{d}$ intersects with the gap between the desk (l_1) and the cupboard (l_2). The intersect footprint on the edge gives two split points g_1 and g_2. Instead of setting g_1 as the intersection, our BASSRT method chooses a substitute point q as the intersection. It uses its attributes to render the final appearance of point s on the virtual sphere M.

to mitigate the damage to the realism of rendered results. Our BASSRT method checks for missing back faces and finds suitable alternative intersections for the traced rays.

Figure 4 shows an example of back-face missing (a) and how our method deals with the discontinuity of the 360° RGBD frame geometry (b). In (a), a target tracing ray is marching to the right and goes into the gap between the desk edge and the cupboard behind. For this case, we do not know exactly which point the ray is intersected with since the back face of the desk is missing in our RGBD buffers. Traditional SSRT will set the intersecting point on the desk edge (g_1 in b) as the substitute intersection, which usually leads to the misslighted or stretched artifact on the virtual objects. Instead, we introduce a BASSRT method to alleviate this unnatural appearance. As shown in Figure 4(b), we give the top view of the setting map of (a). l_1 and l_2 are the layer of the 360° RGBD frame captured from center point e. In this context, l_1 is the desk front surface, and l_2 is the cupboard surface. g_1 and g_2 are the split points of the depth layer, where g_1 belongs to l_1 and g_2 belongs to l_2. VS is the virtual sphere we intend to insert. During the ray tracing process, a ray reflected from point s on the virtual sphere VS is marching into the gap between g_1 and g_2. The actual interact point should be behind the desk; we assume it is point p.

The main idea of our BASSRT method is that we assume that the back face should be similar to the front face of the objects. For example, in Figure 4(a), the back face of the desk should also be off-write and with a wooden top, just as the front face, which means, in (b), the appearance of back-face g_1p is similar to that of surface l_1. Then, If we choose a substitute intersection point from l_1, the appearance of the virtual sphere would be natural. Based on this assumption, we start to find the substitute intersection point. First, we construct a manipulation plane with the ray $\vec{d}$ and the viewpoint e, as shown in Figure 4(b). On this plane, we find the intersection point between ray $\vec{d}$ and the back surface g_1p, where g_1p is vertical to the tangent plane g_1p' of g_1. p' is located by setting the length $|g_1p|$ equal to length $|g_1p'|$. If the lengths are not equal, the final image of the back face on the virtual objects may be elongated or shortened. Finally, we connect ep' and extend it to intersect with l_1 at point q, which is set as the substitute intersection point between ray $\vec{d}$ and the 360° RGBD frame. It is worth noting that under some conditions, ray $\vec{d}$ may not intersect with g_1p when they are nearly parallel or getting away from each other; in other words, $\theta \leq 0$. Then, we set point g_1 as an invalid intersection point and keep tracing until the next intersection. We compare the rendered results with and without the BASSRT method later in the experimental section and discuss their difference.

SPARSE TRACING ACCELERATION

In general, when tracing light from the user's viewpoint toward the screen, we trace one ray for each pixel to obtain a Monte Carlo estimate of the dense image. Sometimes this approach is not effective because two physically adjacent points on a planar diffuse reflective surface usually receive similar irradiance from the deep panoramic surroundings. Based on this observation, we propose a sparse tracing acceleration method to speed up the ray-tracing process and ultimately accelerate our IPRT approach.

Sparse Sampling Mask Generation

The main idea behind sparse tracing acceleration is to generate sparse sampling masks to control whether or not to ray-trace a particular pixel. For each pixel in the screen, we consider the position and normal vector distribution of its neighboring pixels, as well as whether or not it intersects with a virtual object, to decide whether to perform ray tracing. The resultant tracing weight function is described as follows:

$$w_{i+1}(p) = \begin{cases} 1.0 & d_i(p) = 1 \\ \min(1.0, \max(||n_p - n_q||, \\ \sigma_p \cdot ||pos_p - pos_q||)), q \in \Omega & d_i(p) = 0 \end{cases} \quad (3)$$

where p is the target pixel and q is the adjacent pixel of p in the neighborhood area Ω. In practice, we set Ω as a 5×5 pixel range. For frame i, $d_i(p)$ is a binary parameter computed in the irradiance estimation pass to indicate whether the pixel p has been illuminated by the virtual objects in the last ten frames. n_p and pos_p stand for the normal and position of p and the same for q. If $d_i(p) = 1$, p is an active pixel, which means it interacts with virtual objects, and we set the weight to 1.0. If $d_i(p) = 0$, for each q in Ω, we calculate the normal and position difference maximum value to p and set it as the tracing weight. σ_p is the normalization parameter for the pixel position difference, and we set it to 10.0 in practice.

With (3), we can get the tracing weight buffer, and then we separate the screen pixels into three areas with two thresholds δ_1 and δ_2 ($0 < \delta_1 < \delta_2 < 1$). For a pixel p, we record its tracing level l_p as follows:

$$l_p = \begin{cases} 0, & \delta_2 \leq w_p \leq 1 \\ 1, & \delta_1 \leq w_p < \delta_2 \\ 2, & 0 \leq w_p < \delta_1. \end{cases} \tag{4}$$

In practice, we set δ_1 as 0.3 and δ_2 as 0.5. To make the algorithm fit on GPU, we structure the whole screen space into a multiscale grid and emit the tracing rays in these areas with different densities according to the area they belong. As shown in Figure 5, for area level 0 (blue), 1 (green), and 2 (red), we trace 1, 1/4, and 1/16 rays for each pixel. Only pixels with coordinates that are a multiple of 4 are tracked in the pink area. Only pixels with coordinates that are a multiple of 2 are tracked in the green area. Only pixels with coordinates that are a multiple of 1 are tracked in the blue area. For example, the pixel P is inside the 1/4 spp area (green), but it is not on the grid vertex position (its coordinate is not a multiple of 2), so there is no ray traced on P.

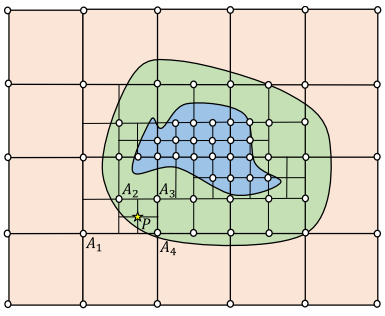

FIGURE 5. Three levels of tracing distribution in screen space. In the outermost area, we trace one ray for every 16 pixels (1/16 spp). In the middle area, we trace one ray for every four pixels (1/4 spp). We trace one ray for each pixel (1 spp) in the innermost area. Pixels that have traced rays are marked with white circles. The value is interpolated from the nearest traced pixels for a pixel P that has not been traced.

Interpolation

After panoramic ray tracing, we get a sparse irradiance image. In order to get a complete image, we perform an interpolation process to compute empty pixels by interpolating the nearest tracked pixels. For example, in Figure 5, the irradiance of an empty pixel P (yellow asterisk) is interpolated by that of A_1, A_2, A_3, and A_4. The process is introduced in Algorithm 2.

Algorithm 2. Interpolation

Input: sparse irradiance buffer Es, tracing level buffer L, pixel p, predefined coordinate offset array O
Output: interpolated irradiance buffer E_p

1: $l_p = L(p.x, p.y)$
2: **if** $l_p == 0$ **then**
3: $E_p = Es(p.x, p.y)$
4: **else**
5: **if** $l_p == 1$ **then**
6: **for** each $i \in [1, 4]$ **do**
7: $A_i.xy = 2 \cdot (p.xy/2 + O(i))$
8: $l_i = L(A_i.x, A_i.y)$
9: **if** $l_i == 2$ **then**
10: $A_i.xy = 4 \cdot (p.xy/4 + O(i))$
11: **if** $l_p == 2$ **then**
12: **for** each $i \in [1, 4]$ **do**
13: $A_i.xy = 4 \cdot (p.xy/4 + O(i))$
14: **for** each $i \in [1, 4]$ **do**
15: $dis_i = length(p.xy, A_i.xy)$
16: $E_p = E_p + dis_i \cdot Es(A_i.x, A_i.y)$
17: $dis = dis + dis_i$
18: $E_p = E_p/dis$
19: **return** E_p

For a pixel p in the screen, The algorithm takes its coordinate C_p, the sparse irradiance buffer E_s from panorama ray tracing pass, the tracing level buffer L, and a predefined offset array O as input. O is defined as $(0,0), (0,1), (1,0), (1,1)$ to get the coordinates of the adjacent traced pixel. The algorithm outputs the final irradiance of p. First, we get the tracing level of p in buffer L (line 1). If p belongs to level 0 (blue area in Figure 5), which means it is a ray-traced pixel, we get the final irradiance from E_s (lines 2-3). Then, if p belongs to level 1 (green area), we first find the adjacent pixels from the same level. If failed, we find a

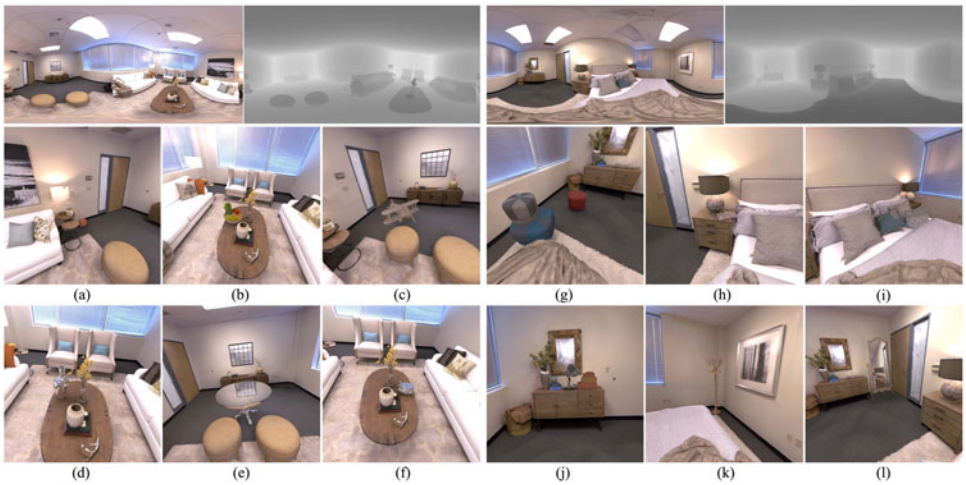

FIGURE 6. Augmented images of scenes *Apartment* (left half) and *Bedroom* (right half). The first row gives the two scenes' color and depth panorama image. From (a) to (f), we insert a basketball, a fruit basket, a wooden plane model, a glossy bottle, a specular table, and a glossy teapot into the scene *Apartment*. From (g) to (l), we insert a set of soft stools, a tissue box, a pillow, a lamp, a glossy coat hanger, and a specular mirror into the scene *Bedroom*. We set their roughness for all glossy virtual models as $\alpha = 0.05$.

substitute pixel from level 2 (light orange area). For example, we choose A_1 as the pixel p's adjacent marked with a yellow asterisk (lines 5-10). For the situation where p belongs to level 2, we find the adjacent pixels from the same level (lines 11-13). Pixels from grid level 2 are all traced. After getting the adjacent pixels, we interpolate p's final irradiance weighted by their distance to p (lines 14-19).

RESULTS AND DISCUSSION

We tested our IPRT method on two real-world and two synthetic 360° deep panoramic images. The system was deployed on a workstation with a 3.6 GHz Intel(R) Core(TM) i7-9700 K CPU, 32 GB of RAM, and an NVIDIA GeForce GTX 2080 graphics card. The resolution of the panoramic images is 4096×2048, and that of the user view is 1024×1024. All frame rate results are averaged over the latest 1000 frames of data. Since our system uses temporal filtering, the resultant images below are output after the frame sequence has stabilized (often 100 frames after program execution).

Quality

Figure 6 shows the rendering result of our IPRT method for two 360° RGBD frame scenes: *Apartment* and *Bedroom*.[3] We insert 12 virtual models into the two scenes with different diffuse, glossy, and specular materials. Our method can generate realistic global illumination effects on these virtual objects. For example, the lighting effect on the wing of the wooden plane (c) and the pillow (i) that changes with the environment, the reflections of the yellow leaves on the surfaces of the bottle (d) and the teapot (f), and also the images of the windows, the lamp, and the cupboard on the surface of the tabletop (e) and the mirror (l). In addition, our method can generate shadows on real surfaces caused by these inserted virtual objects, such as shadows next to basketballs, airplanes, soft stools, coat hangers, etc. Furthermore, we can see the spatially varying effects when the virtual object moves around in the scene; these global illumination effects of the inserted virtual models change coherently (please also reference our accompanying video).

We also render the images in Figure 6 with our sparse tracing acceleration strategy. We compare image quality (measured with MSE, SSIM, and PSNR) and rendering speed data (fps) in Table 1. With sparse tracing acceleration, the maximum MSE of *Apartment* is 10.50 while the rendering frame rate is improved from 98.3 to 107.4 fps, and the maximum MSE of *Bedroom* is 6.56 while the rendering frame rate is improved from 90.1 to 119.4 fps (1024×1024 resolution, average of 1000 frames). For all results, the SSIM values are over 0.96, and the PSNR values are over 37.92, which shows that our acceleration does not lead to an obvious quality decrease. Comparison results show that the error introduced by the sparse tracing acceleration is negligible, and the frame rate improvement is noticeable. These results validate that our sparse

TABLE 1. Quality (measured with MSE, SSIM, and PSNR) and frame rates (fps, averaged in 1000 frames) comparison with and without acceleration.

label	Acc. (Y/N)	fps	MSE	SSIM	PSNR
(a)	N	114.3	0.47	0.998	51.40
	Y	108.6			
(b)	N	119.9	3.26	0.990	42.99
	Y	107.0			
(c)	N	108.6	2.90	0.987	43.50
	Y	92.2			
(d)	N	107.4	10.50	0.990	37.92
	Y	98.3			
(e)	N	115.9	3.28	0.982	42.97
	Y	107.6			
(f)	N	120.3	5.74	0.994	40.54
	Y	103.7			
(g)	N	108.3	4.75	0.962	39.89
	Y	85.2			
(h)	N	111.9	2.05	0.994	45.00
	Y	90.2			
(i)	N	111.0	4.93	0.986	41.21
	Y	91.8			
(j)	N	113.1	5.63	0.982	40.62
	Y	93.2			
(k)	N	119.4	6.56	0.984	39.96
	Y	90.1			
(l)	N	118.2	4.83	0.977	41.94
	Y	106.8			

tracing acceleration can improve rendering speed while maintaining relatively acceptable quality.

Comparison in Synthetic Scenes

To further measure the quality of the rendered result of our IPRT method, we generate two 360° RGBD images from fixed viewpoints in two virtual 3-D scenes: *Kitchen* and *Living-room*.[34] Then, we insert the virtual Stanford Bunny and the Utah Teapot into *Kitchen*, and the Happy Buddha and the Stanford dragon into *Living-room*, one for diffuse and another for glossy material. The result images of our method are compared with images rendered by the IBL method and the ground truth images, which are rendered with full geometries of the scenes using the traditional path tracing method in Mitsuba.

FIGURE 7. Comparison between the images rendered with the IBL method, our IPRT method (Ours), and the ground truth (GT) in two synthetic scenes *Kitchen* and *Living-room*. Rows (A) and (D) show the 360° RGBD images of the two scenes, along with two user views. Rows (B), (C), (E), and (F) show the results of inserting *Stanford Bunny*, *Utah teapot*, *Happy Buddha*, and *Stanford Dragon* virtual objects into their corresponding views.

Figure 7 shows a comparison between the results of the IBL method, our IPRT method, and the ground truth. By contrast, we can see that our rendered inserting results are more similar to the ground truth rendered with whole geometry. In rows (B) and (E), our method produces the shadows of the virtual objects and their self-occluded effects. Our method in a row (C) has almost indistinguishable reflections of the surrounding environment and apparent interreflection effects (the handle image on the lid surface). In row (F), the real table and sofa parts have reflections on the virtual dragon surface. In summary, our IPRT method generates a more geometrically aware appearance of virtual objects with shadows on real surfaces, thus enhancing the user's spatial perception of virtual objects. In addition, since the IBL method does not take into account real-world geometry, it does not produce a coherent effect when virtual objects move. Our approach is to trace the rays of

TABLE 2. SSIM comparison for synthetic scenes in Figure 7, including whole image ($SSIM_w$), image cuts ($SSIM_{cut1}$, $SSIM_{cut2}$).

Object	Method	$SSIM_w$	$SSIM_{cut1}$	$SSIM_{cut2}$
Stanford bunny	IBL	0.828	0.791	0.778
	Ours	0.829	0.858	0.901
Utah Teapot	IBL	0.812	0.523	0.678
	Ours	0.812	0.825	0.784
Happy Buddha	IBL	0.755	0.541	0.657
	Ours	0.759	0.704	0.704
Stanford Dragon	IBL	0.687	0.315	0.424
	Ours	0.722	0.647	0.775

virtual objects and bounce them into a 360° RGBD frame to find intersection points. As virtual objects move, their appearance changes coherently, resulting in a spatially varying lighting effect. For more details, see our accompanying video.

Table 2 gives the SSIM results of IBL and our methods, compared with the ground truth in Figure 7. Comparisons were made between the whole image and the cuts of the two images. The results show that the SSIM results are similar for the whole image. For all image cuts, the SSIM results of our method are larger than those of the IBL method, which indicates that our method produces more reliable rendering results of virtual object regions than the IBL method. We believe this is because the virtual objects' regions are small, and the improvement in this region is diminished when the whole image is considered.

We introduced the BASSRT method during the ray tracing process to find a more reasonable intersection in the 360° RGBD frame. To verify the effectiveness of our method, we render two images in the synthetic scene *Kitchen* with and without the BASSRT method and compare them in Figure 8. Here we give three examples.

In the first row, without the BASSRT method, the image of the reflection of the red pepper on the shiny sphere would be severely stretched. Using our BASSRT method, the light ray is judged to determine if it encounters a depth gap at the edge of the pepper. We then continue to track the light so that it interacts with the back of the pepper or more surfaces in the scene, resulting in a reasonable image that is more similar to the ground truth. In the second row, we insert *Utah Teapot*. Without the BASSRT method, the reflected image of the teapot's surface would also be

FIGURE 8. Virtual *Sphere* (glossy, $\alpha = 0.1$), *Utah Teapot* (glossy, $\alpha = 0.1$), and *Stanford Lucy* (diffuse) rendered by our IPRT method with and without the BASSRT method, compared with the ground truth.

stretched. However, with the BASSRT method, we can see more detail in the teapot's surface, including the window's highlights, the wooden floor below the highlights, and the table's reflection. The combination of the reflective images and their structure makes the whole effect more acceptable compared to the ground truth. In the third row, we can see that without the BASSRT method, the light from behind the table would not have reached the surface of the virtual Stanford Lucy, resulting in an unusually dim effect of the diffusely reflecting material.

Our sparse tracing acceleration strategy used a parameter σ_p in (3). This parameter can affect the image quality. Figure 9 shows the rendering results of our method with/without sparse tracing acceleration. When σ_p increases from 10 to 1000, the SSIM results of the final rendered image cut increase (second row). This is because when σ_p increases, the maximum physical interior between two sparse pixel samples decreases to 0.001 m. That means the amount of samples increases, and the result image quality improves to that without sparse tracing acceleration. Meanwhile, the increasing sample amount also leads to decreasing frame rate.

Performance

Figure 10 and Table 3 show the quality and the performance comparison with and without sparse

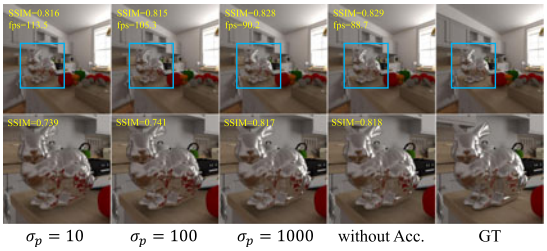

FIGURE 9. Rendering results of our sparse tracing acceleration (first three columns) with different parameters (σ_p = 10, 100, 1000). The fourth column is the rendering result without sparse tracing acceleration. The last column is the ground truth rendered with path tracing. The first row shows the original image, and the second row shows the image cut of the inserted virtual object (*Stanford Bunny*).

tracing acceleration in four scenes: *Kitchen* with the Stanford Bunny, *Living-room* with the Stanford Dragon, *Houseroom* with the Happy Buddha, and *Parlour* with the Lucy.[33] All four scenes are rendered at 1024×1024 resolution, and the data are averaged in 1000 frames. From the SSIM maps, we can see that the differences mainly come from virtual object regions. Because the lesser samples affect the *UpdateGbuffer* step in Algorithm 1, the reflection of the updated buffer changes. From Table 3, the SSIM values in all four scenes are over 0.96, which indicates that our acceleration strategy does not introduce significant quality reduction.

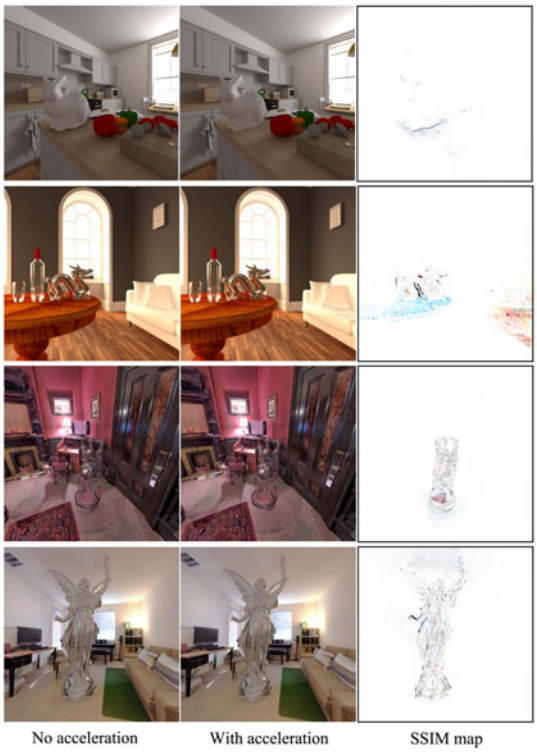

FIGURE 10. Comparison of image quality with and without our sparse tracing acceleration option. The left column shows the results for images without acceleration. The center column shows the results for images with acceleration. The right column shows the SSIM results graph.

CONCLUSION, LIMITATIONS, AND FUTURE WORK

We have presented a real-time rendering framework based on our IPRT method to facilitate photorealistic global illumination in MR for 360° RGBD video. Based on a fast irradiance estimation and a SOTA reconstruction filter, our method allows virtual and real dynamic objects. We give a panoramic ray tracing method to calculate the interaction between rays and the surrounding environment. A BASSRT algorithm is conducted to increase the realism of the result MR images. We have also provided a sparse tracing acceleration strategy to accelerate the frame rates of our framework further, taking into account the different performances of various graphics hardware. Furthermore, we evaluated our method in several captured real-world deep panoramic videos and compared our method's results with the ground truth rendered with the path tracing method in two synthetic scenes. The results showed that our method could produce photorealistic images at pretty high frame rates (over 100 fps).

Our approach enables real-time rendering of virtual and real fusion. It can bring many benefits to different groups. For example, it can create more engaging learning environments for students to interact with virtual science experiments, geographic landmarks, and so on. In the field of design and architecture, the proposed method allows architects to instantly visualize virtual designs in the real world. This allows better evaluation of design solutions and reduces errors.

One limitation of our method is that if the surface of the real world is shiny, our method is unable to render images of virtual objects reflected on it. Our approach is based on the differential rendering method, whose main assumption is that real-world entities are diffusely reflective. The difficulty in solving this problem is that it is difficult to instantly estimate/compute the material parameters of real-world surfaces to meet the high-performance requirements of MR applications. Possible future work could focus on

TABLE 3. Performance (ms) and quality (measured with SSIM) of our method in four scenes (averaged in 1000 frames).

Scene	With sparse tracing (Y/N)	G-buffer Gen.	Sparse mask	Ray tracing	Interpolation	Filtering	Differential Rendering	Total Time	Reduction	SSIM
Kitchen	N	1.08	-	5.53	-	4.37	0.30	11.28	18.3%	0.994
	Y	1.09	0.02	2.89	0.14	4.35	0.34	8.83		
Living-room	N	1.13	-	4.71	-	4.43	0.47	10.74	16.8%	0.977
	Y	1.05	0.03	2.99	0.19	4.37	0.31	8.94		
Bedroom	N	1.15	-	5.20	-	4.55	0.30	11.20	24.9%	0.983
	Y	0.91	0.02	2.62	0.14	4.43	0.29	8.41		
Apartment	N	1.11	-	4.78	-	4.33	0.37	10.59	11.0%	0.962
	Y	0.90	0.03	3.49	0.15	4.37	0.31	9.25		

estimating/computing the material of real-world surfaces (or even BRDFs) to facilitate multiple material interactions between virtual and real objects.

ACKNOWLEDGMENTS

This work was supported in part by the National Key R&D plan under Grant 2019YFC1521102, in part by the National Natural Science Foundation of China through Projects 61932003, in part by Beijing Science and Technology Plan Project Z221100007722004, and in part by the Academic Excellence Foundation of BUAA for Ph.D. Students.

REFERENCES

1. J. Kronander, F. Banterle, A. Gardner, E. Miandji, and J. Unger, "Photorealistic rendering of mixed reality scenes," in *Computer Graphics Forum*. Hoboken, NJ, USA: Wiley Online Library, 2015, vol. 34, pp. 643–665.
2. L. Gao, H. Bai, R. Lindeman, and M. Billinghurst, "Static local environment capturing and sharing for MR remote collaboration," in *Proc. SIGGRAPH Asia Mobile Graphics Interactive Appl.*, 2017, pp. 1–6.
3. M. Rey-Area, M. Yuan, and C. Richardt, "360MonoDepth: High-resolution 360° monocular depth estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 3762–3772.
4. P. Kán and H. Kaufmann, "Differential irradiance caching for fast high-quality light transport between virtual and real worlds," in *Proc. IEEE Int. Symp. Mixed Augmented Reality*, 2013, pp. 133–141.
5. M. Knecht, C. Traxler, C. Winklhofer, and M. Wimmer, "Reflective and refractive objects for mixed reality," *IEEE Trans. Vis. Comput. Graphics*, vol. 19, no. 4, pp. 576–582, Apr. 2013.
6. L. Gruber, T. Langlotz, P. Sen, T. Höherer, and D. Schmalstieg, "Efficient and robust radiance transfer for probeless photorealistic augmented reality," in *Proc. IEEE Virtual Reality*, 2014, pp. 15–20.
7. L. Gruber, J. Ventura, and D. Schmalstieg, "Image-space illumination for augmented reality in dynamic environments," in *Proc. IEEE Virtual Reality*, 2015, pp. 127–134.
8. M. Tuceryan et al., "An empirical evaluation of the performance of real-time illumination approaches: Realistic scenes in augmented reality," in *Proc. Int. Conf. Augmented Reality, Virtual Reality Comput. Graphics*, 2019, pp. 179–195.
9. S. U. Mehta, K. Kim, D. Pajak, K. Pulli, J. Kautz, and R. Ramamoorthi, "Filtering environment illumination for interactive physically-based rendering in mixed reality," in *Proc. EGSR (EII)*, pp. 107–118, 2015.
10. G. Xing, Y. Liu, H. Ling, X. Granier, and Y. Zhang, "Automatic spatially varying illumination recovery of indoor scenes based on a single RGB-D image," *IEEE Trans. Vis. Comput. Graphics*, vol. 26, no. 4, pp. 1672–1685, Apr. 2020.
11. J. Park, H. Park, S.-E. Yoon, and W. Woo, "Physically-inspired deep light estimation from a homogeneous-material object for mixed reality lighting," *IEEE Trans. Vis. Comput. Graphics*, vol. 26, no. 5, pp. 2002–2011, May 2020.
12. P. Pratul et al., "Lighthouse: Predicting lighting volumes for spatially-coherent illumination," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 8080–8089.
13. K. Karsch et al., "Automatic scene inference for 3D object compositing," *ACM Trans. Graphics*, vol. 33, no. 3, pp. 1–15, 2014.

14. M. Garon, K. Sunkavalli, S. Hadap, N. Carr, and J.-F. Lalonde, "Fast spatially-varying indoor lighting estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 6908–6917.
15. Z. Li, M. Shafiei, R. Ramamoorthi, K. Sunkavalli, and M. Chandraker, "Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting, and SVBRDF from a single image," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2475–2484.
16. T. Zhi et al., "Semantically supervised appearance decomposition for virtual staging from a single panorama," *ACM Trans. Graphics*, vol. 41, no. 4, pp. 1–15, 2022.
17. T. Rhee, L. Petikam, B. Allen, and A. Chalmers, "Mr360: Mixed reality rendering for 360 panoramic videos," *IEEE Trans. Vis. Comput. Graphics*, vol. 23, no. 4, pp. 1379–1388, Apr. 2017.
18. L. Wang, H. Wang, D. Dai, J. Leng, and X. Han, "Bidirectional Shadow Rendering for Interactive Mixed 360° Videos," in *Proc. IEEE Virtual Reality 3D User Interfaces*, 2021, pp. 170–178.
19. J. Tarko, J. Tompkin, and C. Richardt, "Real-time virtual object insertion for moving 360 videos," in *Proc. 17th Int. Conf. Virtual-Reality Continuum Appl. Ind.*, 2019, pp. 1–9.
20. W. R. Mark, L. McMillan, and G. Bishop, "Post-rendering 3D warping," in *Proc. Symp. Interactive 3D Graphics*, 1997, pp. 7–ff.
21. T. Sousa, N. Kasyan, and N. Schulz, "Secrets of cryengine 3 graphics technology," in *Proc. ACM SIGGRAPH*, vol. 1, 2011.
22. M. Mara, M. McGuire, D. Nowrouzezahrai, and D. Luebke, "Fast global illumination approximations on deep g-buffers," *NVIDIA Corporation*, vol. 2, no. 4, pp. 1–17, 2014.
23. A. Grace, R. Ollington, and I. Lewis, "Accurate image-space environment reflections in real-time," in *Proc. Int. Conf. Comput. Games, Multimedia Allied Technol.*, Global Science and Technology Forum, 2016, p. 7.
24. M. McGuire and M. Mara, "Efficient GPU screen-space ray tracing," *J. Comput. Graphics Techn.*, vol. 3, no. 4, pp. 73–85, 2014.
25. N. Hofmann, P. Bogendörfer, M. Stamminger, and K. Selgrad, "Hierarchical multi-layer screen-space ray tracing," in *Proc. High Perform. Graphics*. 2017, pp. 1–10.
26. B. Moon, J. A. Iglesias-Guitian, S.-E. Yoon, and K. Mitchell, "Adaptive rendering with linear predictions," *ACM Trans. Graphics*, vol. 34, no. 4, pp. 1–11, 2015.
27. L.-Q. Yan, S. U. Mehta, R. Ramamoorthi, and F. Durand, "Fast 4D sheared filtering for interactive rendering of distribution effects," *ACM Trans. Graphics*, vol. 35, no. 1, pp. 1–13, 2015.
28. C. Schied et al., "Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination," in *Proc. High Perform. Graphics*, 2017, pp. 1–12.
29. J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," *ACM Trans. Graphics*, vol. 26, no. 3, pp. 96–es, 2007.
30. N. Tatarchuk, B. Karis, M. Drobot, N. Schulz, J. Charles, and T. Mader, "Advances in real-time rendering in games, part I (full text not available)," in *Proc. ACM SIGGRAPH 2014 Courses*, 2014, p. 1.
31. P. Debevec, "Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography," in *Proc. ACM SIGGRAPH Classes*, 2008, pp. 1–10.
32. Z. Zeng, S. Liu, J. Yang, L. Wang, and L.-Q. Yan, "Temporally reliable motion vectors for real-time ray tracing," in *Computer Graphics Forum*. Hoboken, NJ, USA: Wiley Online Library, 2021, vol. 40, pp. 79–90.
33. J. Park, Q.-Y. Zhou, and V. Koltun, "Colored point cloud registration revisited," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 143–152.
34. B. Bitterli, "Rendering resources, 2016. [Online]. Available: https://benedikt-bitterli.me/resources/

JIAN WU is working toward the Ph.D degree with the School of Computer Science and Engineering, Beihang University, Beijing, 100191, China. His current research focuses on virtual reality, augmented reality, and visualization. Contact him at lanayawj@buaa.edu.cn.

LILI WANG is a professor with the School of Computer Science and Engineering, Beihang University, Beijing, 100191, China, and a researcher with the State Key Laboratory of Virtual Reality Technology and Systems. Her interests include virtual reality, augmented reality, mixed reality, real-time rendering, and realistic rendering. Wang received her Ph.D. degree from the Beihang University, Beijing, China. She is corresponding author of this article. Contact her at wanglily@buaa.edu.cn.