Interactive Mixed Reality Rendering on Holographic Pyramid

Danqing Dai*
Beihang University

Xuehuai Shi†
Beihang University

Lili Wang‡
Beihang University
Peng Cheng Laboratory

Xiangyu Li§
Beihang University

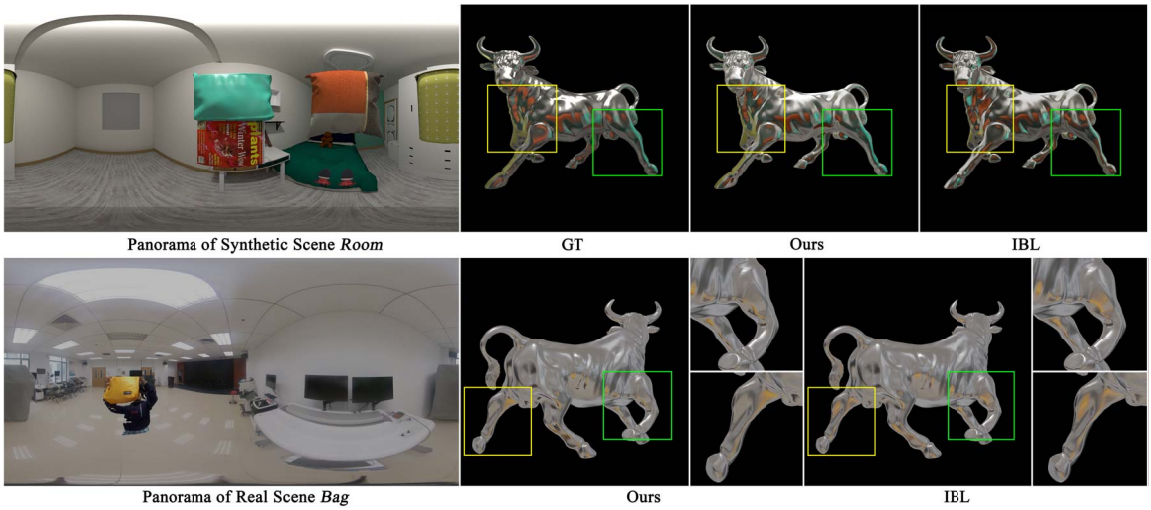

Figure 1: Reflection effects rendered by Monte Carlo path tracing (GT), our method (Ours) and the image-based lighting method (IBL) in the synthetic scene *Room* (top). Compared with IBL, Ours is closer to GT. *MSE* of Ours vs. IBL is 1.12×10^{-2} vs. 4.87×10^{-2}. In the real scene *Bag* (bottom), the foreground reflection effects of Ours are also obviously different from that of IBL. The framerate of our method is comparable to IBL: 37.43 fps vs. 38.78 fps.

Abstract

Currently, ray tracing and image-based lighting (IBL) have shortcomings when rendering the metallic virtual object displayed in the holographic pyramid in mixed reality. Ray tracing can hardly achieve the interactive frame rates, and IBL cannot accurately render the reflection result of the foreground near the virtual object. In this paper, we propose a mixed reality rendering method to render glossy and specular reflection effects on metallic virtual objects displayed in the holographic pyramid based on the surrounding real environment at interactive frame rates. First, we acquire the real environment data with four RGBD cameras and a panoramic camera; then, we introduce a foreground point cloud generation method to extract a temporally stable foreground point cloud from RGBD videos captured in real time; after this, we propose an efficient ray tracing method to render the dynamic glossy and specular reflections on the virtual objects that are displayed in the holographic pyramid. We test our method on several real and synthetic scenes. Compared with IBL and screen-space ray tracing, our method can generate the rendering results closer to the ground truth at the same time cost. Compared with Monte Carlo path tracing, our method is 2.5-4.5× faster in generating rendering results of the comparable quality.

Keywords: Mixed Reality,Virtual Reality, Reflection Rendering

*joint first author, e-mail: 838370860@qq.com
†joint first author, e-mail: shixuehuaireal@buaa.edu.cn
‡corresponding author, e-mail: wanglily@buaa.edu.cn
§e-mail: zy2106321@buaa.edu.cn

1 Introduction

In recent years, with the development of 3D acquisition and modeling technology, the accuracy of 3D models has become higher and higher, which makes it possible to present virtual objects with high fidelity. For example, in the museum, we can use the display equipment such as holographic pyramid to display 3D models of precious cultural relics that cannot be directly displayed. To present them more realistically, ideally these virtual cultural relics can reflect the light of the real scene. For instance, the audience's image can be reflected on a metal vase, and the audience's occlusion can change the lighting effect on the surface of the porcelain. These all require realistic mixed reality rendering. We can use the RGBD cameras to capture the surrounding environment, and then perform lighting calculations with virtual objects to generate mixed reality rendering results, which are presented in the holographic pyramid.

Image-based lighting (IBL) [6] is a real-time rendering technique commonly used in mixed reality (MR), which takes the illumination of the real scene as the environment map and uses this environment map to illuminate virtual objects. Ray tracing and IBL can render the reflection effects of the real environment on metallic virtual objects, but both methods have their shortcomings. The ray tracing method [42] can render the reflection effects of virtual objects accurately when we have the correct and complete geometric shape information of the real scene. However, the direct use of ray tracing to calculate the intersection of the light and the panoramic point cloud to render the glossy reflection effects of the metallic virtual object is too expensive to achieve interactive frame rates. In addition, because we use the RGBD camera to obtain real-time color and depth information of the surrounding real scene, the obtained information is incomplete, and the generated 3D point cloud is unstable. Therefore, the direct use of ray tracing methods will cause artifacts such as reflection color loss or distortion. IBL can render

2022 IEEE on Conference Virtual Reality and 3D User Interfaces (VR) | 978-1-6654-9617-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/VR51125.2022.00068

978-1-6654-9617-9/22/$31.00 ©2022 IEEE
DOI 10.1109/VR51125.2022.00068

the reflections efficiently, it assumes that the real scene around is far away from the virtual object displayed, renders the real scene into an environment map, and then uses this map to compute the reflections on the surface of the virtual object. Thus, if some objects in the real scene are close to the virtual object, we may get the wrong reflections.

In this paper, we propose a mixed reality rendering method for rendering virtual objects displayed in the holographic pyramid based on IBL and ray tracing at interactive frame rates. Our method can render glossy and specular reflection effects on the metallic virtual objects based on the real environment around the holographic pyramid. Firstly, we introduce a real-time mixed reality rendering pipeline, which uses IBL to render the static background reflection effects and uses an efficient ray tracing method to render the dynamic foreground reflection effects, so as to improve the rendering efficiency, and to avoid the problem of artifacts caused by the IBL method when the real object is too close to the virtual object. Then, we introduce a foreground point cloud generation method to extract the temporally stable foreground point cloud from RGBD videos captured in real time. At last, we propose an efficient ray tracing method that can render the dynamic foreground glossy and specular reflection effects on metallic virtual objects based on the surrounding real environment. We test our method on several real scenes and synthetic scenes. Compared with the rendering results using IBL [6], the results of our method are closer to the ground truth. Our method achieves interactive frame rates in the real dynamic environment.

In Fig. 1, we compare the ground truth rendered by using the Monte Carlo path tracing method [38] with 5 bounces and 8192 samples per pixel, the rendering results of our method (Ours), and the IBL method (IBL) for the synthetic scene *Room* (top). Our method shows better reflection effects than IBL. Some artifacts of IBL are shown in the cropped regions. The reflection effects of the cushion's orange part on the glossy bull's neck are stretched. The reflection on the hindleg of the glossy bull should have more green than orange. We also compare the result of our method with the result of IBL in the real scene *Bag* (bottom). The cropped regions indicate the difference between the results of the two methods. For a clearer presentation, we magnify the cropped regions and place them on the right. The reflection effects of the yellow bag on the glossy bull are deformed in IBL. This is because the yellow bag in the dynamic foreground is very close to the glossy bull, and the bag's shape is incorrectly described when constructing the environment map centered on the glossy bull, thus causing artifacts. Our method is efficient and achieves 37.4 fps on average.

In summary, the contributions of our method are as follows:

- An interactive mixed reality rendering method, which can render glossy and specular reflection effects on metallic virtual metal objects displayed in the holographic pyramid to simulate the reflection of the object to the surrounding real environment;
- A real-time foreground point cloud generation method that can extract the temporally stable foreground from RGBD videos and composite the 3D point cloud for the foreground;
- An efficient ray tracing method for rendering glossy and specular reflection effects on metallic virtual objects interactively.

2 Related Work

In this section, we first introduce the rendering frameworks of global illumination (GI) in MR in recent years. Then we discuss the depth filtering techniques, which is the basis of our foreground point cloud generation method.

2.1 Rendering Framework of GI in MR

Mixed reality technology refers to the technology that combines real video images with computer-generated images to enhance their practicability, which was proposed in Fournier et al. [9]. Then, they proposed a GI approximation method, which approximated the ambient light through the image intensity of real video images, and used classical radiosity computation to render the surface of computer-generated images. However, this method requires intensive computation and data preparation to solve the light transport equation. Due to many regions in the photograph remaining unchanged, and the same work being done twice without any visual effect in conventional differential rendering, Grosch. [13] constructed a differential photon map to store the changes in lighting introduced by virtual objects in MR, and performed rendering with only a single photon mapping simulation. Debevec [6] proposed the image-based lighting method (IBL) to render synthetic objects based on the environment illumination maps in MR. To simulate the GI effects of diffuse virtual objects in MR, Grosch. [14] took the HDR (high-dynamic range) image as the light source and computed irradiance volumes from the real region illuminated by the light source, then used irradiance volumes as indirect light sources to render diffuse virtual objects. Kautz et al. [20] proposed prefiltered method based on IBL to render glossy effects in real time. By interpolation calculating with prefiltered environment mipmap, this method can render different roughness metallic objects. Knecht et al. [22] proposed the differential instant radiosity method, which applied instant radiosity to the framework of differential rendering for rendering diffuse virtual objects in MR. Because using IBL to render the reflection effects of the near objects in the scene will cause serious artifacts, Pessoa et al. [31] extended IBL by back-projecting the near scene to the environment to generate a new environment map, then used the new environment map to perform rendering. However, the reflection results rendered by [31] still have physical errors. Santos et al. [8] constructed the virtual ghost objects from the RGBD images of the near scene, and used ray tracing to simulate the specular reflection or refraction effects of the near scene's object on virtual objects. Kan et al. [19] combined Santos's method [8] with the re-projection technique [13] to calculate radiance in the common camera image that does not have depth, then used ray tracing to render specular reflection and refraction effects on virtual objects. Lensing et al. [24] extended reflective shadow maps [5] based on RGBD image to simulate the first bounce on diffuse objects and achieve GI in both directions (real to virtual and vice versa) in MR. Meilland et al. [30] extended Pessoa's method [31], reconstructed the real scene into a 3D HDR environment map based on a dynamic set of images, and then generated the reflection environment maps at the virtual objects' position according to the real scene 3D model to render virtual objects' reflection effects. Rohmer et al. [36] introduced a differential rendering method for rendering diffuse and glossy virtual objects in MR on mobile devices. They projected the HDR images captured in different positions of the real scene into the atlas texture, which is divided into direct and indirect radiance as virtual area lights (VAL). This method can render high glossy reflection and color bleeding effects based on VALs. Mehta et al. [29] proposed a filter based on Fourier analysis to denoise the results from path-tracing to achieve physically correct global illumination in MR. Gruber et al. [15] regarded the surrounding of virtual objects in MR as indirect light sources, and proposed a method to render color bleeding effect between real and virtual object based on the screen-space directional occlusion [35]. Schwandt et al. [39, 40] proposed a method to generate a dynamic 360° environment map for IBL rendering in MR. Firstly, they generated the pre-computed static 360° environment map before rendering. Then they combined it with the real-time image stitchings for a continuous enhancement and update of the lighting information. Wang et al. [46] conducted a bidirectional shadow rendering method for 360° video in MR, which can render the interactive shadows between virtual and real objects.

Some studies have recently focused on learning lighting information such as environment maps or bidirectional reflectance distribution function (BRDF) parameters from RGB images using depth

learning models. Georgoulis et al. [10] presented a deep learning model to estimate the reflectance parameters and an illumination map from a single image that depicts a single-material specular object. Based on Georgoulis's method [10], LeGendre et al. [23] proposed a model that can estimate the reflectance parameters and the illumination map from the low dynamic range input images captured by mobile devices without high-precision images. Due to the previous deep learning models could only predict the lighting at individual 3D locations within the scene, Srinivasan et al. [43] introduced a deep learning model that estimates a 3D volumetric RGBα model of a scene, which can be used to calculate the incident illumination at any 3D location within that volume by standard volume rendering.

However, the above methods cannot render high-quality glossy and specular reflection effects at interactive frame rates in MR. The ray tracing-based methods need many samples to render glossy reflections, which is not efficient enough. The IBL-based methods would produce artifacts when rendering reflection effects for the environment with many dynamic foreground objects in MR. Our method performs mixed reality rendering to simulate glossy and specular reflection effects on virtual objects displayed in the holographic pyramid at interactive frame rates. Our method can achieve high-quality dynamic foreground reflection effects by our efficient ray tracing method.

2.2 Depth Filtering Techniques

Early depth filtering techniques mainly focused on the off-line depth processing of RGBD videos captured by infrared depth cameras to smooth the depth and fill depth gaps in RGBD videos. Marroquim et al. [26] proposed a GPU-based pull-push interpolation method that combined with elliptic box filters, which can reconstruct the rendering result in image space. Matyunin et al. [27] proposed a temporal depth filter to improve the temporal stability and fill occlusion regions of the depth in RGBD videos. However, this filter is an off-line filter and cannot process RGBD images in real time. Camplani et al. [3] combined the joint-bilateral filter and the Kalman filter to fill depth holes and smooth the depth of RGBD images. But this method is aimed at static scenes, which would lead to depth flicker in regions with drastic depth changes in dynamic scenes. Zhou et al. [47] proposed a super-resolution reconstruction for low-resolution depth images based on the modified joint trilateral filter. However, this method is an off-line method and does not consider filling the missing depth of low-resolution depth images.

Besides the off-line depth processing, many studies developed filters for handling depth in RGBD videos in near real time. Ritschel et al. [34] used the pull-push method to fill the depth gaps in the imperfect shadow maps. Knecht et al. [22] extended this work to mixed reality. Richardt et al. [33] developed a depth filter that can be used to smooth the depth of real-time RGBD images. They first removed typical artifacts in the RGBD images, then applied the denoising and upsampling scheme to RGBD images. However, the performance of this method achieves 5.2 fps for a 584×506 RGBD video. Camplani et al. [4] proposed a hole filling strategy based on a joint-bilateral filtering framework, which is used to fill the pixels that miss depth in RGBD images. Lin et al. [25] proposed an exemplar-based inpainting method to remove artifacts in depth maps obtained from RGBD cameras. The above two methods didn't focus on the flicker problem of depth values. Avetisyan et al. [1] refined the depth of RGBD videos using optical flow. They tracked every depth pixel over a sequence of frames in the temporal domain and used the same point's valid depth values to reduce the temporal noise and the flickering artifacts. This method needs the information of future frames, which is difficult to be applied to real-time scenes. Holynski et al. [18] presented an algorithm to propagate sparse depth to every pixel in MR, which needs the sparse simultaneous localization and mapping (SLAM) reconstruction as input. But this method could only roughly estimate the depth of each pixel in an RGB image according to the sparse SLAM reconstruction and can not directly filter the depth of the RGBD image.

We also use the depth filtering technique in our foreground point cloud generation. Our filter is based on the idea of pull-push, and uses temporal coherence of the adjacent RGBD frames to generate the temporally stable point cloud with rational depth.

3 Interactive Mixed Reality Rendering on Holographic Pyramid

We propose a mixed reality rendering method to render the reflection effects of the real environment on metallic virtual objects displayed in the holographic pyramid at interactive frame rates. Fig. 2 shows the five steps of our method.

Step 1 is Real Environment Data Acquisition, in which we use a panoramic camera and four depth cameras to capture the scene around the holographic pyramid. **Step 2 is Foreground Point cloud Generation**, in which we propose a real-time point cloud generation method to generate the temporally stable point cloud for the dynamic foreground. **Step 3 is Efficient Ray Tracing for Dynamic Foreground Reflections**, in which we propose a ray tracing method for rendering dynamic foreground glossy and specular reflection effects on the metallic virtual object interactively. The details of step 1, 2 and 3 are given in Sect. 3.1, Sect. 3.2 and Sect. 3.3 respectively.

In **Step 4: IBL for Rendering Static Background Reflections**, we use IBL [6] to render reflections of the static part of the real environment on the surface of metallic virtual objects. Firstly we generate the background blurred image pyramid ρ_θ based on the static background panorama θ. Then we initialize the static rendering framebuffer ψ_s, which is the same size as the reflection framebuffer ψ_r. ψ_r is generated in step 3, which is used to store the position, the reflected direction and the material of each intersecting point on the virtual object. For each pixel in ψ_s, we first index its intersecting point information in ψ_r, including position, material and reflection direction. Then we index the cubemap constructed from θ according to the position and reflection direction of the intersecting point, and index the specific level of ρ_θ according to the roughness of the material, so as to calculate the radiance corresponding to each pixel and store it in ψ_s.

In **Step 5: Rendering Results Composition**, we interpolate the rendering results of the dynamic rendering framebuffer ψ_d and the static rendering framebuffer ψ_s to composite the final rendering result ψ_f. A weight mask is also generated in Step 3. For each pixel px in the final rendering result ψ_f, we get a weight value w from the weight mask at the same position of px, and directly calculate the final radiance as $\psi_f[px] = \psi_d[px]*w + \psi_s[px]*(1-w)$. We apply HDR rendering [11] for both dynamic foreground reflections and static background reflections rendering.

3.1 Real Environment Data Acquisition

The device for displaying virtual objects includes a holographic pyramid, a panoramic camera, four depth cameras, and a graphics rendering workstation. Fig. 3 shows our hardware setups. The holographic pyramid consists of a base and an inverted glass pyramid. Four 19-inch monitors are lying flat on the base to display four front views of the virtual object. The inverted glass pyramid is placed above the base to reflect the content of four monitors. We render the virtual objects of mixed reality with the graphics rendering workstation, output the rendering results to four monitors, and then project them into the inverted glass pyramid. Users can see the result of mixed reality rendering on the inverted glass pyramid. The holographic pyramid, four depth cameras, and one panoramic camera are connected to the graphics rendering workstation with a 3.6 GHz Intel(R) Core(TM) i7-9700KF CPU, 16 GB memory, and an NVIDIA GeForce RTX 2080 SUPER graphics card. We take the center of the plane on the holographic pyramid's base as the origin

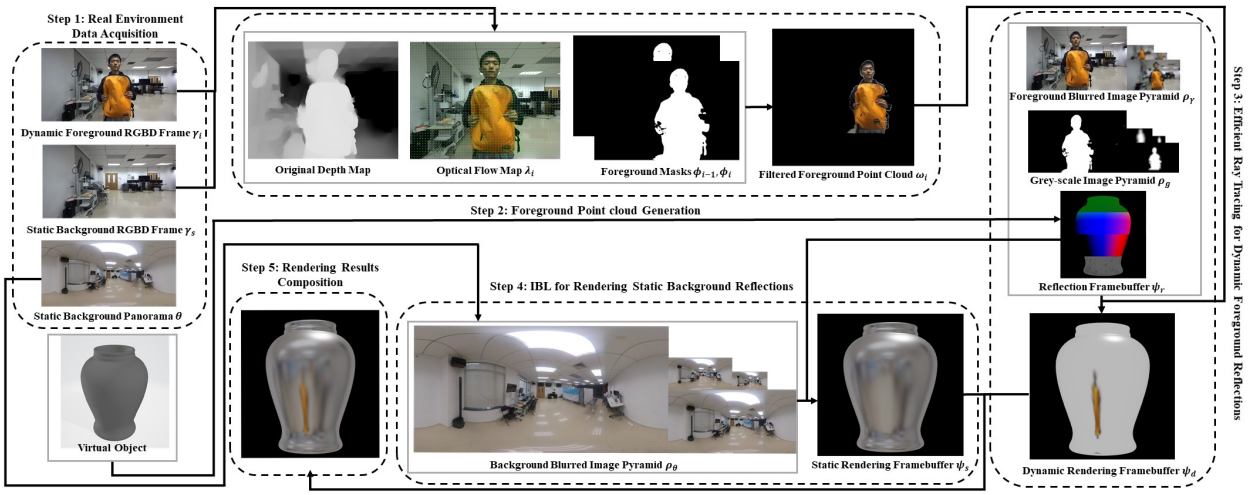

Figure 2: Pipeline of Our Method

O, and establish the MR coordinate system as shown in the Fig. 3. The positions of the panoramic camera, four depth cameras, and the virtual object are calibrated to the MR world coordinate system.

The model of the panoramic camera is Insta360 OneX, which has two 200° fish-eye cameras and 6080×3040 resolution for capturing panoramas. The panoramic camera is used to capture the static background panorama of the real scene around the holographic pyramid. Since the background around the holographic pyramid usually does not change, in the pre-processing, we remove the inverted glass pyramid and fix the panoramic camera above the base to capture the panoramic background. If the user wants to capture the dynamic background, the panoramic camera can be fixed above the center of the pyramid by a fixing device. However, if the fixed device is placed inside an inverted glass pyramid, the user may see the fixed device through the glass, which affects the visual effect.

Four depth cameras, ZED2 (110° horizontal FOV, 70° vertical FOV, $0.2 \sim 20m$ depth range, 720P resolution, and 60 fps video stream), are used to capture the RGBD frames of the real scene around the holographic pyramid from four directions, which are mounted on the four sides of the base. We don't use the panoramic camera to extract dynamic foreground in real time mainly for the following reasons: 1) there is no panoramic depth camera on the market currently, and we need the depth information of the foreground for rendering more accurate reflection effects; 2) the panoramic camera cannot be well fixed on the holographic pyramid.

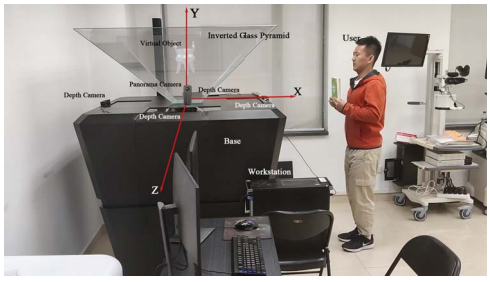

Figure 3: Our device setups, containing one holographic pyramid, four depth cameras, one panoramic camera, and one graphics rendering workstation.

Before the algorithm starts, we need to perform the pre-processing once. In the pre-processing, we remove all the dynamic objects around the holographic pyramid, use the panoramic camera to capture the static background panorama θ, and use each of the four depth cameras to take a static background RGBD frame γ_s. θ is used to generate the static part of the real scene. For the frame i in real time, each of these four depth cameras will capture the dynamic foreground RGBD frame γ_i of the current scene in real time when the user stands around the device to observe the virtual object displayed in the holographic pyramid. γ_s and γ_i are used to extract the dynamic part of the real scene.

3.2 Foreground Point Cloud Generation

In foreground point cloud generation, we first construct the optical flow map λ_i, and the foreground mask ϕ_i for the current frame i. After that, the pull-push based temporal depth filter is introduced to fill and filter the depth of the current dynamic foreground RGBD frame γ_i with considering of the last dynamic foreground RGBD frame γ_{i-1} and the last foreground mask ϕ_{i-1}. Finally, we generate the temporally stable point cloud ω_i for the dynamic foreground.

The optical flow map λ_i is constructed by the optical flow method [2] according to the current dynamic foreground RGBD frame γ_i and the last dynamic foreground RGBD frame γ_{i-1}. Each pixel in the optical flow map λ_i stores a 2D offset vector, which is used to index the precursor pixel of the corresponding pixel in the current dynamic foreground RGBD frame γ_i. Precursor pixel px_{pre} of a given pixel px in the current dynamic foreground RGBD frame γ_i means that px_{pre} in the last dynamic foreground RGBD frame γ_{i-1} shows the same real content as px. The foreground mask ϕ_i is constructed by MOG2 [48] according to the current dynamic foreground RGBD frame γ_i and the background RGBD frame γ_s. The foreground mask ϕ_i is a binary framebuffer and has the same width and height as the current dynamic foreground RGBD frame γ_i, which is used to indicate whether the corresponding pixel in γ_i is the dynamic foreground. If it is 1, it is the foreground,; but if it is 0, it is not.

Our pull-push based temporal depth filter is based on the traditional pull-push method [12]. The traditional pull-push method is often used to fill any gaps in the depth data for the image. It takes the image γ_i as the input, then output a filtered image according to the image pyramid ρ_{pa} that generated based on γ_i. Specifically, it takes γ_i as the 1st layer. Then, every four pixels in the lower layer are processed into one pixel and inserted into the upper layer from 2nd to l_{max}th layer in the step of pull. In the step of push, the pixels of the upper layer are divided into four pixels and inserted into the lower layer from $l_{max} - 1$th to 1st layer, and the 1st layer of the image pyramid ρ_{pa} is regarded as the output.

In our pull-push based temporal depth filter, besides taking current dynamic foreground RGBD frame γ_i and the maximum layer number of the image pyramid l_{max} as inputs, we also take optical flow map λ_i,

current foreground mask ϕ_i, last foreground mask ϕ_{i-1}, last dynamic foreground RGBD frame γ_{i-1}, probability distribution function of confidence PDF_α, probability distribution function of absolute value of depth difference $PDF_{|d|}$, the number of frames used to update the probability distribution function C, and current frame i as inputs.

For each pixel px in the dynamic foreground RGBD frame γ_i, we calculate the confidence α_{px} according to the RGBD values of px and its precursor pixel px_{pre}. The confidence α_{px} of px is used to describe the reliability of px's precursor pixel px_{pre} obtained by λ_i. The smaller px's confidence α_{px}, the more reliable the precursor pixel px_{pre} is. α_{px} is computed with Equation 1:

$$\alpha_{px} = \sqrt{\sum_{c \in r,g,b} (px.c - px_{pre}.c)^2} \tag{1}$$

The pull-push based temporal depth filtering is shown in algorithm 1. Firstly, we initialize a image pyramid ρ_{pa} in line 1. Each pixel in ρ_{pa} stores a *pair*, which contains two floats: depth d and confidence α. ρ_{pa} has l_{max} layers in total. d of each pixel px in 1st layer is the depth d of the corresponding pixel in γ_i, and α of px is temporarily set to 1.

Algorithm 1: Pull-push based Temporal Depth Filtering

input : current foreground mask ϕ_i, the optical flow map λ_i, current dynamic foreground RGBD frame γ_i, last dynamic foreground RGBD frame γ_{i-1}, last foreground mask ϕ_{i-1}, probability distribution function of confidence PDF_α, probability distribution function of absolute value of depth difference $PDF_{|d|}$, the number of frames for PDF update C, the current frame i

output : current dynamic foreground RGBD frame with processed depth γ_i

```
1  ρ_pa ← initImagePyramid(γ_i, l_max);
2  while l ∈ range(0, l_max) do
3      for block ∈ ρ_pa[l] do
4          paList ← initList();
5          if l == 1 then
6              for px ∈ γ_i[block] do
7                  px_pre, α_px ← GetPrePx (px, γ_i, γ_{i-1}, λ_i,
                     φ_i, φ_{i-1});
8                  pa ← initPa();
9                  |d| ← abs(px.d − px_pre.d);
10                 if i < C + 1 then
11                     PDF_|d| ←
                         updatePDF(PDF_|d|, |d|);
12                     PDF_α ←
                         updatePDF(PDF_α, α_px);
13                     return γ_i;
14                 else
15                     P_α ← PDF_α(α_px);
16                     P_|d| ← PDF_|d|(|d|);
17                     d̂ ← (1 − P_d) * px.d + P_d * px_pre.d;
18                     pa ← Pa(d̂, P_α);
19                 paList ← paList ∪ pa;
20         else
21             paList ← paList ∪ ρ_pa[l][block];
22         pa ← avg(minC(paList));
23         ρ_pa[l + 1][block] ← pa;
24 γ_i ← Push(ρ_pa, l_max, φ_i);
25 return γ_i;
```

Lines 2-22 show the process of pull. If the current layer l is lower than the highest layer l_{max}, we traverse the 1th layer of ρ_{pa} with 2×2 as a *block* (lines 2-3). We initialize the empty pair list *paList* in line 4. If the current layer l is 1, we will traverse each pixel px in *block* (lines 5-6). For each pixel px in *block*, we get the precursor pixel px_{pre} and the confidence α_{px} using Equation 1 of px (line 7). We initialize an empty pair pa in line 8. Then we calculate the absolute value of the depth difference $|d|$ between $px.d$ and $px_{pre}.d$ (line 9). In lines 10-18, we adjust the depth of the pixel px based on its confidence α_{px} and absolute value of the depth difference $|d|$. If the current frame i is within the first C frames of the system start to run (line 10), we count $|d|$ and get the probability distribution function of the absolute value of the depth difference $PDF_{|d|}$ (line 11), and count α_{px} and get the probability distribution function of the confidence PDF_α (line 12). We do not calculate a certain function expression but express the function by enough discrete function values in the domain. After that, we return directly without further processing(line 13). If the current frame i is not smaller than $C+1$ (line 14), we assume $PDF_{|d|}$ and PDF_α are stable enough. Then we get the probability P_α that the confidence is less than or equal to α_{px} (line 15), and the probability $P_{|d|}$ that the absolute value of the depth difference is less than or equal to $|d|$ (line 16). The larger of $|d|$, the smaller the probability of occurrence greater than or equal to this value. So we assume that the depth value of px is abnormal, and use more depth value of px_{pre} to correct it (line 17). We set the depth of pa as $\hat{d}$ and the confidence as P_α (line 18). Then we add pa to the list *paList* (line 19). If the current layer l is larger than 1, we only need to add all pairs in $\rho_{pa}[l][block]$ into list *paList* (lines 20-21). For interpolating each pair pa in $l+1$ layer, we get the average pair pa in $\rho_{pa}[l][block]$ with the minimum confidence (line 22), and assign pa to $\rho_{pa}[l+1][block]$ (line 23).

After pulling all layers in ρ_{pa}, we push the depth from l_{max}th layer to 1st layer to fill and filter the depth in the dynamic foreground part of γ_i according to ϕ_i (line 24) and return it (line 25).

3.3 Efficient Ray Tracing for Glossy and Specular Reflections on Virtual Objects

To render the dynamic foreground reflection effects on the virtual objects displayed in the holographic pyramid, an efficient ray tracing is proposed. The input of the ray tracing method is the foreground blurred image pyramid ρ_γ, the grey-scale image pyramid ρ_g, and the expanded foreground point cloud ω_i. The output is the dynamic rendering framebuffer ψ_d and the weight mask ϕ_w.

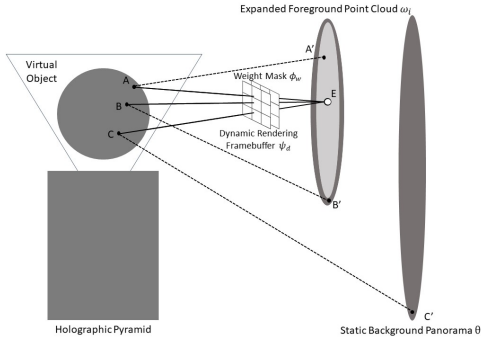

Figure 4: Procedure of the efficient ray tracing for rendering dynamic foreground reflection effects on the virtual object.

ρ_γ is an image pyramid to store the RGB of the dynamic foreground RGBD frame γ_i with different blurring levels in different layers. The higher layer of ρ_γ has higher blurring level, which is used to correspond to the higher roughness of the metallic virtual objects. ρ_g has the same size as ρ_γ. Each pixel in ρ_g stores the weight from 0 to 1, which is used to interpolate the corresponding pixel

in ρ_γ. We use KawaseBlur [21] to generate the foreground blurred image pyramid ρ_γ based on the dynamic foreground RGBD frame γ_i, and the grey-scale image pyramid ρ_g based on the foreground mask ϕ_i. ρ_γ and ρ_g both have l_{max} layers. Each pixel in the 1st layer of the foreground blurred image pyramid ρ_γ is the corresponding pixel in the dynamic foreground RGBD frame γ_i. Each value in the 1st layer of the grey-scale image pyramid ρ_g is the corresponding value in the foreground mask ϕ_i. We use the KawaseBlur method [21] to map each layer of ρ_γ and ρ_g to a blurred image with the specific roughness. The higher the layer of ρ_γ and ρ_g, the higher the corresponding roughness of this layer, so the higher the blurring of this layer's image. Examples of ρ_γ and ρ_g generated according to the real scene are shown in Fig. 5 (a) and (b).

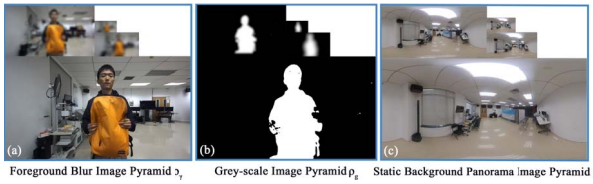

Figure 5: Real examples of (a) the foreground blurred image pyramid ρ_γ, (b) the grey-scale image pyramid ρ_g, and (c) the background blurred image pyramid ρ_θ generated according to the real scene.

ω_i is the foreground point cloud that expanded from the foreground point cloud generated in Sect. 3.2, which is used to prevent the reflection effects of the static background and the dynamic foreground on the virtual object from being too stiff. To expand the foreground point cloud ω_i, we obtain pixels in the dynamic foreground RGBD frame γ_i whose mask values are in the range of greater than 0 and less than 1 in the highest layer l_{max} of ρ_g, and add them to the point cloud ω_i in combination with the depth value d_{bor}. d_{bor} is computed by averaging all dynamic foreground edge pixels in ϕ_i.

ψ_d is the framebuffer that is used to store the dynamic foreground reflection effects on the virtual objects displayed in the holographic pyramid. ϕ_w is the mask that is used to store the weight for interpolating the rendering result of the dynamic foreground reflection and the static background reflection. We use the ray tracing-based reflection rendering method to render ψ_d and ϕ_w.

Fig. 4 shows our ray tracing procedure. A virtual object is displayed in the holographic pyramid. We trace rays from the eye (E) to the virtual objects (gray sphere) and use *hitpoint*s to refer to the intersecting points on the virtual object (A,B and C). Then the rays are reflected from *hitpoint*s to the dynamic foreground point cloud ω_i and the static background panorama θ, the intersections on ω_i and θ are referred to as *collisionpoint*s (A',B' and C').

The eye position (E) is captured by the Kernelized Correlation Filter [17] according to RGBD data in real time. Then, we use rasterization to generate *hitpoint*s on the virtual object based on the tracing paths (solid lines) from the eye through the pixels in the dynamic rendering framebuffer ψ_d. After that, we calculate the reflected rays (dotted lines) according to normal vectors of *hitpoint*s. We store the position, the reflected direction and the material of each *hitpoint* in the reflection framebuffer ψ_r, Then, we intersect the dynamic foreground point cloud ω_i and the static background panorama θ at the same time with the reflected rays. To intersect with ω_i, we voxelize ω_i and use one-bounce voxel walk [16] to generate the *collisionpoint*s on ω_i. To intersect with θ, we generate the cubemap based on θ [7] and index the corresponding *collisionpoint*s in the cubemap directly according to the *hitpoint*'s position and the direction of *hitpoint*'s reflected ray in ψ_r.

For each pixel px on the dynamic rendering framebuffer ψ_d, if px's *collisionpoint* is on the foreground point cloud ω_i, we calculate the radiance $\psi_d[px]$ according to the foreground blurred image pyramid ρ_γ, and the weight $\phi_w[px]$ according to the grey-scale image pyramid ρ_g. If px's *collisionpoint* is not on ω_i, we only need to calculate the radiance according to the static part of the real scene, and assign the radiance $\psi_d[px]$ and the weight $\phi_w[px]$ to null. Fig. 5 shows the real examples of (a) the foreground blurred image pyramid ρ_γ, (b) the grey-scale image pyramid ρ_g, and (c) the background blurred image pyramid ρ_θ.

The *collisionpoint* A' and B' in Fig. 4, we index the corresponding pixel position pa in the foreground blurred image pyramid ρ_γ and index the specific layer l on ρ_γ according to the object material on the corresponding *hitpoint*, and finally assign the radiance of $\rho_\gamma[l][pa]$ to $\psi_d[px]$ and assign the weight of $\rho_g[l][pa]$ to $\phi_w[px]$. The *collisionpoint* C' in Fig. 4 is not on ω_i, so we set the radiance $\psi_d[px]$ and the weight $\phi_w[px]$ to null.

4 Results and Discussion

In this section, we tested our method with quality experiments and performance experiments. In the quality experiments, we compared the visual quality between our method and the image based lighting method (IBL) [6]. Both real scenes and synthetic scenes are used, because we want to see not only qualitative quality results, but also quantitative analysis. In order to prove the depth rationality and temporal stability of the generated foreground point cloud, we also tested our foreground point cloud generation method separately in real scenes. In the performance experiments, we compared the time cost of each step between our method and IBL.

4.1 Quality Experiments

4.1.1 Real Scenes

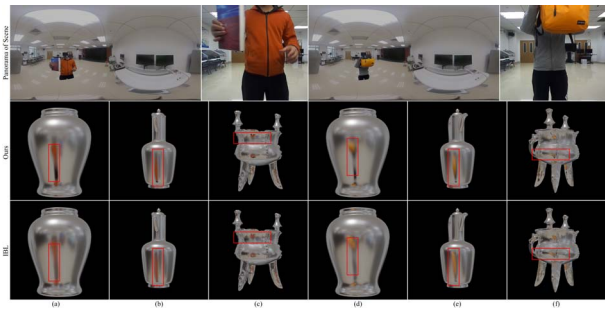

Figure 6: The comparison between Ours and IBL in real scenes: *Book* and *Bag*, with *Vase*, *Kettle* and *Tripod*.

Fig. 6 compares the rendering quality between our method and IBL in two real scenes *Book* and *Bag* with three different virtual objects *Vase*, *Kettle* and *Tripod*. Line 1 of Fig. 6 shows the panorama and foreground RGBD images of two real scenes *Book* and *Bag*. Line 2 shows the rendering results of our method (Ours), and line 3 shows the rendering results of IBL.

Compared with Ours, some artifacts of IBL are shown in the cropped regions. Column (a), the user reflection effects on *Vase* are too blurry. Column (b), the book in the user's hand is stretched in the reflection effects on *Kettle*. Column (c), the shape of user reflection on *Tripod* is deformed, and user reflection effects on *Tripod* are too blurry. Columns (d), the yellow reflected by the foreground yellow bag on *Vase* is not obvious enough and too blurry. Columns (e), the reflection shape of the foreground yellow bag on *Vase* is stretched incorrectly. Column (f), the reflection of the foreground yellow bag on *Tripod* has the wrong shape and is too blurry. This is because the IBL method lacks the geometric information of the real scene object, and cannot correctly draw the reflection of the nearby real object on the surface of the virtual object [32].

We compare the rendering quality of our method (Ours) and the screen-space ray tracing method (SSRT) [28] at the same time cost in Fig. 7. We magnify the cropped regions and place them on the right. Compared with Ours, the reflection effects of the yellow bag on *Vase* have noisy points in the green cropped region and holes in

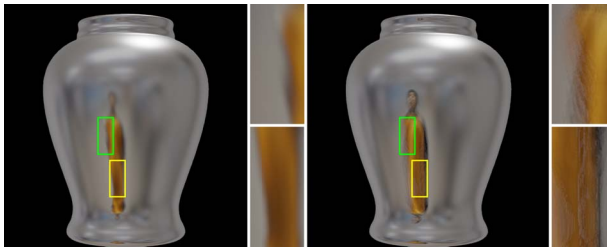

Figure 7: The comparison between Ours (left) and SSRT (right) in the real scene *Bag* with *Vase*.

the yellow cropped region in SSRT. The reason for SSRT having noisy points is that it cannot render high-quality glossy reflection effects with few samples per pixel (spp), while our method can use 1 spp to simulate the glossy reflection effects on the virtual object. The reason for SSRT having holes is that some pixels are lost during the reprojection of the RGBD image. Using SSRT, we need to reproject the RGBD image captured by the depth camera using the camera located at the origin O of the MR world coordinate. During reprojection, multiple pixels in the original RGBD image may overlap on the same pixel position in the new RGBD image, and only the nearest pixel is retained, while other pixels are discarded. Therefore, the intersection calculation may return null values and generate holes.

4.1.2 Synthetic Scenes

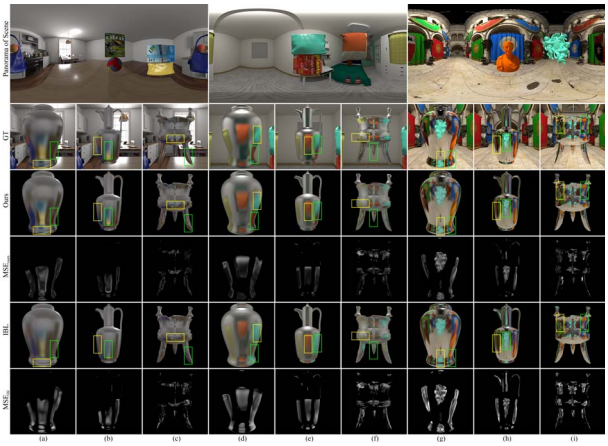

Figure 8: The comparison among GT, Ours and IBL in synthetic scenes: *Kitchen*, *Room* and *Sponza*, with *Vase*, *Kettle* and *Tripod*.

To quantitatively test our method, the rendering quality of our method is compared with those of ground truth (GT) and IBL in synthetic scenes. The rendering results are shown in Fig. 8. Line 1 shows the panorama of *Kitchen*, *Sponza*, *Room*. The ground truth images are obtained using the Monte Carlo path tracing method [38] with 5 bounces and 8192 samples per pixel (spp) (GT, line 2). The rendering results and *MSE* visualizations of our method (Ours) are shown in line 3 and line 4. The rendering results and *MSE* visualizations of IBL are shown in line 5 and line 6. To construct synthetic scenes for GT and IBL, we first put all virtual objects included in synthetic scenes into the MR coordinate system, and project the depth cameras and the panoramic camera in the MR coordinate system. Then we render the static background panorama θ according to the panoramic camera, render the static background RGBD frame γ_s and the dynamic foreground RGBD frame γ_i according to the depth cameras. After that, we directly extract the foreground point cloud ω_i based on the static background RGBD frame γ_s and the dynamic foreground RGBD frame γ_i. Then we combine ω_i with θ to construct synthetic scenes.

The results of Ours are closer to the results of GT than those of IBL. Some artifacts of IBL are shown in the cropped regions. In *Kitchen*: column (a), the blue color reflected on the base of *Vase* due to the reflection effects of the blue book, and the red color reflected on the body of *Vase* due to the reflection effects of the red part of the ball should be more obvious; column (b), the reflection color of the apple on *Kettle* is not obvious enough, and the reflection color of blue books and yellow bags on *Kettle* is offset; column (c), the reflection color of the blue book on *Tripod* should be more obvious, and the reflection color of the foot should be mainly yellow rather than blue. In *Room*: column (d), the reflection color of the green leather bag on the body of *Vase* is overstretched, and the reflection color of the red book on *Vase* body is not obvious enough; column (e), the reflection color of the orange part on the cushion and the green leather bag on *Kettle* are offset; column (f), the body of *Tripod* should have obvious yellow reflection formed by the yellow cushion, and the reflection color of the foot should be mainly orange rather than blue. In *Sponza*: column (g), the reflection color of the green sculpture and the orange sculpture at the bottom of *Vase* is offset; column (h), the reflections of shield and green sculpture on *Kettle* have the wrong stretching effects; column (i), the reflection of shield and orange sculpture on *Tripod* is deformed.

Because our method uses the same method as IBL to render the static background reflection region without introducing additional error, we focus on comparing the dynamic foreground reflection region on the virtual object. We quantify the quality of Ours and IBL with the mean squared error (*MSE*) for all the pixels in the dynamic reflection region of the virtual object with GT, which is computed by comparing GT and the ground truth image without dynamic foreground objects (absolute RGB distance ≥ 0.2). *MSE* of the rendering result is defined as the average squared Euclidean distance of all foreground pixels between the rendering result and GT [45].

Table 1: *MSE* ($\times 10^{-2}$) of Ours and IBL in the dynamic foreground reflection region

Scene	*Vase*		*Kettle*		*Tripod*	
	Ours	IBL	Ours	IBL	Ours	IBL
Kitchen	1.02	3.36	0.89	2.74	1.32	3.38
Room	1.13	4.21	0.79	3.15	2.19	4.89
Sponza	3.18	11.00	3.23	8.57	4.69	10.80

Table 1 shows the *MSE* comparison of our method and IBL for the images in Fig. 8. According to the table, *MSE* of our method is 2-4$\times$ smaller than IBL in all scenes. The images in line 4 of Fig. 8 visualize *MSE* for the images in line 3 that are rendered by our method, and the images in line 6 visualize *MSE* for the images in line 5 that are rendered by IBL, the whiter pixels represent the larger error. The error visualization shows that *MSE* of our method is always smaller than that of IBL.

4.1.3 Foreground Point Cloud Generation Test

In order to test our point cloud generation method $FPCG_{our}$, we compare $FPCG_{our}$ with the point cloud generation method based on median filter [44] $FPCG_{mf}$ and the point cloud generation method based on temporal anti-aliasing [37] $FPCG_{taa}$.

Fig. 9 shows the comparison of $FPCG_{our}$, $FPCG_{mf}$, and $FPCG_{taa}$ in the real scene *Book*, *Bag* and *Bucket*. In Fig. 9, column 1 shows the RGBD images taken from the position of the depth camera. For better observing the depth error of the generated foreground point clouds, we visualize the foreground point cloud from the position that moves 0.8 units along the negative direction of the z-axis and 0.2 units along the positive direction of the x-axis from

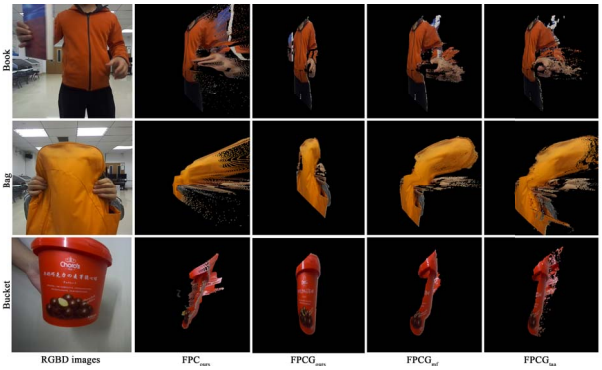

Figure 9: The comparison among $FPCG_{our}$, $FPCG_{mf}$ and $FPCG_{taa}$ in *Book*, *Bag* and *Bucket*.

Table 2: $MSE'(\text{cm}\times10^{-2})$ of $FPCG_{ours}$, $FPCG_{mf}$ and $FPCG_{taa}$

Scene	$FPCG_{our}$	$FPCG_{mf}$	$FPCG_{taa}$
Book	8.31	1765.02	935.12
Bag	3.72	481.09	219.77
Bucket	2.31	316.29	186.66

the original camera position. Column 2 shows the foreground point clouds FPC_{ori} that are directly generated from the RGBD images of column 1. It can be seen in column 2 of Fig. 9 that there are serious errors in the depth of FPC_{ori}. The depth of the point cloud generated by $FPCG_{mf}$ and $FPCG_{taa}$ is smoother, but due to apparent errors in the depth of the filling, the object in the point cloud is obviously deformed. The depth of the point cloud generated by $FPCG_{our}$ is smooth, and the object is not deformed.

We also measure the temporal mean square error MSE' for the depth of each foreground point cloud generated by $FPCG_{our}$, $FPCG_{mf}$ and $FPCG_{taa}$ in *Book*, *Bag* and *Bucket*. MSE' uses Equation 2 to calculate the average squared Euclidean distance of all pixels' depth between two adjacent frames in the (n-1) consecutive frame pairs [41].

$$MSE' = \frac{\sum_{i=2}^{n} MSE_{i,i-1}}{n-1} \tag{2}$$

Table 2 shows the comparison of MSE' of $FPCG_{ours}$, $FPCG_{mf}$ and $FPCG_{taa}$ in *Book*, *Bag* and *Bucket*. MSE' of $FPCG_{ours}$ is 129-212× smaller than that of $FPCG_{mf}$, and 59-112× smaller than that of $FPCG_{taa}$. The temporal depth of the foreground point cloud generated by $FPCG_{ours}$ is the most stable, and it is far from other foreground point cloud generation methods.

4.2 Performance Experiments

Table 3: Performance (ms) and fps of our method compared with IBL in different steps.

Scene		step 1	step 2	step 3	step 4	step 5	fps
Book	Ours	8.7	13.6	8.7	0.1	0.2	31.9
	IBL	8.8	2.4	-	14.6	-	38.8
Bag	Ours	9.0	13.7	10.5	0.1	0.2	29.9
	IBL	9.5	3.1	-	15.6	-	35.5
Bucket	Ours	9.1	12.8	8.5	0.2	0.3	32.4
	IBL	8.8	2.9	-	14.4	-	38.3

We give the time cost of each step of our method and IBL separately in Table 3. Our method has five steps: step 1 real environment data acquisition, step 2 foreground point cloud generation, step 3 efficient ray tracing for dynamic foreground reflections, step 4 image based lighting for rendering static background reflections, and step 5 rendering results composition. IBL has three steps: real environment data acquisition, foreground point cloud generation, and image based lighting for rendering both static background reflections and dynamic foreground reflections. The average frame rates of our method and IBL are also compared in Table 3.

In step 1, our method consumes as much time as IBL. In step 2, besides generating the foreground point cloud from the RGBD images, our method also needs to process the depth of the foreground point clouds. Our method consumes 9.93-11.23ms more than IBL. In step 3, our method needs to render the dynamic foreground reflection effects on the virtual object by the efficient ray tracing, which takes 8.50-10.50ms. In step 4, our method is 83-130× faster than IBL. This is because IBL renders all reflection effects in this step. IBL needs to re-project the foreground point cloud from the origin of the MR coordinate system to the center of the virtual object, and generate the environment map from the center of the virtual object by combining the foreground point cloud and the static background panorama, which is time-consuming. In step 5, we composite the render results of step 3 and step 4 for generating the final rendering result, which takes 0.17-0.29ms. From the fps data in Table 3, we can see that our method achieves the comparable frame rates as IBL in all scenes.

We compare the performance between our method and the Monte Carlo path tracing method implemented by Optix (MCPT_q, MCPT_t) with the virtual object *Vase* in three synthetic scenes: *Kitchen*, *Room*, and *Sponza* . With comparable *MSE* for *Kitchen*, *Room*, and *Sponza*, MCPT_q's frame rates are 7.2, 12.1, and 9.6 fps, and our method speeds up by 2.5-4.5×. In the case of the same rendering time for *Kitchen*, *Room*, and *Sponza*, *MSE* of MCPT_t is 4.86, 5.81 and 6.477×10^{-2}, while the *MSE* of our method is reduced by 1.8-3.9×.

5 Conclusion, Limitations and Future Work

We have proposed an interactive rendering method on the holographic pyramid, which can render glossy and specular reflection effects on metallic virtual objects based on the real environment around the holographic pyramid in real time. Compared with IBL, the rendering results of our method are more similar to that of path tracing with 5 bounces and 2048 samples per pixel, while maintaining the high frame rates comparable to that of IBL.

The first limitation of our method is that the depth of the filled points is uniformly set as the average depth of the current foreground edge point d_{bor} when expanding the foreground point cloud in step 3 of our method. Therefore, the expanded points of the foreground point cloud may have wrong depth. So the first future work is to combine the nearby points' depth of the filled points with d_{bor} to generate an adaptive depth for each filled point, so as to reduce the rendering artifacts caused by the wrong depth. The second limitation is that our method can not support transparent meterials. Because rendering transparent virtual objects needs multiple bounces, which is more challenging for rendering performance. So the other possible future work is to extend our method to support transparent materials while maintaining interactive frame rates.

Acknowledgments

This work was supported in part by the National Key R&D Plan 2019YFC1521102, by the National Natural Science Foundation of China through Projects 61932003 and 61772051, by the Beijing Natural Science Foundation L182016, by the Beijing Program for International S&T Cooperation Project Z191100001619003, by the funding of Shenzhen Research Institute of Big Data (Shenzhen 518000).

REFERENCES

[1] R. Avetisyan, C. Rosenke, M. Luboschik, and O. Staadt. Temporal filtering of depth images using optical flow. 2016.

[2] A. Burton and J. Radford. *Thinking in perspective: critical essays in the study of thought processes*, vol. 646. Routledge, 1978.

[3] M. Camplani and L. Salgado. Adaptive spatio-temporal filter for low-cost camera depth maps. In *2012 IEEE International Conference on Emerging Signal Processing Applications*, pp. 33–36. IEEE, 2012.

[4] M. Camplani and L. Salgado. Efficient spatio-temporal hole filling strategy for kinect depth maps. In *Three-dimensional image processing (3DIP) and applications Ii*, vol. 8290, p. 82900E. International Society for Optics and Photonics, 2012.

[5] C. Dachsbacher and M. Stamminger. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pp. 203–231, 2005.

[6] P. Debevec. Image-based lighting. In *ACM SIGGRAPH 2006 Courses*, pp. 4–es. 2006.

[7] P. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *ACM SIGGRAPH 2008 classes*, pp. 1–10. 2008.

[8] A. L. Dos Santos, D. Lemos, J. E. F. Lindoso, and V. Teichrieb. Real time ray tracing for augmented reality. In *2012 14th Symposium on Virtual and Augmented Reality*, pp. 131–140. IEEE, 2012.

[9] A. Fournier, A. Gunawan, and C. Romanzin. Common illumination between real and computer generated scenes. 1993.

[10] S. Georgoulis, K. Rematas, T. Ritschel, E. Gavves, M. Fritz, L. Van Gool, and T. Tuytelaars. Reflectance and natural illumination from single-material specular objects using deep learning. *IEEE transactions on pattern analysis and machine intelligence*, 40(8):1932–1947, 2017.

[11] N. Goodnight, R. Wang, C. Woolley, and G. Humphreys. Interactive time-dependent tone mapping using programmable graphics hardware. In *ACM SIGGRAPH 2005 Courses*, pp. 180–es. 2005.

[12] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 43–54, 1996.

[13] T. Grosch. Differential photon mapping-consistent augmentation of photographs with correction of all light paths. In *Eurographics (Short Presentations)*, pp. 53–56, 2005.

[14] T. Grosch, T. Eble, and S. Mueller. Consistent interactive augmentation of live camera images with correct near-field illumination. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, pp. 125–132, 2007.

[15] L. Gruber, J. Ventura, and D. Schmalstieg. Image-space illumination for augmented reality in dynamic environments. In *2015 IEEE Virtual Reality (VR)*, pp. 127–134. IEEE, 2015.

[16] S. Guntury and P. Narayanan. Raytracing dynamic scenes on the gpu using grids. *IEEE Transactions on Visualization and Computer Graphics*, 18(1):5–16, 2011.

[17] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE transactions on pattern analysis and machine intelligence*, 37(3):583–596, 2014.

[18] A. Holynski and J. Kopf. Fast depth densification for occlusion-aware augmented reality. *ACM Transactions on Graphics (ToG)*, 37(6):1–11, 2018.

[19] P. Kán and H. Kaufmann. High-quality reflections, refractions, and caustics in augmented reality and their contribution to visual coherence. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 99–108. IEEE, 2012.

[20] J. Kautz, P.-P. Vázquez, W. Heidrich, and H.-P. Seidel. A unified approach to prefiltered environment maps. In *Eurographics Workshop on Rendering Techniques*, pp. 185–196. Springer, 2000.

[21] M. Kawase. Frame buffer postprocessing effects in double-steal (wrechless). In *Game Developers Conference 2003, 3*, 2003.

[22] M. Knecht, C. Traxler, O. Mattausch, W. Purgathofer, and M. Wimmer. Differential instant radiosity for mixed reality. In *2010 IEEE international symposium on mixed and augmented reality*, pp. 99–107. IEEE, 2010.

[23] C. LeGendre, W.-C. Ma, G. Fyffe, J. Flynn, L. Charbonnel, J. Busch, and P. Debevec. Deeplight: Learning illumination for unconstrained mobile mixed reality. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5918–5928, 2019.

[24] P. Lensing and W. Broll. Instant indirect illumination for dynamic mixed reality scenes. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 109–118. IEEE, 2012.

[25] B.-S. Lin, M.-J. Su, P.-H. Cheng, P.-J. Tseng, and S.-J. Chen. Temporal and spatial denoising of depth maps. *Sensors*, 15(8):18506–18525, 2015.

[26] R. Marroquim, M. Kraus, and P. R. Cavalcanti. Efficient point-based rendering using image reconstruction. In *PBG@ Eurographics*, pp. 101–108, 2007.

[27] S. Matyunin, D. Vatolin, Y. Berdnikov, and M. Smirnov. Temporal filtering for depth maps generated by kinect depth camera. In *2011 3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON)*, pp. 1–4. IEEE, 2011.

[28] M. McGuire and M. Mara. Efficient gpu screen-space ray tracing. *Journal of Computer Graphics Techniques (JCGT)*, 3(4):73–85, 2014.

[29] 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 *EGSR (EI&I)*, pp. 107–118, 2015.

[30] M. Meilland, C. Barat, and A. Comport. 3d high dynamic range dense visual slam and its application to real-time object re-lighting. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 143–152. IEEE, 2013.

[31] S. A. Pessoa, G. d. S. Moura, J. P. S. d. M. Lima, V. Teichrieb, and J. Kelner. Rpr-sors: Real-time photorealistic rendering of synthetic objects into real scenes. *Computers & Graphics*, 36(2):50–69, 2012.

[32] V. Popescu, C. Mei, J. Dauble, and E. Sacks. Reflected-scene impostors for realistic reflections at interactive rates. In *Computer Graphics Forum*, vol. 25, pp. 313–322. Wiley Online Library, 2006.

[33] C. Richardt, C. Stoll, N. A. Dodgson, H.-P. Seidel, and C. Theobalt. Coherent spatiotemporal filtering, upsampling and rendering of rgbz videos. In *Computer graphics forum*, vol. 31, pp. 247–256. Wiley Online Library, 2012.

[34] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM transactions on graphics (tog)*, 27(5):1–8, 2008.

[35] T. Ritschel, T. Grosch, and H.-P. Seidel. Approximating dynamic global illumination in image space. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pp. 75–82, 2009.

[36] K. Rohmer, W. Büschel, R. Dachselt, and T. Grosch. Interactive near-field illumination for photorealistic augmented reality with varying materials on mobile devices. *IEEE transactions on visualization and computer graphics*, 21(12):1349–1362, 2015.

[37] C. Schied, A. Kaplanyan, C. Wyman, A. Patney, C. R. A. Chaitanya, J. Burgess, S. Liu, C. Dachsbacher, A. Lefohn, and M. Salvi. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*, pp. 1–12. 2017.

[38] V. Schüssler, E. Heitz, J. Hanika, and C. Dachsbacher. Microfacet-based normal mapping for robust monte carlo path tracing. *ACM Transactions on Graphics (TOG)*, 36(6):1–12, 2017.

[39] T. Schwandt and W. Broll. A single camera image based approach for glossy reflections in mixed reality applications. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 37–43. IEEE, 2016.

[40] T. Schwandt, C. Kunert, and W. Broll. Glossy reflections for mixed reality environments on mobile devices. In *2018 International Conference on Cyberworlds (CW)*, pp. 138–143. IEEE, 2018.

[41] S. P. Singh and P. Dayan. Analytical mean squared error curves in temporal difference learning. In *NIPS*, pp. 1054–1060, 1996.

[42] J. Spanier and E. M. Gelbard. *Monte Carlo principles and neutron transport problems*. Courier Corporation, 2008.

[43] P. P. Srinivasan, B. Mildenhall, M. Tancik, J. T. Barron, R. Tucker, and N. Snavely. Lighthouse: Predicting lighting volumes for spatially-coherent illumination. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8080–8089, 2020.

[44] J. W. Tukey et al. *Exploratory data analysis*, vol. 2. Reading, Mass.,

1977.

[45] T. Veldhuizen. Measures of image quality. *CVonline: The Evolving, Distributed, Non-Proprietary, On-Line Compendium of Computer Vision*, 2010.

[46] L. Wang, H. Wang, D. Dai, J. Leng, and X. Han. Bidirectional shadow rendering for interactive mixed 360° videos. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pp. 170–178. IEEE Computer Society, 2021.

[47] D. Zhou, R. Wang, X. Yang, Q. Zhang, and X. Wei. Depth image super-resolution reconstruction based on a modified joint trilateral filter. *Royal Society open science*, 6(1):181074, 2019.

[48] Z. Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 2, pp. 28–31 Vol.2, 2004.