Noname manuscript No.
(will be inserted by the editor)

Reflection Reprojection using Temporal Coherence

Naiwen Xie · **Lili Wang** ·
Philip Dutré

the date of receipt and acceptance should be inserted later

Abstract A powerful approach for rendering high-quality images at low cost is to exploit temporal coherence by projecting already computed images into a novel view. However, conventional temporal coherence projection methods assume pixel values remain almost unchanged from frame to frame, which does not extend well to reflection rendering. We present a novel projection method to reuse reflections from adjacent frames. A novel reflection reprojection method is introduced to establish the mapping of reflections between individual frames. By reusing the information from the reference frame, our method can reduce the overall workloads of reflection computation, which makes rendering efficiently.

Keywords Specular reflections · view-dependent rendering · temporal coherence rendering

1 Introduction

In most real-time 3D computer graphics applications, reflections are integrated to convey the relative position between reflectors and reflected objects, and to give users better shape and material perception. Still, most applications apply only environment mapping (an image-based approximation of reflectors surrounding the rendered object at a large distance) as a substitution for physically accurate reflection rendering.

The main reason is the complexity of tracing reflected rays into the scene. Nowadays, GPUs are powerful in computing the intersections of viewing rays and the scene in a feed-forward rendering pipeline fashion. Although they are efficient in generating the shading point at each pixel sample, they are unsuitable for modeling reflected rays which usually do not pass through a projection center that provide a fast triangle projection in the pipeline. Ray tracing is a general approach that naturally supports reflection rendering, but the overall performance is often a concern.

The idea of exploiting Temporal Coherence (TC) to alleviate expensive shading operations has evolved to handle multiple rendering effects such as diffuse global illumination rendering [14], soft shadows [28,26] and ambient occlusion [18]. The basic scenario is to generate a new frame by reprojecting data from a previously shaded frame. These approaches are efficient, in that TC rendering uses the correlation between adjacent frames rather than wastefully regenerating every pixel from scratch. The assumption of this approach is that pixel values remain almost unchanged, only depend on the scene geometry and configuration of the light source (view-independent component). The shading information of reference frames can be projected with depth value in-between viewports. This depth-based projection method is efficient to generate images containing view-independent components. However, it fails to render convincing images with scenes containing reflective materials. The reason is reflections is view-dependent and rendering rendering reflections imply solving the additional problem of tracing each reflected ray into the scene for each reflective pixel. Therefore, reflections cannot be approximated well by view-independent component and consequently, classic depth-based projection methods produce large errors for reflections as illustrated in Figure 1 and Figure 6.

In this paper, we propose a new TC reflection rendering method to handle the projection of reflections in-between two frames. Our method is a general method of TC projection and is based on the idea that most specular reflection colors shift rather than attach still to the surface on reflectors from frame to frame (Figure 2). We can reuse these coherences to amortize the total workloads since fetching the reflections shading information in the reference image is more effective than computing the intersection between the reflected ray and the scene. In our method, reflection colors and depth values calculated by ray-tracing are stored in the reference frame's buffers and are potentially projected between adjacent frames. An offset vector buffer is used to store per-pixel forward movement vector that approximates the shifting of reflections on the surface. The new image from the new viewpoint can be efficiently generated by shifting the reference image pixels based on the corresponding forward offset vector. For the reflective pixels whose reflections generated by reflected objects that do not exist in the reference frame's buffer, or those whose confidence of their footprints in the reference image is too low, ray trac-

Naiwen Xie · Lili Wang
Beihang University

Philip Dutré
KU Leuven

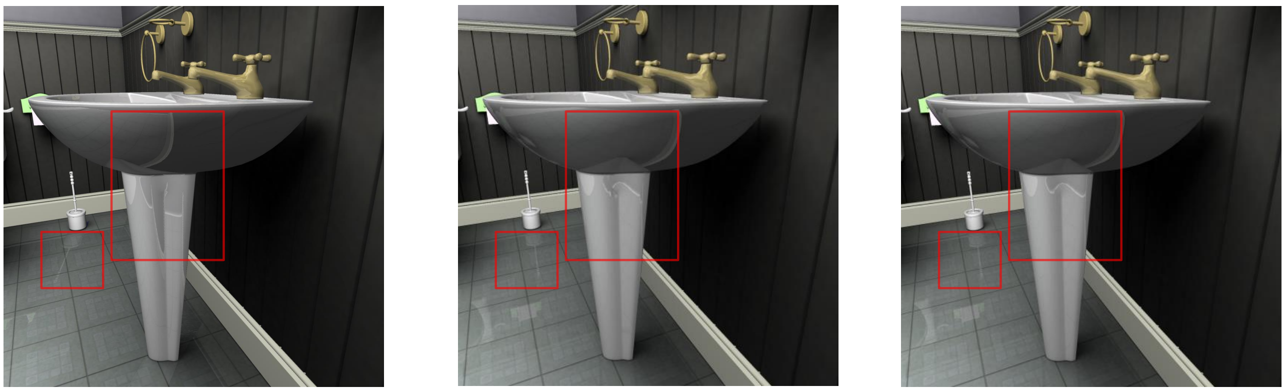

Fig. 1: *Specular reflections generated by naïve image warping (left), our method (middle) and ray-tracing (right). Naïve image warping produces inconsistent reflections because it assumes the reflection color remains unchanged in-between viewpoints. Our method produces convincing reflections compared to the ground truth while 77.1 percent of reflection intersection workloads are reduced in total.*

ing is used to re-calculate the reflections for these pixels. We reduce a large portion of reflection rendering workloads (77.1% for *Toilet*, 82.4% for *Living Room* in total) without introducing obvious errors for the subsequent frames and overall performance is improved.

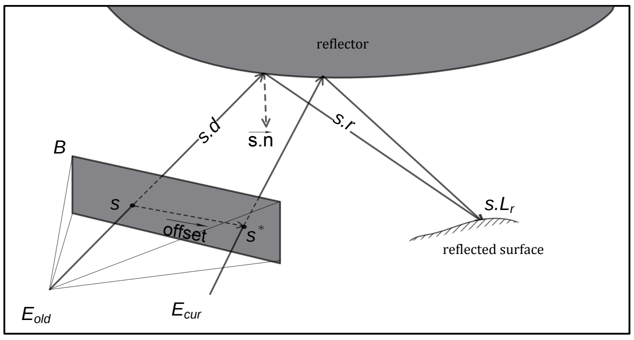

Fig. 2: Reflections shift in the reference image from s to s^* when the camera moves from E_{old} to E_{cur}.

The main contributions of our work are summarized as:

⋆ *Reflection buffers* introduced to represent the geometry surfaces intersected not only by viewing rays but also by reflection rays.
⋆ An analytic method for screen space pixel in reflection buffer to measure how well this pixels reflected geometry matches the reflection function when the incident ray shot from the current camera position.
⋆ A novel temporal coherent approach to accelerate reflection rendering by re-using reflections in reflection buffers.

Our method is independent of ray tracing techniques and can be integrated into most of ray-tracing methods which can return the reflected rays depth and value to reduce the workloads of reflection rendering.

2 Related work

2.1 Reflection Rendering

Ray tracing: The classical ray tracing method [36,7] can be used to render reflective objects, but is seldom used for interactive applications due to its high computation costs. Moreover some scenes including a lot of glossy or reflective objects with complex BRDFs are very expensive to simulate. CPU based ray tracing acceleration algorithms [34,21] and GPU based parallel algorithms [25] are able to improve performance. Reorganizing the 3D scene into KD-trees [5], BVHs [2] or LODs [37] gives further improvements of rendering speeds. Li et al. [16] used a texture lookup on the geometry fields to replace conventional expensive intersection testings, which achieved real-time frame rates. Yu et al. [39] explored a mipmap for geometry images to accelerate ray tracing, and can render glossy reflections interactively. While both geometry fields and geometry images the reflected objects in the scenes need to been generated with pre-computation, these two methods are hard to use for deformable objects.

Image-based rendering: Light field methods [8,15] capture reflections from different viewpoints, and synthesize reflections from the new viewpoints with indexing and interpolate the color from the sample images without significant extra cost. Heidrich et al. [11] created and stored ray-to-ray mapping in a lumigraph, which allowed the users to change reflective objects or the reflected scene independently. Taguchi et al. [32] captured light fields when moving the camera according to the reflectors rotation axis, and reduced the aliasing artifacts caused by planar light field for curved reflectors. Yu et al. [38] combined the light field with environment maps, and constructed an environment light field map, which can be used to render accurate reflections nearby. Light field based methods can only render static reflective scenes because of off-line light field sampling and generation.

Projection approximation: Projection approximation methods use the forward rendering pipeline to generate the reflection of a given reflector, and achieve high performance for interactive applications. Explosion map based methods [1] transfer and deform the potential reflected objects, and render them into a reflection image with ordinary perspective projections. Estalella et al. [4] rendered the vertices and normals of the reflector into textures, and executed local searching using the fragment shader, which allowed parallel computation of the virtual vertices. The reflection image was merged to the primary image to synthesize the final result. These three methods suffer three main problems. 1) They need to determine the reflectors and all potential reflected objects before the virtual reflected objects generation; 2) A lot of virtual reflected objects need to be constructed and rendered if the scene contains many objects; 3) There are some occlusion errors that still need to be fixed. Sample-based cameras [24] are presented to avoid the problems above. It is a group of pinhole cameras which are organized in a BSP tree, and can capture accurate reflections of curved reflectors.

Reflected-scene approximation methods: The main idea of the reflected-scene approximation method is to compute reflection rays according to the surface BRDFs, and then approximate the reflected objects with more simple geometries, such as a cubic environment map [12,9]. Kautz and McCool [13] introduced an optimization algorithm to filter specular environment maps to render the glossy reflection of the object with single or multiple lobes of BRDFs. Green et al. [10] used a nonlinear approximation of the BRDF as a weighted sum of isotropic Gaussian functions, and precomputed self-occlusion that enabled accurate and efficient prefiltered environment map rendering. Kalos et al. [31] added the distance values for environment map texels, and obtained the hit point by a reflection or refraction ray. Popescu et al. [23] presented a method to construct impostors with depth maps instead of real geometry to accelerate computation. Wang et al. [35] proposed an approach to approximate the geometry reflected by cluster cameras. McGuire and Mara [19] presented an efficient GPU solution for screen-space ray tracing and Vardis et al. [33] extended it on a multiview and multilayered scheme. Hybrid approaches combining screen-space approximation methods with ray tracing techniques were used to offer off-screen reflections [6] and achieve a significant performance improvement [3]. Our method fits in this category by regarding the previous rendered reflection image with depth values as an approximation of reflected objects.

2.2 Temporal Coherence Rendering

The temporal coherence approach reuses data from previously generated frames by reprojecting it into a new frame, and was proposed independently by Scherzer et al. [26] (referred as *Reverse Reprojection*) and Nehab et al. [20]. This technique records an off-screen buffer, the so-called *history buffer*, *payload buffer* or *cache* holding the previously generated pixel data. When rendering consecutive frames, data in the buffer is reprojected to its new image space position that represent the same world space position according to the scene motion whenever possible; otherwise a new value has to be calculated from scratch.

A variety of algorithms have been developed by using Reverse Reprojection approach as a framework, including rendering acceleration by reformulating previous data as temporal incremental [28,26]; quality increment by taking into account generated rendered results in previous frames [18, 30,29], or by relocating rendering workloads into a series of frames [27].

Reflective and refractive image warping is discussed in [17], and proposes an approach to synthesize novel images with reflection and refraction shading by image warping. In contrast to this research, we concentrate on rendering an accurate reflection rendering affect, and suggest a stable approach for projecting reflections into the new frame.

3 Reflections projection using Temporal Coherence

Our method renders reflections using TC with the following procedure. First, the diffuse component is generated by taking a rendering pass. Second, for the key frames, the reflections are generated with ray-tracing techniques and stored in the reflection buffers together with other attributes (Section 3.1) and followed by a blending operation. For the other frames, we reduce the total workloads of reflection rendering by projecting the reflection color from the corresponding key frame into the current view in the reflection projection stage (Section 3.2). In the final compositing stage, the reflection colors of the area where the reflection projection has failed have to be re-evaluated and a blend operation is used to combine different components to render the current frame (Section 3.3).

Algorithm 1 Rendering reflection with temporal coherence

```
1: Shade an image with diffuse components from the output viewpoint
2: if current rendering frame is the key frame then
3:     Generate reflection buffers
4:     Blend the output image with the reflection buffers
5: else
6:     Temporal coherence based reflection reprojection using the key frame reflection buffer
7:     Output image composition
8: end if
```

3.1 Reflection buffers generation

Inspired by *history buffers* from [26], we introduce the term *reflection buffers* as a series of textures representing the geometry surfaces intersected with the reflected rays as well as the viewing rays. To generate reflections in novel views, they are used to produce a reflection projection method to be applied to each pixel in the reflection buffers. The reflection buffers include a world normal texture, a depth texture, a material texture and a reflection texture.

Given a scene S modeled with triangle mesh consisting of diffuse and reflective objects, and a camera position E, for each pixel s, the reflection buffers B contain the following information (Figure 2):

B.1 : the normal $s.n$ and depth $s.d$;
B.2 : specular factor $s.f$ ranging from 1 for perfectly specular surface to 0 for perfectly diffuse surface;
B.3 : reflecting color $s.L_r$ encoding the illuminance of reflected surfel as RGB component;
B.4 : distance $s.r$ storing the reflected ray's traveling distance from the reflector surface to the reflected surfel.

Besides the data commonly required for shading diffuse computation in deferred shading such as *B.1*, other information like *B.3* and *B.4* is used to restore the shading and position information of reflected geometries via Snells Law, e.g., in Lochmann et al. [17]. After the reflection projection, the result image is rendered by interpolating the diffuse component and the projected reflections with the specular factor held in *B.2*. The first two parts in the reflection buffers (*B.1* and *B.2*) are generated by taking a shading pass over the entire scene. Normal and depth information are generated by interpolation of the hit triangle. The last two parts (*B.3* and *B.4*) can be generated with most reflection ray-tracing techniques. The reflection buffers are updated when rendering the reference image and reused to generate the following frame sequences.

3.2 Temporal coherence based reflection reprojection (TCRR)

Depth-based warping is used as a common method using temporal coherence. The main idea is to find which pixel in the new frame has also been recorded in the reference image by creating a temporal coherence mapping function from the history buffers into the current view. This mapping function describes a pixel's offset move vector in the reference image due to the camera movement between two frames.

With regard to a view-independent component mapping function, the mapping function is based on the notion that there is little shading difference over the visible surface between two consecutive frames. Given a current camera E_{cur}, for every pixel sample s in the reflection buffers with view-independent component and depth $s.d$, the sampled 3D surface position W can be calculated by back-projection into the world coordinate space. Hereby we define W together with the normal value $s.n$ as the *sampling geometry* which describes the tangent plane of the shading point intersected by viewing ray at sample s. Consequently, in the new image, we can simply use the Model View Projection Matrix of the current camera to find the corresponding location s_{cur}. This validity of depth projection is checked by comparing the depth difference between s_{cur} and s to a given threshold ε: $|s_{\text{cur}}.d - s.d| < \varepsilon$. To avoid the situation that multiple pixels map to the same new image location, the depth of the new pixel $s_{\text{cur}}.d$, as well as view-independent component is passed into reprojection procedure as the z value for the purpose of depth testing.

With regard to the view-dependent component mapping function, the assumption of depth-based warping is invalid because the view-dependent component varies according to the change in camera position. Taking reflection as an example, the reflective objects extend and perturb rays according to the law of reflection, and the hit-points of reflected rays vary accordingly. Thus, reflection changes second-order with respect to the camera movement.

We present a novel projection method to reuse reflections using temporal coherence. The idea comes from a generalization of the assumption in depth-based warping that reflections have coherence in-between frames by shifting over the reflector's surface when the camera moves. Consider the reflections in the reference frame, many of them can find their existence in subsequent frames. We exploit this idea by first finding the best reflection color matching pixels in the reflection buffers for each pixel in the current frame (Section 3.2.1). Then we project the reflections in reflection buffers towards its matched position in the current frame to generate an *intermediate reflection image* (Section 3.2.2) and finally warp this image into the novel view, as per the following steps (Figure 3):

1 *Reflection pixel matching* step. For every pixel s in reflection buffers B, this step determines the most fitting pixel position s^* in B that its reflection color best approximates the reflection color in the novel view. Afterwards, the offset from s to s^* , which approximates the reflections movement on the corresponding scene surfaces, is stored in an offset vector buffer (Figure 2).
2 *Reflection shifting* step. Given the offset vector buffer, the intermediate reflection image with shifted reflections is produced by projecting each pixel with its reflection color to the offsetting position indicated in the offset vector buffer.
3 *Reflection warping* step. Depth-based warping method is used to project the intermediate reflection image into the current view.

3.2.1 Reflection pixel matching

The reflection pixel matching step defines a per-pixel mapping in the reference image from its position to the best matched position with the most similar reflection color when the reflected ray's incident ray is shot from the current camera position. However, finding a mapping between the current and the reference reflection is challenging. The pixel in the current reflection defines a reflected ray, and the reference reflection defines a sampling of the scene geometry, so one option is to intersect the reflected ray with the sampled geometry, which is expensive. There is no closed form mapping between the current and reference reflection, since such a mapping depends on the scene geometry. Given a pixel in the current frame for which we have to compute the reflection, we search for the corresponding pixel in the reflection buffers. Our method searches the best matched pixel by giving an distance function to each pixel describing how well this pixel's sampling geometry matches the reflection function with the incident ray shot from the current camera position. The pixel with the minimum distance defines the best matched pixel. We design this distance function based on the following principles:

a : This function should be flexible, supportive of concave, planar and convex reflectors;
b : This function should be local, only taking the pixel in reflection buffers as input parameters;
c : This function should be convergent, the function value generated by the best matched point has the minimum value.

Given a pixel sample s in reflection buffers B and a camera moving from E_{old} to E_{cur}, the reflected object R position can be easily calculated by retrieving the reflection distance in reflection buffers. The distance function with respect to a pixel s_n defines how well s_n can approximate the reflected ray that emitted from E_{cur} and reflected to the target reflected position R. We regard the pixel with minimum distance as the best matched pixel of s when the camera movement is applied. Our method uses the distance from E_{cur} to the ray that starts at the mirroring point M of R and passes through s_n's sample surface W_n as the distance function (see Figure 4). The virtual position M is determined by mirroring R with respect to the sampling geometry's tangent plane defined by W_n and $s_n.n$. The distance value d with respect to s_n is then defined as the Euclidean distance from E_{cur} to the ray defined by M and W_n.

Given this distance function, selecting the best matched pixel in the reflection buffers with the minimum distance to the current camera position is still a computationally hard optimization problem because computing the analytical expression of the sampling geometry in entire image space is not feasible as they consist of arbitrary shapes of reflectors. We use a local neighborhood pixels search approach to find a pixel minimizing this distance function. The local neighborhood search method moves from the initial pixel to its best neighborhood pixel where the minimum distance is found and proceeds until an optimum is found or out of space. Our approach assumes the distance function converges monotonously within the reflector's projection area in the reference image. Otherwise this approach could be stuck in a locally optimal point when no improving pixels are found in the neighborhood. However, we found that even for detailed object surfaces, observers have difficulty in recognizing this false solution. For this reason, the neighborhood pixel searching solution results in a coherent image with quality reflection rendering. Our neighborhood pixel searching approach proceeds according to Algorithm 2. Note that, in neighborhood searching, the further away the distance from the current frame to the reference image, the more search steps are needed. For efficiency, the initial searching pixel can be offset using previous offset buffer to reduce the local search steps. When the new reference image is generated, we fill the offset buffer with an INVALID bit mask to indicate that the pixel searching should start from the beginning pixel. The reflector's silhouette is defined by a depth discontinuity in the depth texture of the reference image.

3.2.2 Reflection shifting

The reflection shifting step describes how the data in reflection buffers is retrieved, reused and mapped to generate the intermediate reflection image. In this step, we directly process the reflection buffers and project every pixel to its new position based on its offset value stored in offset vector buffer with a forward reprojection method. Most forward reprojection strategies require a pixel mapping technique to fill the new picture with the pixels in the reference image. Basically, there are two primary methods of pixel mapping: (1) the splatting method, i.e., splatting each example using a

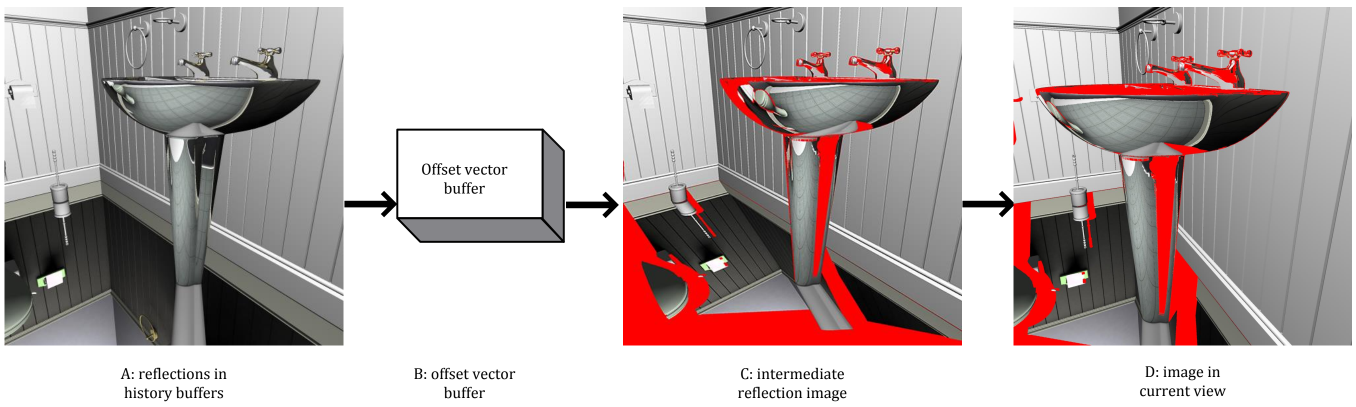

Fig. 3: TCRR pipeline. A: Specular reflection information is stored in the reflection buffers. Diffuse objects are visualized in grey. B: An offset vector buffer is generated after reflection pixel matching step. C: The intermediate reflection image is generated by shifting the reflection image with the value in the offset vector buffer in reflection shifting step. Pixels with no valid footprints in the reflection buffers are set an invalid bit mask (shown in red). D: The intermediate reflection image is projected with depth buffer in reflection warping step. After the TCRR, if an invalid bit mask is detected, a new reflection color is detected by the ray-tracing techniques; otherwise, the stored reflections are reused.

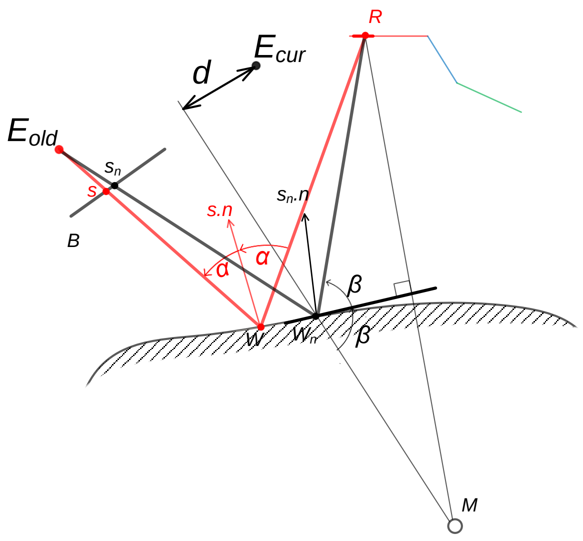

Fig. 4: Illustration of calculating the distance value with respect to s's neighbored pixel s_n

Algorithm 2 Neighborhood pixel searching for the best matched pixel

Input: reflection buffers B, the old viewpoint E_{old}, the new viewpoint E_{cur}, the offset vector buffer O_p from the previous frame.
Output: The updated offset vector buffer O_c recording the best matched pixel's offset vector at the current frame.

```
 1: for each reflective sample $s$ at location $(u,v)$ in $B$ do
 2:     Fetch sample's depth $s.d$, normal $s.n$, reflection depth $s.r$ at $(u,v)$ from $B$
 3:     Surface position $W \leftarrow \text{backProj}(s.r,u,v)$
 4:     Reflected position $R \leftarrow \text{reflect}(s.n,E_{\text{old}}) \cdot s.r + W$
 5:     Fetch sample's previous offset $o$ at $(u,v)$ from $O_p$
 6:     $s_c \leftarrow (o \neq \text{INVALID})\ ?\ s+o\ :\ s$
 7:     formerD $\leftarrow \text{distance}(s_c,R,E_{\text{cur}})$;
 8:     while $s_c$ is inside $B$ do
 9:         $N \leftarrow \{$a $3\times3$ hollow square neighborhood centered on $s_c\}$
10:         $(s_{\min},d_{\min}) \leftarrow \min_{\forall s_n\in N}\text{distance}(s_n,R,E_{\text{cur}})$
11:         if $d_{\min} >$ formerD then
12:             if $s_c$ is inside the reflector's silhouette then
13:                 record $s_c - s$ at $(u,v)$ in $O_c$ return
14:             else
15:                 record INVALID at $(u,v)$ in $O_c$ return
16:             end if
17:         end if
18:         formerD $\leftarrow d_{\min}$
19:         $s_c \leftarrow s_{\min}$
20:     end while
21:     record INVALID at $(u,v)$ in $O_c$
```

kernel function into a depth buffer with no explicit connectivity (2) The grid method, i.e., drawing meshes constructed by samples with explicit connectivity.

The multi-splatting rendering method has been developed that avoids the need of connectivity of samples. The difficulties for this approach is the estimation of the sample's output image footprint which has to be done with utmost accuracy to avoid having holes between neighboring samples, while preventing excessive overdraw. Meanwhile, the grid method utilizes connectivity information defined by uniform structure of a typical depth image. In that process, two triangles are used to connect the four adjacent pixels, unless there is a depth discontinuity that cuts through the pixels. As GPUs power has been increased dramatically, it can be done efficiently to render meshes by connecting depth image pixels into two triangles.

Our reflection projection method uses the grid approach because it achieves a watertight reflection rendering warping images with its explicit connectivity, while the holes in the splatting method will enlarge if the camera moves towards the reflectors. Even worse, the holes in the splatting method can be filled with other reflected objects due to motion parallax. Thus, hierarchical hole filling method such as [8] can hardly be used under these circumstances (Figure 5). Our method uses a grid mesh with the resolution of reflection buffers to generate the intermediate reflection image. The grid mesh's vertex (each corresponding to the center position in a pixel) is rendered to its new position by adding an offset given in the offset vector buffer, such that its attached reflection information is automatically warped into the new frame. Moreover, conventional depth buffer naturally resolves reflection overlaps (in Figure 3, the toilet brush overlaps the wall behind) by passing $s_d + s_r$ as depth value.

The grid meshes that are separated by a depth discontinuity form gaps in the intermediate reflection image due to motion parallax. These gaps, as well as the areas that are un-covered by the grid mesh due to invalid offset vectors, are set with an additional single bit mask indicating new reflections should be re-computed. If a shifted triangle's size is magnified above a certain threshold such that the confidence of extrapolated reflection color is too low, our method considers it as invalid as well.

3.3 Output image composition

The intermediate reflection image indicates the reflection component with respect to the current camera, attached to the sampling geometry surfaces of viewing rays from the camera belonging to the referable frame. A common depth-based warping is used to warp the reflection matching image into the current view. The disocclusion area caused by depth-based warping and the pixels with invalid bit mask are filled with new shading with reflections using a ray-tracer. The final pixel color is computed by interpolating the diffuse component and reflection component with weights defined by Fresnel terms computed from camera position, the geometry normal and the pixels specular level.

4 Results and discussion

4.1 Implementation

The effects of reflection rendering using temporal coherence are tested on two indoor scenes with specular reflections: *Living Room* (286K triangles, 156K diffuse, 132K reflective, Figure 6 top), *Toilet Room* (90K triangles, 44K diffuse, 46K reflective, Figure 1 and Figure 6 bottom). All measurements were recorded on a PC workstation with an Intel(R) Core i7-4720 CPU, with 16 GB of RAM and an NVIDIA GeForce GTX 980M graphics card. We use NVIDIA's Optix [22] ray tracer with the bounding volume hierarchy acceleration to compute the reflection color of reflection rays. Our method focuses on reducing the workloads of tracing reflected rays by projecting the reflections from previous frames into the current frame. Thus, we choose the naive projection method and the pure ray-tracer (Optix) as our control groups. Our method is independent of other ray tracing techniques and most methods which are able to return the reflected ray's depth and value can reduce their tasks with our method.

We implemented our method in a C++ framework. In reflection buffers generation, the first two recording parts (*B.1* and *B.2*) are generated with an OpenGL shader while the last two parts (*B.3* and *B.4*) are computed with Optix. Reflection buffers are stored as a two image-size RGBA textures by normalizing the normal factor into a two channel vector. We perform the reflection pixel matching (Section 3.2.1) step on the GPU in CUDA with one thread per pixel. Reflection shifting step (Section 3.2.2) requires invalid detection for each projected triangle primitive in the grid. We perform this step with an OpenGL geometry shader. In practice, we set the confidence threshold of area magnification as 16 (i.e. if the projected triangle's size is over 16 times of its original size, we discard this triangle.

4.2 Quality

As shown in Figure 1, Figure 6 and the accompanying video, our method generates specular reflections with a comparable quality to the ground truth. Figure 7 highlights the area recalculated by the ray tracer in red. Unlike naïve image warping, there is no obvious reflection discontinuity between reflection projections area and new value detection areas because we calculate reflections according to the current configuration of the viewing camera. Throughout this paper and the video, both output frames of our method and the control groups are rendered at a resolution of 1024 $\times$ 1024. The reflection buffers are generated using an equivalent 1024 $\times$ 1024 image resolution.

Our method renders high-quality specular reflections comparable to ray-tracing. Figure 8 shows difference images between our method and ray-tracing. The average absolute difference intensities are small, i.e. 0.13, 0.45, 0.12 for the three rows respectively. The values of difference intensities are measured by scaling RGB channels values from [0, 1] to [0, 255]. One reason for these artifacts is caused by local optimal in the pixel searching step where the searching step fails to converge to the global optimal solution. Another reason for the artifacts is the different sampling strategy of reflected geometry. Our method assumes the new image's pixel reflection colors exist in the reference reflection buffers and

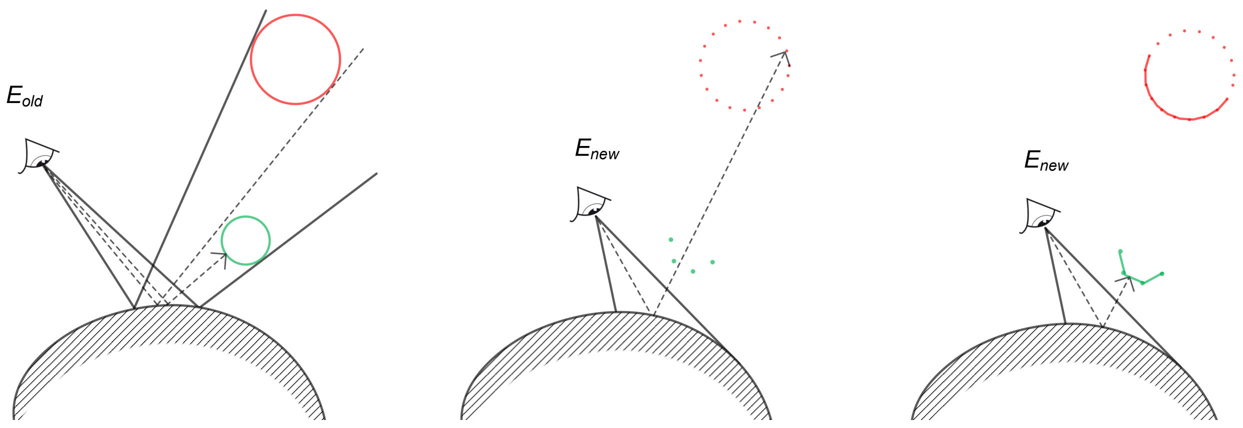

Fig. 5: Left: Reflection buffers sample reflected geometry surfaces. Middle: The splatting method suffers from holes magnification when the current camera moves close to the reflector. These holes can be filled with other reflected objects. The points in this figure correspond to pixels in the reference images. Right: The grid approach connects samples with similar depth and removes this issue

uses bilinear interpolation to sample the provided reflection buffer, while Optix generates the reflection color directly by computing the exact reflected ray with the scene. In reality, the actual reflection shading of one pixel in the current view (almost) never finds its footage in the reflection buffers due to the limited sampling rate.

Artifacts are also caused by disocclusion errors introduced by reflected surfaces that are not present in the reflection buffers but should be visible in the new image. The reflected rays in the new images do not travel the exact same path as in the reference image so it can happen that a few reflected rays miss the objects which are not captured in the reference image but hit other captured surfaces. Figure 9 illustrates this disocclusion error. One pillar is hidden behind another pillar in the reference image. In the reflection pixel matching step, different offset vectors are generated for the pixels of the front pillar and of the back wall with respect to the camera movement. In the reflection shifting step, the empty spaces behind the front pillar are filled with the color from the wall behind. Then the ray-tracer does not regard these areas as invalid. Thus no-further reflection recomputation is made and the disocclusion errors are made.

4.3 Reflection-Reuse Strategy

Reusing reflections involves certain quality/performance trade-offs. In term of quality, the reflection buffers can be projected into many frames continuously and they become stale with the artifacts discussed in Section 4.2. In term of performance, the total workloads of reflection rendering are amortized by TCRR since fetching the reflections shading information in the reference image is more effective than computing the intersection between the reflected ray and the scene. The overall time cost is largely limited by the reflection re-computing step. The time to do the TCRR is independent of the scene configuration and almost a constant. After the reflection projection, we use relatively time-consuming ray-tracer to fill the areas where TCRR fails to find valid footprints. Therefore, the time to create the novel reflections largely depends on the number of reflection re-computing pixels.

The total time-cost could be controlled to gain a certain quality/performance trade-off by setting a threshold for the total reflection re-computing pixels. We select the reference images by measuring the time-consuming ray-tracing pixels' ratio. At every frame, after TCRR, we count the ratio of the re-computing pixels' number to the number of reflective pixels. If the ratio is beyond our pre-defined threshold, we update the reference image with new reflection buffers. Otherwise we use the reference image to predict the reflection rendering. In this way we can balance the workload and guarantee the reflection quality at the same time. Alternative strategy like Nehab et al. [20] could be used to update the pixel by updating pixels in tiled regions periodically. However, the nonuniformity of reflection quality between updated and un-updated pixels becomes obvious because of reflection's high-frequency signal property (i.e., reflection changes rapidly in space). Our method is opted for reflection projection as it is simple to implement, adjustable, predictable and stable. The decision on better selecting the pre-defined threshold value depends on several configurations including the scene, the light, the amount of movement and the desired reflection quality.

4.4 Performance

Table 1 gives the performance of our method and the comparison to ray tracing. Performance was recorded by render-

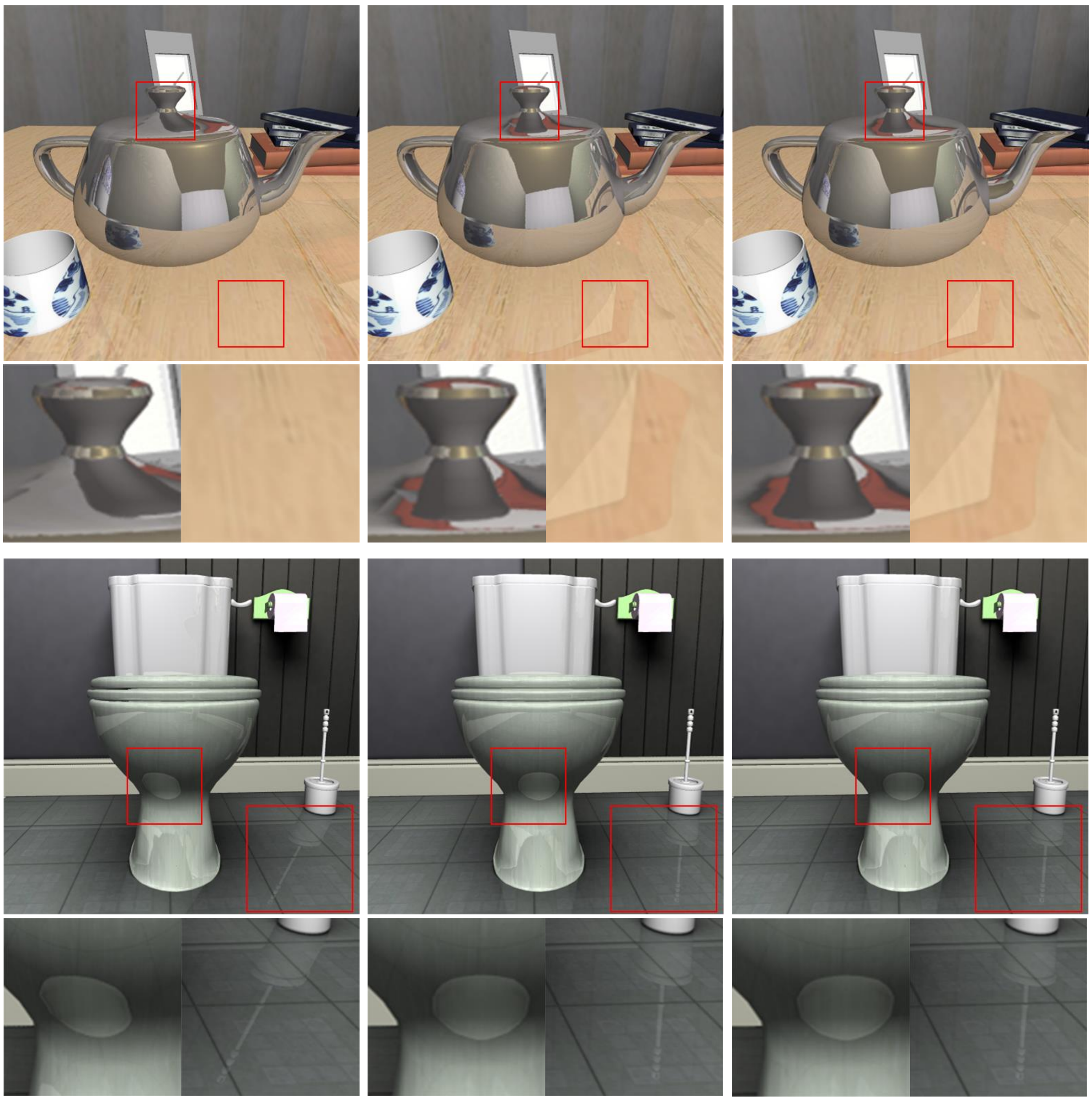

Fig. 6: Reflections rendering results from *Living Room* and *Toilet* with close-up views. Naïve approach (left) suffers from artifacts compared to the ground truth (right). Our method (middle) can produce specular reflection images of high quality.

ing two 500-frame sequences along two individual motion paths in the *Living Room* and *Toilet*. We use 0.3 as the re-computed pixels' pre-defined threshold value. That is, if the re-computed pixels' number is above 30 percent of whole reflective pixels' number, we refresh the reference image with new reflection buffers. Otherwise, the reference image is used to generate reflection of the following frames. The last column of Table 1 *Re-comp. pixels* defines the ratio of pixels in *Specular pixels* where TCRR is invalid to the number of reflective pixels. *Key frame intervals* illustrates the frequency of reference images during the frames sequence.

Our method sustains 1.8 speed up factor over ray-tracing for the simple *Toilet* scene and an average speedup factor of 3.7 could be achieved for the more complex *Living Room*. The reason why the minimal performance of our method is comparable to ray-tracing is that these performances are recorded when the ray-tracing technique is used to generate the reflections in the key frames. In other words, we use the similar technique with ray-tracing to generate the reference images in the key frames. The difference is that our method stores the reflected information in the reflection buffers while ray-tracing does not. We breakdown the ap-

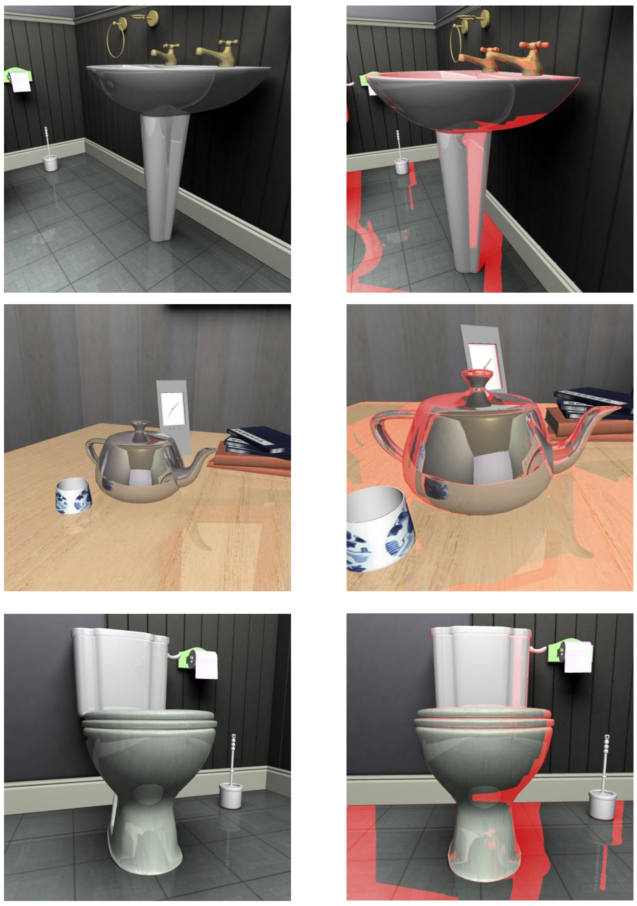

Fig. 7: Reference images (left) and the newly generated images (right). The new images are rendered by projecting the reference images with reflection buffers into the new view. The area that needs re-evaluation is marked in red. The new rendering areas stay consistent with the reflection projection areas and we save time by reducing the total reflection workloads.

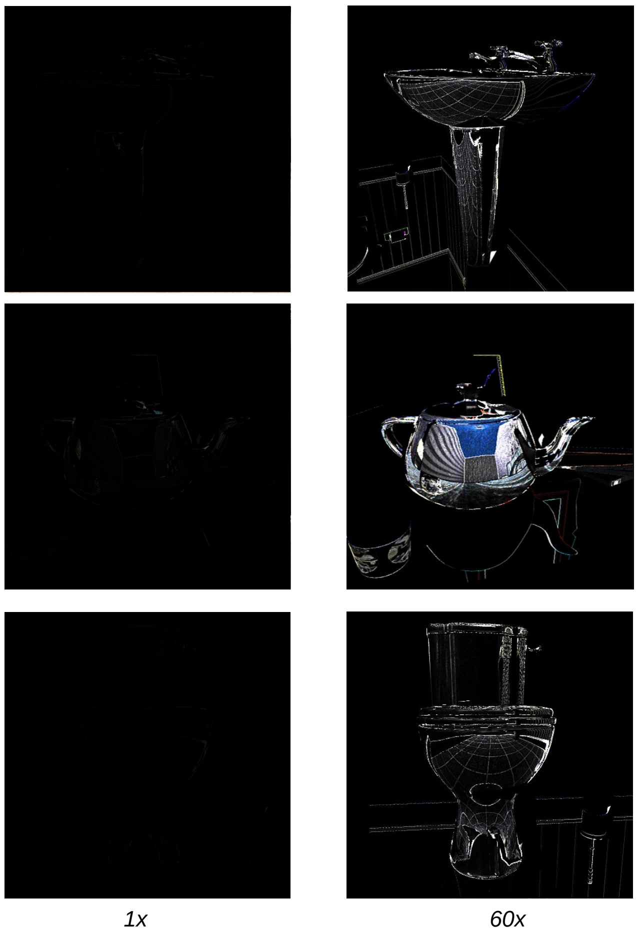

Fig. 8: Left: The difference images between our method and ray-tracing for images in Figure 1 and Figure 6. Right: Same images with difference intensities scaled up by a factor of 60.

proach analysis into individual steps to better explain our method in the Table 2.

The average times in milliseconds for the main steps of our algorithm is given in Table 2. The core of our method, TCRR method, takes a small portion of total rendering time. The reflection re-computation step is quite expensive and accounts for more than half of rendering time. The reason is that although we reduce the ray-tracing workloads by 4/5, the remaining reflected-rays have a higher degree of fragmentation and therefore lose the benefits brought by rays' coherence which is exploited in Optix engine. A constant software overhead per launch and not generating enough load on the GPU to keep all streaming multiprocessors busy also hinder reflection re-computation steps archiving high rendering performance.

5 Discussion and Conclusion

We have proposed a new approach to accelerate the time-consuming reflection rendering computation by exploiting the correlation of reflections between two individual frames. We introduce a general projection method to embrace the reflection projecting from the reference images into the current view by a three-step procedure. First, an offset vector buffer is stored by selecting the best matched pixel with respect to the current camera for each pixel. Second, the reflection matching image is rendered by shifting each reflective pixel with the vector in the formerly generated buffer. Finally a common warping operation is used to warp the reflection matching image into the current view. By using a mesh-based pixel mapping technique, it can be easily decided where the reflection color in the reference image can be projected or the value has to be re-computed.

Table 1: Performance of our method compared to Optix

scene	Specular pixels[%]		Optix [fps]			Our method [fps]			Key-frame intervals		Re-comp. pixels[%]
	max	avg	max	min	avg	max	min	avg	max	avg	avg
Toilet	55	45	23.0	13.4	18.5	54.9	13.0	33.4	51	20.6	20.8
	58	44	22.4	14.0	18.0	49.5	12.6	34.9	63	20.0	18.9
Liv.Room	62	50	18.6	7.5	9.4	50.8	7.0	34.7	164	53.1	18.7
	58	44	12.8	6.4	8.8	48.4	6.6	32.0	92	24.2	18.5

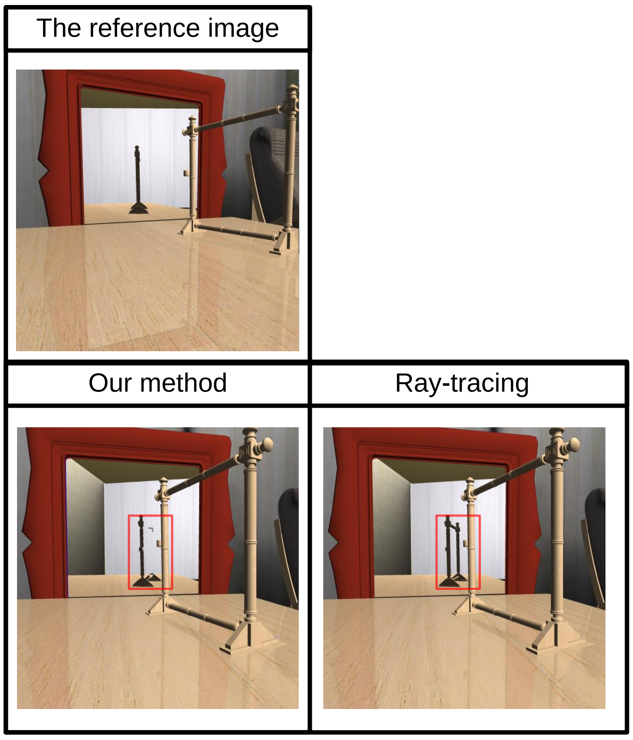

Fig. 9: Disocclusion errors are introduced by the reflected rays that miss the back pillar but hit the wall.

Table 2: Performance in Milliseconds for Various Algorithm Steps

Stage	Step	Toilet	Living Room
		time (ms)	
Diffuse component shading		0.6 (2.0%)	0.7 (2.2%)
TCRR	Pixel searching	3.4 (11.6%)	2.2 (7.1%)
	Reflection shifting	1.3 (4.4%)	1.7 (5.5%)
	Reflection warping	2.7 (9.2%)	2.1 (6.9%)
Final composing	Reflection re-comp.	20.3 (69.2%)	23.2 (75.8%)
	Blend shading	0.9 (3.1%)	0.7 (2.2%)
Total		29.3	30.5

Our method only depends on camera movement, and supports concave and convex reflectors. It is conceptually simple and can easily be integrated in most reflection ray-tracing techniques where ray-tracing is a bottleneck of the rendering system, and can be used for reflection rendering for a variety of concave, convex and planar reflectors. It can produces reflection image with no obvious error while greatly reduce the workloads of tracing reflecting rays by projecting the reflections in reflection buffers. Thus, our method can be used to generate a preview sequence in a scene with reflective objects by given a few key-frames, or be integrated in a client device to synthesis reflections from streaming images from a ray-tracer server.

The workload reduction and performance speed up of our method comes at a cost of reflection rendering quality artifacts. The artifacts come through 1. the assumption that distance function varies monotonously over reflective surfaces and finally converges at the best matched pixel, 2. insufficient geometry sampling in reference buffers, and 3. the disocclusion error due to motion parallax of reflected rays. One limitation of our method is it cannot predict reflections in scenes with moving objects. Our approach can efficiently reconstruct the reflected objects by assuming the geometry's position is static, and use this static reflected position to search optimal reflection point in image space. We can still use our method as long as the movement of each reflected object is provided. Another limitation is the width of efficient reflection projection's time window. Reflected rays vary dramatically with respect to camera movement, the reflected values of a reference image will thus have little correlation with the new image rendered by a distant camera. Our method can handle this situation by assigning an invalid value to these non reflection temporal coherent relevant areas and our method degrades into a pure ray-tracing method.

Our method can handle multi-reflections projection by extending reflection buffers further to represent the geometry surfaces intersected by higher-order rays, and by applying multi TCRR passes recursively. However, multi-reflections are usually unnoticeable; therefore, our method focuses on the approximating reflections by one-order reflected rays and we ignore high-order reflections because they do not warrant the additional cost.

Our method demonstrates that projection techniques using temporal coherence can generate not only view independent components, but also the view dependent components. One direction of our future work is to extent our work to other view-dependent components like refraction and glossy reflections. Another direction of our future work involves detecting the fidelity of the projected reflections more precisely for eliminating the possible gaps between the true reflections to the false predictions. Light weight ways of reducing these artifacts such as a blurring scheme could be used to implement a smooth transition from the reflection re-compute area to the low-fidelity reflection approximate area.

References

1. Edward Angel. Interactive computer graphics. *Image*, 1:2, 2007.
2. Nathan A Carr, Jared Hoberock, Keenan Crane, and John C Hart. Fast gpu ray tracing of dynamic meshes using geometry images. In *Proceedings of Graphics Interface 2006*, pages 203–209. Canadian Information Processing Society, 2006.
3. Daniel Valente de Macedo and Maria Andréia Formico Rodrigues. Real-time dynamic reflections for realistic rendering of 3d scenes. *The Visual Computer*, pages 1–10, 2016.
4. Pau Estalella, Ignacio Martin, George Drettakis, and Dani Tost. A gpu-driven algorithm for accurate interactive reflections on curved objects. In *Eurographics Symposium on Rendering*, page 7. ACM, 2006.
5. Tim Foley and Jeremy Sugerman. Kd-tree acceleration structures for a gpu raytracer. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 15–22. ACM, 2005.
6. Per Ganestam and Michael Doggett. Real-time multiply recursive reflections and refractions using hybrid rendering. *The Visual Computer*, 31(10):1395–1403, 2015.
7. Andrew S Glassner. *An introduction to ray tracing*. Elsevier, 1989.
8. Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM, 1996.
9. Paul Green, Jan Kautz, and Frédo Durand. Efficient reflectance and visibility approximations for environment map rendering. In *Computer Graphics Forum*, volume 26, pages 495–502. Wiley Online Library, 2007.
10. Paul Green, Jan Kautz, and Frédo Durand. Efficient reflectance and visibility approximations for environment map rendering. In *Computer Graphics Forum*, volume 26, pages 495–502. Wiley Online Library, 2007.
11. Wolfgang Heidrich, Hendrik Lensch, Michael F Cohen, and Hans-Peter Seidel. Light field techniques for reflections and refractions. In *Proceedings of the 10th Eurographics conference on Rendering*, pages 187–196. Eurographics Association, 1999.
12. Jan Kautz and Michael D McCool. Approximation of glossy reflection with prefiltered environment maps. In *Graphics Interface*, volume 2000, pages 119–126, 2000.
13. Jan Kautz and Michael D McCool. Approximation of glossy reflection with prefiltered environment maps. In *Graphics Interface*, volume 2000, pages 119–126, 2000.
14. Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaakko Lehtinen, and Timo Aila. Incremental instant radiosity for real-time indirect illumination. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 277–286. Eurographics Association, 2007.
15. Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42. ACM, 1996.
16. Shengying Li, Zhe Fan, Xiaotian Yin, Klaus Mueller, Arie E Kaufman, and Xianfeng Gu. Real-time reflection using ray tracing with geometry field. *Eurographics 06, Short papers*, pages 29–32, 2006.
17. Gerrit Lochmann, Bernhard Reinert, Tobias Ritschel, Stefan Mller, and Hans-Peter Seidel. Real-time Reflective and Refractive Novel-view Synthesis. pages 9–16, Darmstadt, Germany, 2014. Eurographics Association.
18. Oliver Mattausch, Daniel Scherzer, and Michael Wimmer. High-quality screen-space ambient occlusion using temporal coherence. In *Computer Graphics Forum*, volume 29, pages 2492–2503. Wiley Online Library, 2010.
19. Morgan McGuire and Michael Mara. Efficient GPU screen-space ray tracing. *Journal of Computer Graphics Techniques (JCGT)*, 3(4):73–85, December 2014.
20. Diego Nehab, Pedro V Sander, Jason Lawrence, Natalya Tatarchuk, and John R Isidoro. Accelerating real-time shading with reverse reprojection caching. In *Graphics hardware*, volume 41, pages 61–62, 2007.
21. Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. In *ACM SIGGRAPH 2005 Courses*, page 173. ACM, 2005.
22. Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. Optix: a general purpose ray tracing engine. *ACM Transactions on Graphics (TOG)*, 29(4):66, 2010.
23. Voicu Popescu, Chunhui Mei, Jordan Dauble, and Elisha Sacks. Reflected-scene impostors for realistic reflections at interactive rates. In *Computer Graphics Forum*, volume 25, pages 313–322. Wiley Online Library, 2006.
24. Voicu Popescu, Elisha Sacks, and Chunhui Mei. Sample-based cameras for feed forward reflection rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 12(6):1590–1600, 2006.
25. Timothy J Purcell, Ian Buck, William R Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 703–712. ACM, 2002.
26. Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 45–50. Eurographics Association, 2007.
27. Daniel Scherzer, Michael Schwärzler, Oliver Mattausch, and Michael Wimmer. Real-time soft shadows using temporal coherence. In *Advances in Visual Computing*, pages 13–24. Springer, 2009.
28. Michael Schwärzler, Christian Luksch, Daniel Scherzer, and Michael Wimmer. Fast percentage closer soft shadows using temporal coherence. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 79–86. ACM, 2013.
29. Pitchaya Sitthi-amorn, Jason Lawrence, Lei Yang, Pedro V Sander, and Diego Nehab. An improved shading cache for modern gpus. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 95–101. Eurographics Association, 2008.
30. Pitchaya Sitthi-amorn, Jason Lawrence, Lei Yang, Pedro V Sander, Diego Nehab, and Jiahe Xi. Automated reprojection-based pixel shader optimization. In *ACM Transactions on Graphics (TOG)*, volume 27, page 127. ACM, 2008.
31. László Szirmay-Kalos, Barnabás Aszódi, István Lazányi, and Mátyás Premecz. Approximate ray-tracing on the gpu with distance impostors. In *Computer graphics forum*, volume 24, pages 695–704. Wiley Online Library, 2005.
32. Yuichi Taguchi, Amit Agrawal, Srikumar Ramalingam, and Ashok Veeraraghavan. Axial light field for curved mirrors: Reflect your perspective, widen your view. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 499–506. IEEE, 2010.

33. Kostas Vardis, Andreas A Vasilakis, and Georgios Papaioannou. A multiview and multilayer approach for interactive ray tracing. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 171–178. ACM, 2016.
34. Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. In *Computer graphics forum*, volume 20, pages 153–165. Wiley Online Library, 2001.
35. Lili Wang, Naiwen Xie, Wei Ke, and Voicu Popescu. Second-order feed-forward renderingfor specular and glossy reflections. *Visualization and Computer Graphics, IEEE Transactions on*, 20(9):1316–1329, 2014.
36. Turner Whitted. An improved illumination model for shaded display. In *ACM SIGGRAPH Computer Graphics*, volume 13, page 14. ACM, 1979.
37. Sung-Eui Yoon, Christian Lauterbach, and Dinesh Manocha. R-lods: fast lod-based ray tracing of massive models. *The Visual Computer*, 22(9-11):772–784, 2006.
38. Jingyi Yu, Jason Yang, and Leonard McMillan. Real-time reflection mapping with parallax. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 133–138. ACM, 2005.
39. Xuan Yu, Rui Wang, and Jingyi Yu. Interactive glossy reflections using gpu-based ray tracing with adaptive lod. In *Computer Graphics Forum*, volume 27, pages 1987–1996. Wiley Online Library, 2008.