# Accelerated Stereo Rendering with Hybrid Reprojection-Based Rasterization and Adaptive Ray-Tracing

Niko Wißmann *†
TH Köln, Computer Graphics Group

Martin Mišiak *‡
TH Köln, Computer Graphics Group
University Würzburg, HCI Group

Arnulph Fuhrmann§
TH Köln, Computer Graphics Group

Marc Erich Latoschik¶
University Würzburg, HCI Group

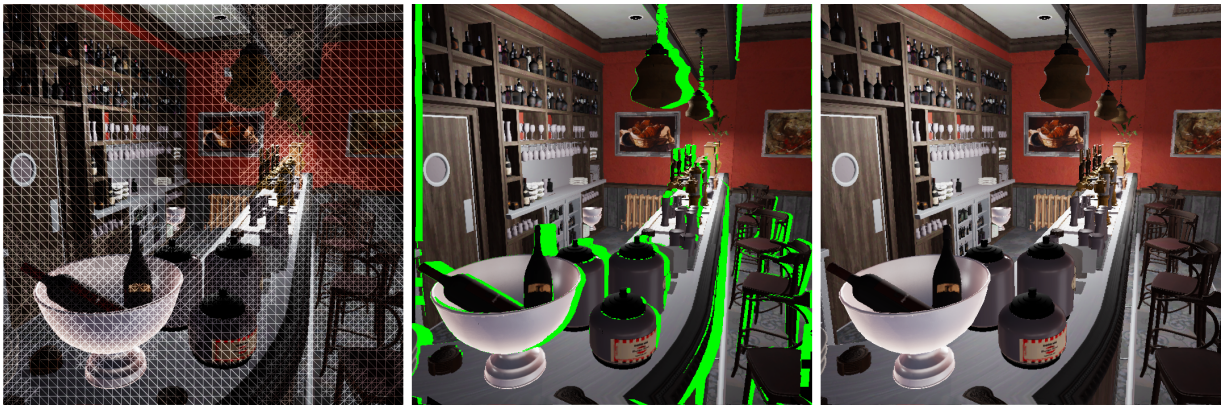


Figure 1: The proposed hybrid rendering system reprojects the source image of the left viewpoint into the right viewpoint by using an adaptive 3D grid warping approach (left), detects disoccluded regions (middle, marked in green) and resolves them correctly through adaptive real-time ray-tracing (right).

## Abstract

Stereoscopic rendering is a prominent feature of virtual reality applications to generate depth cues and to provide depth perception in the virtual world. However, straight-forward stereo rendering methods usually are expensive since they render the scene from two eye-points which in general doubles the frame times. This is particularly problematic since virtual reality sets high requirements for real-time capabilities and image resolution. Hence, this paper presents a hybrid rendering system that combines classic rasterization and real-time ray-tracing to accelerate stereoscopic rendering. The system reprojects the pre-rendered left half of the stereo image pair into the right perspective using a forward grid warping technique and identifies resulting reprojection errors, which are then efficiently resolved by adaptive real-time ray-tracing. A final analysis shows that the system achieves a significant performance gain, has a negligible quality impact, and is suitable even for higher rendering resolutions.

**Index Terms:** Computing methodologies—Computer Graphics—Rendering—Ray tracing; Computing methodologies—Computer Graphics—Rendering—Rasterization; Computing methodologies—Computer Graphics—Graphics systems and interfaces—Virtual reality

*Authors contributed equally to this work.
†e-mail: niko.wissmann@gmail.com
‡e-mail: martin.misiak@th-koeln.de
§e-mail: arnulph.fuhrmann@th-koeln.de
¶e-mail: marc.latoschik@uni-wuerzburg.de

## 1 Introduction

As recent years have shown, the Virtual Reality (VR) community continues to grow steadily. More and more hardware manufacturers are launching new head-mounted displays (HMDs) on the market. This includes mobile systems such as the Oculus Go or the more advanced Oculus Quest. But also classic desktop systems like the HTC Vive Pro will remain on the market. Nevertheless, all systems have one characteristic in common - they require stereoscopic rendering with a high spatial and temporal resolution. Despite continuous improvements in the underlying GPU hardware, the increasing rendering requirements continue to pose a challenge for the latest VR devices. Not only increasing resolutions and refresh rates but also increasing demands on visualization fidelity can have a considerable impact on the rendering performance. A high degree of immersion is essential for every VR application. A low frame rate or a high system latency can have a negative impact on it and can lead to a reduced *sense of presence* [29], or even cause motion sickness. That is why the underlying rendering of a VR application has to meet the given requirements.

Optimization methods for stereoscopic rendering, frequently used in popular game engines, often target only at the CPU-side application loop or improve geometry processing to a certain extent. For example, single-pass stereo rendering using geometry duplication in a geometry shader or using instanced API drawcalls. But these optimization options do not reduce the workload in the fragment shaders. Still, all pixels of both output frames of the stereo image pair have to be shaded entirely in the fragment shaders.

This paper presents a rendering system that accelerates the rendering process of one image of the stereo pair using spatial reprojection. The reprojection technique is integrated into a classic deferred rendering pipeline, which generates the left reference image via regular rasterization and fragment shading. This is followed by a forward reprojection step using an adaptive 3D grid warping ap-

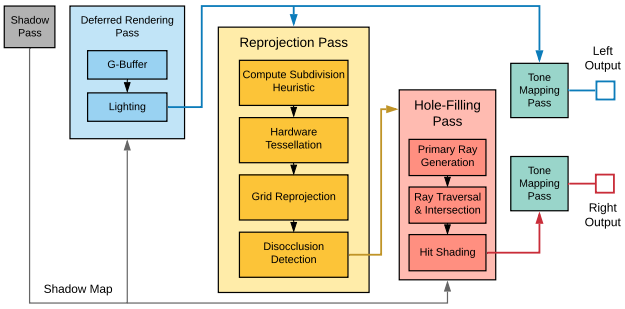



Figure 2: Overview of the hybrid rendering system, consisting of a standard deferred rasterization pass for the left viewpoint and a subsequent reprojection pass with ray-traced hole-filling for the right viewpoint.

proach [16, 34] to reproject the pixel information of the left image into the right viewpoint. Reprojection errors and appearing visual artifacts are correctly solved via an adaptive real-time ray-tracing hole-filling technique. To this end, the DirectX Raytracing (DXR) API is used to perform hardware-accelerated ray-tracing on supported hardware, and hence eliminating the typical bottleneck that arises from performing compute-intensive ray-tracing operations in parallel to rasterization. The resulting image (cf. Fig. 1) correctly captures disocclusions and is very close to a reference rendering.

In our paper we make the following contributions:

- A hybrid reprojection system that combines rasterized rendering with adaptive ray-tracing to accelerate stereoscopic rendering
- A perceptually motivated heuristic for an adaptive grid subdivision used during the 3D grid warping stage
- A comprehensive performance analysis of said system, including a comparison between hardware-accelerated ray-tracing and rasterization for the shading of disoccluded regions

## 2 Related Work

Increasing display resolutions and higher frame rates do not necessarily imply an equal increase in the required computational effort to render a frame. Reprojection algorithms leverage existing spatial and temporal coherences within a scene to reuse previously computed values. A large body of work exists for these algorithms extending over various different fields such as view interpolations of video footage, acceleration of preview image rendering for offline renderers, up to performance optimization for real-time rendering. Motivated by our use case of stereoscopic rendering, we focus primarily on reviewing reprojection algorithms aimed at real-time rendering.

### 2.1 Reprojection

Graphics rendering was accelerated via reprojection very early [1–3, 19]. Reprojection could shorten the computation times of first ray-tracing-based renderings of stereoscopic images by exploiting the coherence between the two perspectives. The first usage of reprojection for a head-tracked stereoscopic display is given by McMillan et al. [28].

Many approaches rely on forward reprojection, where each pixel of the input image is mapped to one pixel in the output image [39]. Since the mapping is surjective, a direct reprojection does not assign a value to all pixels of the output image, resulting in visible holes. This fact can be mitigated by using point splatting [33, 38], or by interpreting the input image as a connected mesh [26] whose vertices mirror the depth values of the pixels. Such a *warping grid* can then be transformed via a vertex shader into another perspective [15], where rasterization and shading take place. This approach does not suffer from the aforementioned pixel holes. Instead, foreground and background objects are connected by so-called "rubber-sheets".

A native resolution warping grid (one-to-one mapping between vertex and pixel) is not necessary, as adjacent pixels with similar attributes can be warped together as bigger patches into the target perspective [12]. Didyk et al. [16] subdivide only grid cells, which cover pixels of a higher depth disparity. Using this adaptive subdivision approach results in less grid triangles and hence in better warping performance without compromising quality. Schollmeyer et al. [34] subdivide the warping grid based on a colinearity measure, which prevents over-tessellation of slanted surfaces and reduces the number of grid triangles even further. In addition the authors also handle the reprojection of transparent geometry via ray-casting into an A-Buffer.

A reprojection can also be done in *backward* order, where each pixel in the target image searches for corresponding pixels in one or more source images. Nehab et al. [30] presented the *reverse reprojection cache*, which allowed fragments to reuse shading results from previous frames. This is achieved by explicitly storing and transforming vertex attributes for multiple frames. Another approach is proposed by Bowles et al. [11], which extends the work of Yang et al. [41] on fixed point iteration to search for a given pixels location in the previous frame. The advantage of the method is that no additional vertex attribute is needed for the position in the previous frame, and also a second transform of the vertex into the previous position is not required. A more recent usage of fixed point iteration is proposed by Lee et al. [24], but only in the context of depth buffer warping to accelerate, e.g., occlusion culling, and not for the rendering of color images.

### 2.2 Hole-Filling

The common problem of all mentioned reprojection methods is that they cannot render a perfect image for a new camera perspective. If previously hidden geometry becomes visible in the new perspective, *disocclusions* occur, as no color information is available in the source image. These pixel regions cannot be filled properly by reprojection alone, and erroneous pixels, also called *holes*, remain in the image. A large body of work has been done in the field of digital image and video post-processing on hole-filling, or inpainting, techniques [10, 14, 40, 42]. However due to performance considerations, these are rarely used in a real-time context. Instead authors rely on simpler techniques, as they are required to execute in a fraction of the available frame-time budget.

The correct, but also the most expensive solution is to rerender the missing information. While the shading calculations can be restricted only to specific pixels [30], in raster-based rendering the entire captured scene geometry has to be processed nonetheless, introducing a significant overhead. Didyk et al. [16] use a simple inpainting strategy, which selects random neighbouring pixels to cover up resulting holes. While very efficient, this method is not suitable for prominent disoccluded regions. Bowles et al. [11] clone the pixel values of the background texture surrounding the disoccluded region. Schollmeyer et al. [34] employ a hole-filling solution using low-pass filtered intermediate result images. After the image warping process, the output image is subject to a multi-stage low-pass filtering process, where only non-hole pixels, or pixels that have a greater depth than the previously calculated average are considered. This approach works well for small disocclusions or to cover up largely homogeneous regions. However for more detailed background textures, the inpainting becomes obvious.

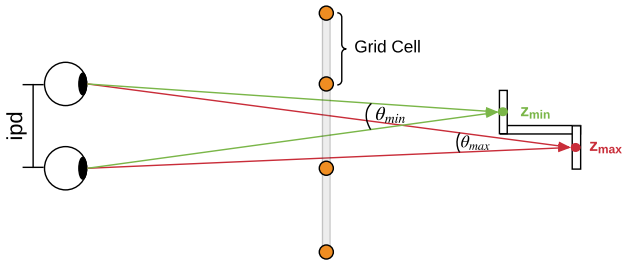



Figure 3: Grid subdivision heuristic using binocular parallax. Based on the depth values inside of a grid cell, the maximum and minimum binocular parallax angles, $\theta_{max}$ and $\theta_{min}$, are determined. If their difference is above a certain threshold, the cell needs to be subdivided.

### 2.3 Ray-Tracing

*Adaptive ray-tracing* has also been used very early as a hole-filling technique. For example, Adelson et al. [4] employ it to accelerate the generation of fully ray-traced animation frames. Previously rendered frames are reprojected into the current view perspective and validated for resulting errors which are marked for the subsequent adaptive ray-tracing hole-filling pass. Only a small number of pixels have to be reshaded from scratch with newly traced primary rays. Unlike the previously mentioned inpainting and filtering strategies, this technique leads to exact ray-traced animation frames with reported time savings of up to 92 %, compared to a full rendering. Further usage can be found in a voxel-based terrain rendering system for a flight simulator [32] and for stereoscopic voxel-based terrain rendering [36]. It has also been used in the context of foveated rendering to create a higher sampling density in the user's central vision and to correct reprojection errors [37]. However, it is only usable in certain scenes designed for ray-tracing, as pointed out by Friston et al. [20].

Since the introduction of hardware accelerated ray-tracing on consumer graphics cards, this new feature was primarily used in *hybrid rendering* systems for improving the fidelity of rendering effects like reflections, global illumination (GI) and shadows for a wide range of 3D scenes [21]. The work of Barré-Brisebois et al. [8] should be mentioned here on the hybrid rendering pipeline for the *PICA PICA* experimental showcase. It combines standard rasterization with ray-traced visual features in real-time. Another usage can be found in the work of Majercik et al. [25] for efficient GI rendering by extending irridiance probes. It is also used in the field of anti-aliasing (AA) as shown by Marrs et al. [27] to improve Temporal AA [23].

In contrast, our work leverages hardware accelerated ray-tracing to improve the rendering performance of stereoscopic applications. We build upon the work of both Didyk et al. [16] and Schollmeyer et al. [34] who also target stereoscopic rendering using a grid based reprojection approach. However, our approach differs in a few important ways. Our grid subdivision is done in a single pass using the hardware tessallator, instead of a multi-pass geometry shader approach. We employ a perceptually motivated heuristic to subdivide our grid cells, which is not based on absolute measures. And lastly, we do not perform any inpainting to hide disocclusion artifacts. Instead, we use adaptive raytracing to correctly reconstruct the missing information.

## 3 Hybrid Stereo Rendering

The reprojection system proposed in this paper consists of two main components - a reprojection pass, and a hole-filling pass. An overview of the system is shown in Fig. 2. For our integration, we use a standard deferred rendering pipeline, which is divided into a G-buffer pass and a lighting pass. The left image of the stereo pair is generated as usual by the deferred pass. Shadow mapping is integrated by generating a simple shadow map for the primary light source of the scene and forwarding it to the lighting pass. The output frame of the lighting pass is directly submitted to the left portion of the output display, but also passed on, including the depth buffer of the left image, to the following reprojection pass.

The reprojection pass performs a forward 3D grid warping and reprojects the left source image into the right viewpoint. Any resulting reprojection errors (holes) are detected and filled via the subsequent hole-filling pass using adaptive real-time ray-tracing. The ray-tracing pass executes the same shading code (including shadow factor calculation) in the *ray-trace hit* shaders as the fragment shader of the lighting pass. Thus, the holes are filled with correctly shaded color information, and the resulting image is then submitted to the right portion of the display.

It is important to note that our reprojection system does not depend on any specific rendering pipeline. It requires only the depth and radiance values of a rendered viewport, and can therefore be integrated into a large variety of rendering setups. Also the reprojection direction is arbitrary and not limited to any specific direction. However for the sake of simplicity, we will assume a left-to-right reprojection in our explanation.

### 3.1 3D Grid Warping

The warping grid is initialized at the system's start in NDC coordinates with a cell size of $16 \times 16$ pixels and bound to the GPU. As already pointed out by Didyk et al. [16], 3D warping does not require a native grid resolution. However, a $16 \times 16$ cell is not sufficient enough to warp geometry surfaces distortion-free at borders of foreground and background objects. For this reason, a dynamic hardware tessellation is used to adapt the grid cell size to the underlying source image. The tessellation is done via the hardware tessellator stage introduced in DirectX 11, which can be controlled by specific hull and domain shaders. In order to decide dynamically whether a grid cell should be tessellated or not, a compute shader is executed before the actual reprojection takes place. In it, a fast parallel reduction is performed on the depth buffer of the source image to obtain min/max depth values residing inside a grid cell. Based on them, we compute our subdivision heuristic and pass it to the next stage. In the hull shader, a threshold condition is used to decide whether the given grid cell should be tessellated with the factor of 16 (native resolution equivalent) or left unchanged.

A common source of visual artifacts when using adaptive tessellation are T-junctions. They arise when neighbouring patches use different tessellation factors [31], and can introduce visible cracks into a displaced mesh. In order to prevent these artifacts from happening, we additionally tessellate all grid cells in a 4-neighborhood around the cell in question. While this increases the amount of generated grid triangles, it also prohibits the creation of T-junctions in the direct vicinity of displaced vertices.

The actual reprojection then takes place in the domain shader. Here, all default vertices and new vertices created by the tesellator are transformed into the target viewpoint with a pre-computed reprojection matrix and the depth value of each vertex position in the source image. The reprojection matrix consists of the inverse view-projection matrix of the left camera and the view-projection matrix of the right camera. The grid is then rasterized and shaded via a bilinear texture lookup into the source image.

### 3.2 Subdivision Heuristic

In order to determine if a grid cell of $16 \times 16$ pixels needs to be further subdivided, we employ a perceptually motivated heuristic based on binocular parallax (cf. Fig. 3). Binocular parallax is the angle $\theta$ between the lines of sight of both eyes, separated by an

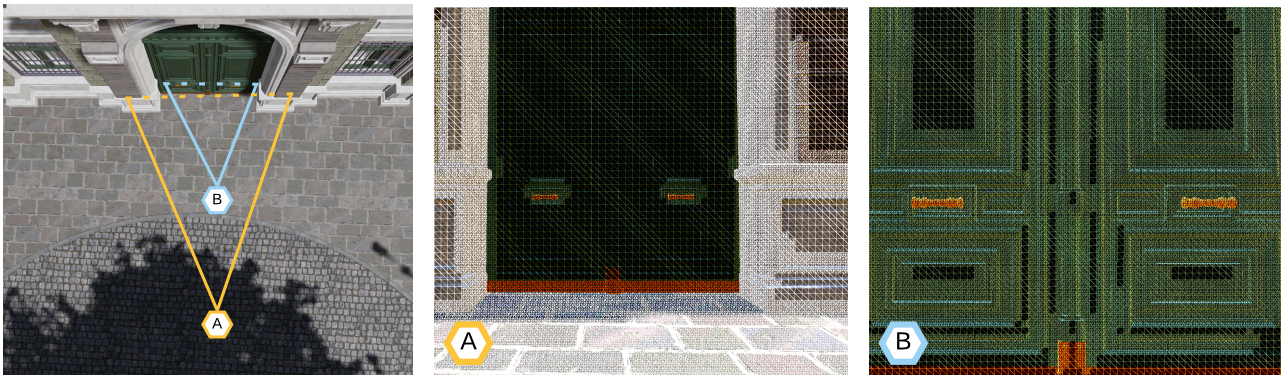



Figure 4: Wireframe rendering of the tessellated warping grid (coarsely subdivided for illustrative purposes). Only grid cells whose depth values exceed a certain binocular parallax difference are further subdivided. This allows for an adaptive subdivision based on the distance to the viewer (see the subdivision on the door for exemplary viewer positions A and B).

interpupillary distance $ipd$, when converged on a point at distance $z$.

$$\theta(z) = 2\tan^{-1}\left(\frac{ipd}{2z}\right) \tag{1}$$

When comparing two points at a depth discontinuity, two distinct binocular parallax angles will be present. The difference between these angles $\theta_\Delta$, is what determines if an observer can differentiate between the two depth values (based on stereoscopic vision alone). The discrimination threshold is referred to as *stereo acuity*, and is on average between 0.5 and 1 minutes of arc [13].

In order to provide a conservative estimate for our grid subdivision, we search for the minimum ($z_{min}$) and maximum ($z_{max}$) depth value inside of a given cell. It should be noted, that when working with binocular parallax, depth values have to be linearized first. A difference $\theta_\Delta$ between the associated binocular parallax angles is computed

$$\theta_\Delta(z_{min}, z_{max}) = \theta(z_{min}) - \theta(z_{max}) \tag{2}$$

and compared against a threshold value. Values under the threshold are indicative of indistinguishable depth differences, for which a subdivision would be redundant. For values over the threshold, on the other hand, a subdivision becomes necessary. In contrast to subdivision approaches based on absolute depth [16] or colinearity differences [34], our heuristic interprets depth disparities in relation to the distance at which they occur (cf. Fig. 4).

### 3.3 Disocclusion Detection

The result of the grid warping stage can be seen in Fig. 6 (a). Resulting "rubber-sheets" can cover disoccluded regions, as well as occlude regions, where correct information is available in the source image.

We discard these regions and flag them (i.e. green regions in the middle of Fig. 1) for a subsequent hole-filling process. To this end, a Sobel operator is used in the domain shader, to filter the input depth buffer at a given vertex position. The resulting per-vertex depth gradient is passed on to the fragment shader, where a threshold condition decides if a fragment should be discarded. We also tested if afflicted grid triangles could be directly discarded via a geometry shader, to avoid unnecessary rasterization. However the presence of the geometry shader became a significant bottleneck in our pipeline, and therefore we resorted to the fragment discarding described above.

### 3.4 Hole-Filling

Within the warping process, the texture target of the frame buffer object (FBO) is cleared as usual for each new frame. This also sets the value of the alpha channel to 0. If a fragment in the fragment shader is successfully shaded via the texture lookup, the alpha value becomes non-zero. When discarding a fragment, on the other hand, the alpha value remains unchanged. Based on this alpha value, the *ray generation* shader of the hole-filling pass can decide whether a ray for a certain pixel should be generated and traced or not.

The hole-filling process needs three different shaders that are predefined by the DXR API. A *ray generation* shader as the initial entry point to start the ray-tracing. A *closest hit* shader used for the actual hit point shading and a *miss* shader, in case no geometry is hit. There was no need for any custom intersection shader, so the standard DXR ray-triangle intersection shader is used together with the provided standard acceleration structure for the ray traversal and intersection stage. If the intersection shader reports a successful triangle hit, the same shading code as in the fragment shader of the lighting pass will be executed. The only difference is the calculation of the used texture MIP-level.

Unlike rasterization, where pixels are shaded within a $2 \times 2$ pixel block and four differentials can be generated to compute an appropriate texture MIP-level, this cannot be done by a single ray. One solution is to use *cone tracing* [6]. Using the projected footprint of the cone, the texture MIP-level can also be calculated for a single hit point of a ray. A practical implementation of this technique is described by Akenine-Möller et al. [5]. The texture MIP-level $\lambda$ is therefore determined by

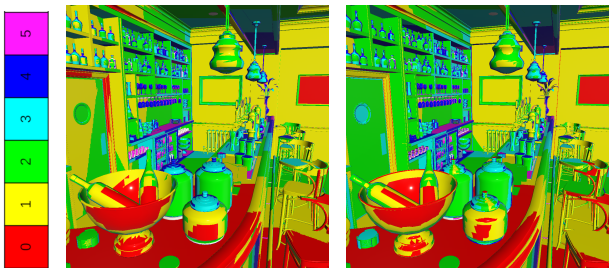
$$\lambda = \Delta_0 + 0.5\log_2\left(\alpha\|\mathbf{d}_{hit}\|\frac{1}{|\hat{\mathbf{n}}\cdot\hat{\mathbf{d}}_{hit}|}\right), \tag{3}$$

where $\alpha$ is the *spread angle*, derived from the vertical field of view and the height of the viewport, $\mathbf{d}_{hit}$ the vector from the camera center to the hit point, and $\mathbf{n}$ is the normal at the hit point. If the distance increases, the footprint size also increases. If the angle between $\mathbf{n}$ and $\mathbf{d}_{hit}$ increases, i.e., a surface is viewed at an oblique angle, the footprint also increases. Together with the base texture MIP-level $\Delta_0$, which according to Ewins et al. [18] can be approximated by

$$\Delta_0 = 0.5\log_2\left(\frac{t_a}{p_a}\right), \tag{4}$$

using the double texel-space area $t_a$ and the double triangle area $p_a$, the correct texture MIP-level can be calculated for each ray hit. This approach allows for a comparable MIP-level selection as with rasterization (cf. Fig. 5).

When using our hole-filling approach, even large disoccluded regions containing previously hidden information, are correctly reconstructed (cf. Fig. 6).




(a) (b) Texture MIP-levels as computed by standard rasterization. (c) Texture MIP-levels as computed by a cone-tracing approximation.

Figure 5: A visualization of texture MIP-levels (a) used by our rasterization (b) and ray-tracing (c) stages.

(a) (b)

Figure 6: Results of our reprojection algorithm. (a) Performing only the grid warping step results in "rubber-sheet" artifacts in areas not visible from the reprojecting viewpoint. (b) These areas are efficiently and correctly resolved via adaptive ray-tracing.

### 3.5 Rasterization-based Hole-Filling

In addition to our hybrid reprojection approach, we wanted to compare the hole-filling performance of dedicated ray-tracing hardware against the GPUs rasterizer. We implemented therefore a second reprojection method. It performs the same base reprojection as the hybrid ray-tracing system, but remaining holes are not filled in by ray-tracing, but via rasterization and subsequent fragment shading (G-buffer and lighting pass). To keep the shading costs minimal, a stencil mask is used which covers only previously discarded pixels.

## 4 Performance Analysis

For our implementation, the open-source real-time research framework *Falcor* [9] (version 3.2.1) was used. Falcor supports the DirectX 12 API together with the DXR API extension for real-time ray-tracing. All shaders were implemented using Falcor's integrated shading language *Slang* [22], a language extension for DirectX HLSL shaders. The performance measurements have been done in the four test scenes shown in Fig. 7 with the corresponding properties listed in Table 1. For our subdivision heuristic, we chose a $\theta_\Delta$ of 1 minute of arc. In each test run, 1000 frames were captured during a fixed camera movement defined by a static camera path through the scenes. All tests were performed on a workstation with a 3.5 GHz AMD Ryzen Threadripper 1920X 12-Core processor with 16 GB RAM and a single NVIDIA GeForce RTX 2080 Ti with 8 GB video memory. All time measurements were done for HD (1920 $\times$ 1080) and 4K (3840 $\times$ 2160) per eye rendering resolutions.

### 4.1 Test Setup

Three rendering systems have been compared within our performance analysis. The naive stereo rendering implementation, using two full render passes, serves as a reference system and is referred to as *Plain Stereo*. The presented reprojection system with hybrid rasterization and adaptive real-time ray-tracing is called *Repro RT*, while the same system but with rasterization-based hole-filling is called *Repro ReRaster*. All three systems use CPU side frustum culling and GPU side backface culling.

### 4.2 Results

The measured frame times for all test scenes and resolutions are shown in Fig. 8. In all instances, the hybrid rendering system Repro RT achieves the best performance. Especially the result charts for 4K resolution show large differences to the Plain Stereo renderer. The Repro ReRaster system is also consistently faster than Plain Stereo, however only at 4K resolution. For HD, it struggles in some scenes to keep up with Plain Stereo, providing no speed up at all. The strong differences in the Repro ReRaster results for HD resolution can be attributed to the single-thread performance of our test system. Repro Reraster has the overhead of issuing all drawcalls twice, in addition to rendering the warping grid. Here, the ray dispatch call of the ray trace system has a clear advantage.

This behavior can also be confirmed by looking at the achieved speed-up factors, which are listed in Table 2. Repro RT achieves the best speedup factors in all test cases. The numbers also clearly show that the achieved speed-ups for both reprojection systems increase with output resolution. This can be explained by the fact, that a higher percentage of screen pixels can be successfully reprojected than it would be the case for a lower resolutions. When put in perspective of constantly increasing display resolutions on new devices, reprojection algorithms will continue to play an important role in the future.

Furthermore, Fig. 9 (left and middle column) shows the GPU times of the individual render stages (exemplary for the Bistro Interior scene), without any CPU overhead. It can be seen that the depth buffer compute stage for the grid tessellation factor (Comp. tess.) contributes only to a very small part of the system's rendering time. The average time for HD is 0.05 ms and for 4K 0.18 ms. It is also noticeable that the grid warping performance is very similar over the whole measured range, although the grid is continuously rendered with a varying number of triangles depending on the camera's position and angle. On average, the grid tessellation and rendering time in all scenes for HD resolution is between 0.2 ms and 0.25 ms. For 4K, it is between 0.73 ms and 0.82 ms. The charts also show that even without the CPU overhead of issuing additional draw calls, Repro ReRaster's hole-filling performance is worse when compared to Repro RT.

### 4.3 Discussion and Limitations

The results show very well that the hybrid rendering system is much faster than the naive reference stereo renderer. On average, the right image is synthesized for HD below 1 ms and for 4K within a maximum of 2 ms. Also, the Repro RT hole-filling technique performs better than Repro ReRaster, especially for higher resolutions. This emphasizes the advantages of individual per-pixel primary rays for the reshading of small screen portions, as opposed to full-screen rasterization.

As can be seen in the right column of Fig. 9, the performance of Repro RT depends mostly on the number of ray-traced pixels. Therefore, in a scenario with massive disocclusions, ray-tracing could lose its advantage over rasterization. However such a scenario is unlikely to be encountered, as our test scenes demonstrate. We explicitly chose the Bistro Interior scene due to its high depth complexity and disocclusion potential. The percentage of ray-traced pixels was on average 6 % with a maximum of 11.5 %.

The tested scenes contained only static geometry, which allowed

(a) Bistro Interior (b) Bistro Exterior (c) Bistro Night (d) Temple

Figure 7: These figures show the used test scenes. Three test scenes were created out of the Amazon Lumberyard Bistro asset [7] and one scene out of the UE4 Sun Temple asset [17].

Table 1: Test scene properties.

| Scene | **Triangles** | **Textures** | **Texture size** | **Lights** |
|---|---|---|---|---|
| Bistro In. | 1,022,521 | 211 | $1 \times 1$ to $2048 \times 2048$ | 1 directional and 13 point |
| Bistro Ex. | 2,832,120 | 405 | $1 \times 1$ to $2048 \times 2048$ | 1 directional |
| Bistro Night | 3,845,871 | 607 | $1 \times 1$ to $2048 \times 2048$ | 1 directional and 31 point |
| Temple | 606,376 | 148 | $512 \times 512$ to $2048 \times 2048$ | 1 directional and 13 point |

Table 2: Total frame time in ms for each rendering system, measured over 1000 frames. The speedup factors are in comparison to the Plain Stereo renderer.

| Reso-lution | Scene | Renderer | **Total time [ms]** | **Speedup factor** |
|---|---|---|---|---|
| HD | Bistro In. | Plain Stereo | 3,074 | |
| | | Repro ReRaster | 2,778 | 1.11 |
| | | Repro RT | 2,404 | 1.28 |
| | Bistro Ex. | Plain Stereo | 5,485 | |
| | | Repro ReRaster | 5,878 | 0.93 |
| | | Repro RT | 3,779 | 1.45 |
| | Bistro Night | Plain Stereo | 7,602 | |
| | | Repro ReRaster | 7,989 | 0.95 |
| | | Repro RT | 5,250 | 1.45 |
| | Temple | Plain Stereo | 3,747 | |
| | | Repro ReRaster | 2,912 | 1.29 |
| | | Repro RT | 2,545 | 1.47 |
| 4K | Bistro In. | Plain Stereo | 11,673 | |
| | | Repro ReRaster | 8,700 | 1.34 |
| | | Repro RT | 8,297 | 1.41 |
| | Bistro Ex. | Plain Stereo | 12,901 | |
| | | Repro ReRaster | 9,719 | 1.33 |
| | | Repro RT | 8,638 | 1.49 |
| | Bistro Night | Plain Stereo | 18,989 | |
| | | Repro ReRaster | 13,771 | 1.38 |
| | | Repro RT | 12,215 | 1.55 |
| | Temple | Plain Stereo | 14,598 | |
| | | Repro ReRaster | 10,392 | 1.40 |
| | | Repro RT | 9,409 | 1.55 |

us to build the acceleration structure in a way to maximize ray traversal performance. However, dynamic scenes are also possible with DXR since the acceleration structure can be updated without a complete rebuild from scratch. This can have, of course, a negative impact on the overall performance of the system. Therefore, the use of our reprojection technique on animated scenes should be further evaluated. Due to the single-pass grid subdivision approach, our warping grid consists of more triangles than a comparable multi-pass approach [16, 34]. For our tested resolutions of HD and 4K, we did not notice this to be any bottleneck in our algorithm, as the increased rendering costs are compensated by the low overhead of the hardware tessellation stage.

As a limitation, it has to be stated that our reprojection system does currently not support warping and hole-filling of semi-transparent geometry. In addition, the reprojection does not produce correct specular reflections as we reproject pixel values under the assumption of a Lambertian surface. To a certain extent, this leads to a reduction in highlight disparity for very sharp or close reflections [35].

## 5 Conclusion and Future Work

This paper describes a hybrid reprojection-based rasterization rendering system with adaptive ray-tracing for accelerating stereoscopic rendering. The speed up is achieved by reprojecting the first half of the stereo image pair into the second view using a warped 3D grid mesh. A binocular parallax heuristic is used to adaptively refine the grid mesh via hardware tessellation to avoid distortions during the warping process, while minimizing the number of subdivisions. Fragments associated with occlusion-based reprojection errors are identified and correctly reshaded via hardware accelerated ray-tracing.

Our performance analysis has shown that the system achieves a significant speed up over a naive stereo rendering, while introducing negligible loss in quality. Due to the use of adaptive ray-tracing, the system does not suffer from typical inpainting problems and scales very well with scene complexity and rendering resolution, even surpassing a rasterization-based hole-filling approach.

For future work, we plan on evaluating the visual quality of our rendering system in a user study, especially in the context of our perceptually motivated grid subdivision heuristic. Furthermore, we would like to exploit the temporal coherence of a scene and improve the system's efficiency by additionally reprojecting from previous frames. Lastly, an efficient handling of semi-transparent geometry and perceptually correct specular reflections, using hardware accelerated ray-tracing, would be another interesting research topic.

## Acknowledgments

We would like to thank the anonymous reviewers for their valuable feedback. This work was partially funded by the German Federal Ministry of Education and Research (BMBF) under grant number 02K16C232 as part of the project *Retail 4.0*.

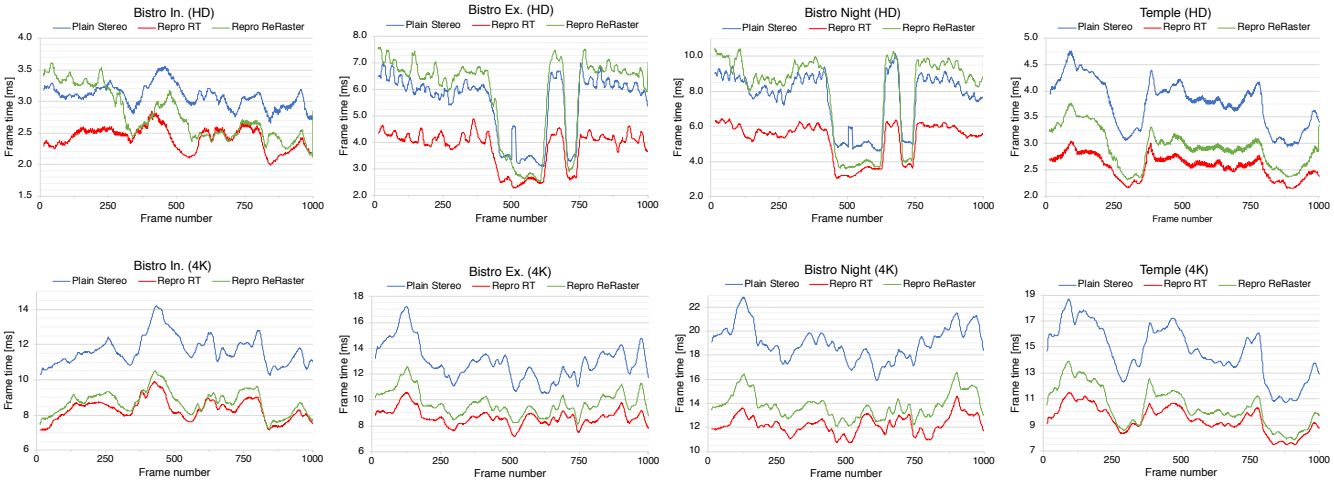



Figure 8: Frame times of all rendering systems for all four test scenes measured for HD and 4K.

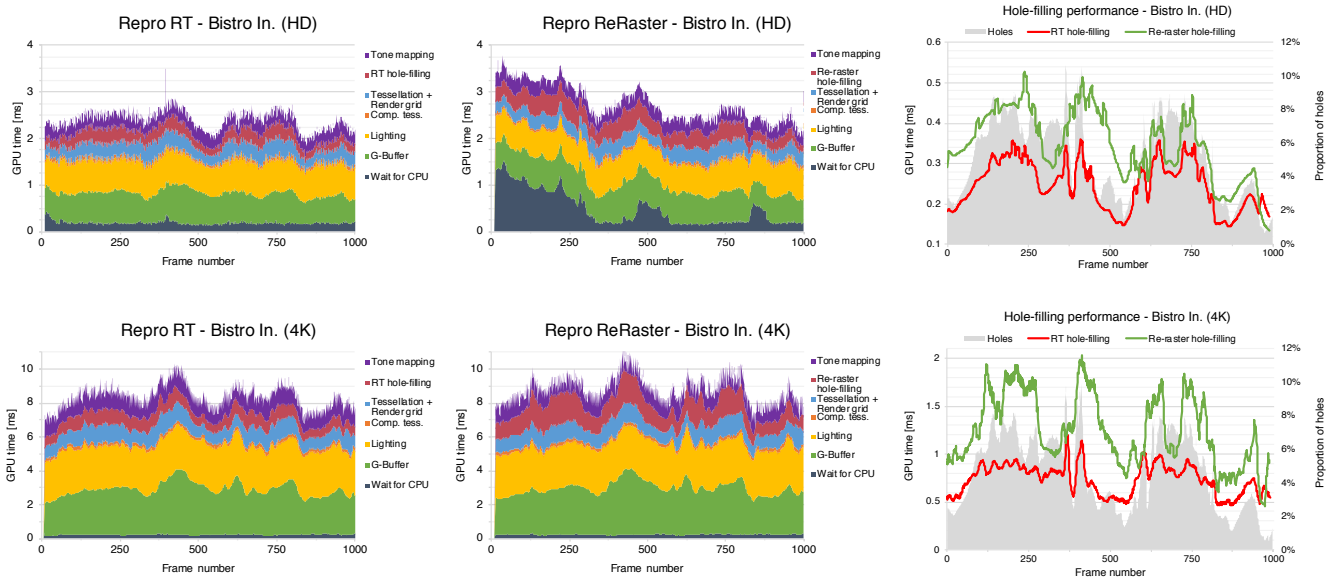



Figure 9: GPU times for individual render stages measured in the Bistro Interior scene (left and middle) and the isolated performance measurements for each hole-filling pass (right).

## REFERENCES


[1] S. J. Adelson, J. B. Bentley, I. S. Chong, L. F. Hodges, and J. Winograd. Simultaneous generation of stereoscopic views. *Computer Graphics Forum*, 10(1):3–10, 1991. doi: 10.1111/1467-8659.1010003

[2] S. J. Adelson and L. F. Hodges. Visible surface ray-tracing of stereoscopic images. In *Proceedings of the 30th Annual Southeast Regional Conference*, pp. 148–156. ACM, 1992. doi: 10.1145/503720.503778

[3] S. J. Adelson and L. F. Hodges. Stereoscopic ray-tracing. *The Visual Computer*, 10(3):127–144, 1993. doi: 10.1007/BF01900903

[4] S. J. Adelson and L. F. Hodges. Generating exact ray-traced animation frames by reprojection. *IEEE Computer Graphics and Applications*, 15(3):43–52, 1995. doi: 10.1109/38.376612

[5] T. Akenine-Möller, J. Nilsson, M. Andersson, C. Barré-Brisebois, R. Toth, and T. Karras. Texture level of detail strategies for real-time ray tracing. In E. Haines and T. Akenine-Möller, eds., *Ray Tracing Gems*, pp. 321–345. Apress, 2019.

[6] J. Amanatides. Ray tracing with cones. *SIGGRAPH Computer Graphics*, 18(3):129–135, 1984. doi: 10.1145/964965.808589

[7] Amazon Lumberyard. Amazon lumberyard bistro, open research content archive (orca), 2017. `http://developer.nvidia.com/orca/amazon-lumberyard-bistro`.

[8] C. Barré-Brisebois, H. Halén, G. Wihlidal, A. Lauritzen, J. Bekkers, T. Stachowiak, and J. Andersson. Hybrid rendering for real-time ray tracing. In E. Haines and T. Akenine-Möller, eds., *Ray Tracing Gems*, pp. 353–370. Apress, 2019.

[9] N. Benty, K.-H. Yao, T. Foley, M. Oakes, C. Lavelle, and C. Wyman. The Falcor rendering framework, 2018. `https://github.com/NVIDIAGameWorks/Falcor`.

[10] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 417–424. ACM, 2000. doi: 10.1145/344779.344972

[11] H. Bowles, K. Mitchell, R. W. Sumner, J. Moore, and M. Gross. Iterative image warping. *Computer Graphics Forum*, 31(2):237–246, 2012.

doi: 10.1111/j.1467-8659.2012.03002.x

[12] S. E. Chen and L. Williams. View interpolation for image synthesis. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, p. 279–288. Association for Computing Machinery, New York, NY, USA, 1993. doi: 10.1145/166117.166153

[13] B. E. Coutant and G. Westheimer. Population distribution of stereoscopic ability. *Ophthalmic and Physiological Optics*, 13(1):3–7, 1993. doi: 10.1111/j.1475-1313.1993.tb00419.x

[14] A. Criminisi, P. Pérez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on image processing*, 13(9):1200–1212, 2004.

[15] P. Didyk, E. Eisemann, T. Ritschel, K. Myszkowski, and H.-P. Seidel. Perceptually-motivated real-time temporal upsampling of 3d content for high-refresh-rate displays. *Computer Graphics Forum*, 29(2):713–722, 2010. doi: 10.1111/j.1467-8659.2009.01641.x

[16] P. Didyk, T. Ritschel, E. Eisemann, K. Myszkowski, and H.-P. Seidel. Adaptive image-space stereo view synthesis. In *Vision, Modeling and Visualization Workshop*, pp. 299–306, 2010.

[17] Epic Games. Unreal engine sun temple, open research content archive (orca), 2017. `https://developer.nvidia.com/ue4-sun-temple`.

[18] J. P. Ewins, M. D. Waller, M. White, and P. F. Lister. Mip-map level selection for texture mapping. *IEEE Transactions on Visualization and Computer Graphics*, 4(4):317–329, 1998. doi: 10.1109/2945.765326

[19] J. D. Ezell and L. F. Hodges. Some preliminary results on using spatial locality to speed up ray tracing of stereoscopic images. In *Stereoscopic Displays and Applications*, vol. 1256, pp. 298–306. SPIE, 1990. doi: 10.1117/12.19912

[20] S. Friston, T. Ritschel, and A. Steed. Perceptual rasterization for head-mounted display image synthesis. *ACM Trans. Graph.*, 38(4):97:1–97:14, 2019. doi: 10.1145/3306346.3323033

[21] E. Haines and T. Akenine-Möller, eds. *Ray Tracing Gems*. Apress, 2019. doi: 10.1007/978-1-4842-4427-2

[22] Y. He, K. Fatahalian, and T. Foley. Slang: Language mechanisms for extensible real-time shading systems. *ACM Trans. Graph.*, 37(4):141:1–141:13, 2018. doi: 10.1145/3197517.3201380

[23] B. Karis. High quality temporal anti-aliasing. *Advances in Real-Time Rendering for Games, SIGGRAPH Courses*, 2014.

[24] S. Lee, Y. Kim, and E. Eisemann. Iterative depth warping. *ACM Trans. Graph.*, 37(5):177:1–177:13, 2018. doi: 10.1145/3190859

[25] Z. Majercik, J.-P. Guertin, D. Nowrouzezahrai, and M. McGuire. Dynamic diffuse global illumination with ray-traced irradiance fields. *Journal of Computer Graphics Techniques (JCGT)*, 8(2):1–30, 2019.

[26] W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3d warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pp. 7–16. ACM, 1997. doi: 10.1145/253284.253292

[27] A. Marrs, J. Spjut, H. Gruen, R. Sathe, and M. McGuire. Adaptive temporal antialiasing. In *Proceedings of the Conference on High-Performance Graphics*, pp. 1:1–1:4. ACM, 2018. doi: 10.1145/3231578.3231579

[28] L. McMillan and G. Bishop. Head-tracked stereoscopic display using image warping. In *Proceedings of SPIE*, vol. 2409, pp. 21–30, 1995. doi: 10.1117/12.205865

[29] M. Meehan, S. Razzaque, M. C. Whitton, and F. P. Brooks. Effect of latency on presence in stressful virtual environments. In *Proceedings of the IEEE Virtual Reality*, pp. 141–148, 2003. doi: 10.1109/VR.2003.1191132

[30] D. Nehab, P. V. Sander, J. Lawrence, N. Tatarchuk, and J. R. Isidoro. Accelerating real-time shading with reverse reprojection caching. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, pp. 25–35, 2007.

[31] M. Nießner, B. Keinert, M. Fisher, M. Stamminger, C. Loop, and H. Schäfer. Real-time rendering techniques with hardware tessellation. *Computer Graphics Forum*, 35(1):113–137, 2016. doi: 10.1111/cgf.12714

[32] H. Qu, M. Wan, J. Qin, and A. Kaufman. Image based rendering with stable frame rates. In *Proceedings of IEEE Visualization 2000*, pp. 251–258, 2000.

[33] S. Rusinkiewicz and M. Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 343–352. ACM Press/Addison-Wesley Publishing Co., 2000.

[34] A. Schollmeyer, S. Schneegans, S. Beck, A. Steed, and B. Froehlich. Efficient hybrid image warping for high frame-rate stereoscopic rendering. *IEEE Transactions on Visualization and Computer Graphics*, 23(4):1332–1341, 2017. doi: 10.1109/TVCG.2017.2657078

[35] R. Toth, J. Hasselgren, and T. Akenine-Möller. Perception of highlight disparity at a distance in consumer head-mounted displays. In *Proceedings of the 7th Conference on High-Performance Graphics*, pp. 61–66. ACM, 2015. doi: 10.1145/2790060.2790062

[36] M. Wan, N. Zhang, A. Kaufman, and H. Qu. Interactive stereoscopic rendering of voxel-based terrain. In *Proceedings of IEEE Virtual Reality 2000*, pp. 197–206, 2000. doi: 10.1109/VR.2000.840499

[37] M. Weier, T. Roth, E. Kruijff, A. Hinkenjann, A. Pérard-Gayot, P. Slusallek, and Y. Li. Foveated real-time ray tracing for head-mounted displays. *Computer Graphics Forum*, 35(7):289–298, 2016. doi: 10.1111/cgf.13026

[38] L. Westover. Footprint evaluation for volume rendering. *ACM Siggraph Computer Graphics*, 24(4):367–376, 1990.

[39] G. Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, 1990.

[40] Z. Xu and J. Sun. Image inpainting by patch propagation using patch sparsity. *IEEE Transactions on Image Processing*, 19(5):1153–1165, 2010. doi: 10.1109/TIP.2010.2042098

[41] L. Yang, Y.-C. Tse, P. V. Sander, J. Lawrence, D. Nehab, H. Hoppe, and C. L. Wilkins. Image-based bidirectional scene reprojection. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, pp. 150:1–150:10. ACM, 2011. doi: 10.1145/2024156.2024184

[42] R. A. Yeh, C. Chen, T. Yian Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do. Semantic image inpainting with deep generative models. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.