DOI: 10.1111/cgf.70215
High-Performance Graphics 2025
A. Knoll and C. Peters
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 8

Image-Based Spatio-Temporal Interpolation for Split Rendering

M. Steiner[†1], T. Köhler[†1], L. Radl[1], B. Budge[2], and M. Steinberger[1]

[1]Graz University of Technology, Austria
[2]Meta Reality Lab Research, USA

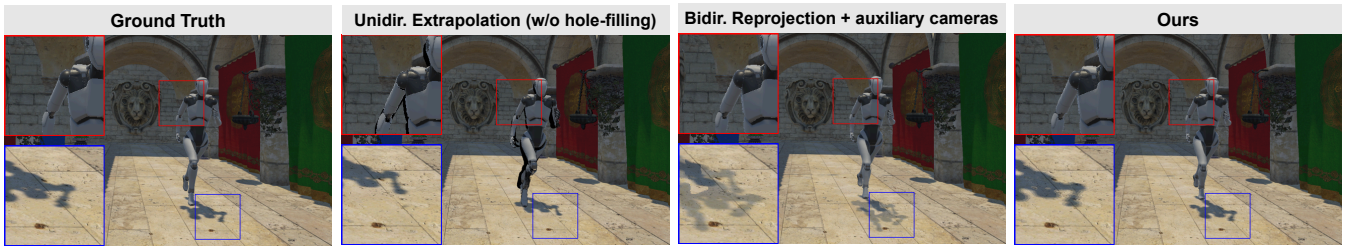


Figure 1: *Our method performs image-based spatio-temporal interpolation in a challenging split rendering setting, where the low-powered client receives server frames that are far apart, both temporally and spatially: Pure extrapolation from previous frames leads to unrealistic motion for fast objects, and reveals many disocclusions. Bidirectional reprojection between the previous and next predicted pose, combined with information from auxiliary cameras, helps to produce smooth motion and fill in disocclusions. Finally, our method encodes and transfers additional high-frequency shading information, like dynamic shadows, to produce high-quality interpolated frames.*

Abstract

Low-powered devices – such as small form factor head-mounted displays (HMDs) – struggle to deliver a smooth and high-quality viewing experience, due to their limited power and rendering capabilities. Cloud rendering attempts to solve the quality issue, but leads to prohibitive latency and bandwidth requirements, hindering use with HMDs over mobile connections or even over Wifi. One solution – split rendering – where frames are partially rendered on the client device, often either requires geometry and rendering hardware, or struggles to generate frames faithfully under viewpoint changes and object motion. Our method enables spatio-temporal interpolation via bidirectional reprojection to efficiently generate intermediate frames in a split rendering setting, while limiting the communication cost and relying purely on image-based rendering. Furthermore, our method is robust to modest connectivity issues and handles effects such as dynamic smooth shadows.

CCS Concepts

• ***Computing methodologies*** → ***Image-based rendering;*** *Rendering;*

1. Introduction

Immersive applications like virtual gaming, telepresence, and training simulations demand high-fidelity, low-latency graphics—delivered wirelessly and seamlessly to mobile head-mounted displays (HMDs). Yet current devices are severely constrained by thermal, power, and hardware limitations, making native rendering of complex, dynamic scenes at high frame rates increasingly infeasible. Meanwhile, content complexity continues to rise: modern rendering pipelines—such as Nanite in Unreal Engine—leverage vast amounts of detailed geometry, pushing the boundaries of what lightweight clients can handle. Rendering such content or even just rasterizing the geometry directly on-device is not a viable option. Cloud rendering can bridge the performance gap but comes at a steep cost: prohibitive motion-to-photon latency, high bandwidth requirements, and a dependence on stable, low-latency connectivity. These drawbacks are especially damaging in untethered VR, where unpredictable and fast head motion makes even minor delays unacceptable.

Split rendering—distributing the rendering workload between server and client—offers a promising compromise. However, many existing approaches either transmit geometric data [MVD*18; HSV*22], which is increasingly impractical given bandwidth and decoding constraints, or rely purely on image-based warp-

† Both authors contributed equally to this work

ing [BMS*12; YTS*11], which breaks down under fast motion, large viewpoint changes, and dynamic lighting.

In this work, we present an *image-based split rendering technique* that addresses these issues through *spatio-temporal interpolation*, without requiring geometry on the client. Our method is designed to handle extreme frame rate mismatches—e.g., a server rendering at 10 FPS while the client displays at 120 FPS—and remains robust even under delayed server data or unpredictable head movement. To generate high-quality intermediate frames without relying on rasterization hardware, we employ *bidirectional reprojection* based on 3D scene flow, allowing us to interpolate and extrapolate views temporally. We use *forward splatting with iterative backward search* to improve the reprojection accuracy and minimize artifacts from occlusions or flow mismatches. To address disocclusions caused by head motion and scene changes, we introduce *auxiliary cameras* that provide additional coverage and visibility. Finally, we develop a compact *dynamic shadow encoding and reprojection* scheme that preserves realistic shading effects across frames, even in the presence of fast object or light movement.

While recent works like Yang et al. [YZZ*24] explore frame generation, they only focus on minor frame rate increases (e.g. 60→120 FPS) and do not operate under a split rendering paradigm. In contrast, we tackle much more demanding scenarios—both temporally and spatially—and demonstrate our method's robustness across a range of dynamic scenes. Our results demonstrate visually consistent frames at high frame rates (120 FPS on the client) and efficient bandwidth usage, all while avoiding the need for geometry transmission. Our main contributions are:

- a novel spatio-temporal interpolation method tailored for image-based split rendering,
- a robust reprojection pipeline leveraging bidirectional scene flow, iterative search, and auxiliary cameras,
- a dynamic shadow encoding and reprojection mechanism that preserves shading realism under fast object motion,
- and a complete system optimized for both visual fidelity and bandwidth efficiency, evaluated across challenging scenarios.

These contributions enable a practical, high-quality rendering pipeline for mobile VR, delivering smooth, immersive visuals even under sparse server updates and real-world network conditions.

2. Related Work

Image-Based Rendering Foundations and Advances. Image-based rendering (IBR) has a long history of generating novel views without full geometry. Early techniques like view interpolation [CW93], the Lumigraph [GGSC96], and Layered Depth Images (LDI) [SGHS98] provided foundational tools for this task. Later work introduced more dynamic and efficient methods, including post-rendering warping [MMB97], image-space acceleration [NSL*07], and bidirectional reprojection [YTS*11]. Iterative extensions improved depth refinement [BMS*12; LKE18], leveraged reflective surfaces [LRR*14], or used proxy-guided warping [RKR*16], and were even deployed in mobile contexts [SNC12].

Additional efforts focused on perceptual quality and realism: perceptually adaptive rendering [DER*10; DRE*10] and image-based depth-of-field effects [YWY10; LKC08] helped enhance interactive rendering fidelity. Meanwhile, frame interpolation and hole-filling methods have been developed to address temporal upsampling and view completion, using neural networks [BDM*21; WKZ*23; WVS*24], gradient-domain editing [PGB03], or metameric inpainting [dAWA*23]. Yang et al. [YZZ*24] use neural methods for minor temporal upsampling.

While these works have substantially advanced IBR, they often assume access to full geometry, rely on the availability of a rasterizer, or target relatively modest upsampling tasks. Our method builds on these ideas but is designed for challenging split-rendering settings, extending image-based reprojection to handle both significant spatio-temporal interpolation and dynamic lighting effects, without requiring geometry or high bandwidth.

Cloud Rendering and Free-Viewpoint Systems. The limitations of mobile devices have driven extensive work in cloud-assisted rendering. Systems like CloudVR [KSEY18], FlashBack [BCC16], and Furion [LHC*17] stream rendered content from the cloud to lightweight clients, while approaches like Outatime [LCC*15] exploit frame prediction to mitigate latency. Others reduce transmission costs through view-region optimization [LBR*18], impostor-based rendering [MFL21], or by decoupling shading and geometry for efficient texture streaming [MVD*18].

In parallel, free-viewpoint and 3D-TV systems [MFY*09; ZDD10; NKD*11; NKDW08; GHZ*24; LTV*16; DBZD12; SA12] have explored depth-based view synthesis and GPU-accelerated pipelines for immersive multi-view rendering, often under fixed-view or wired conditions.

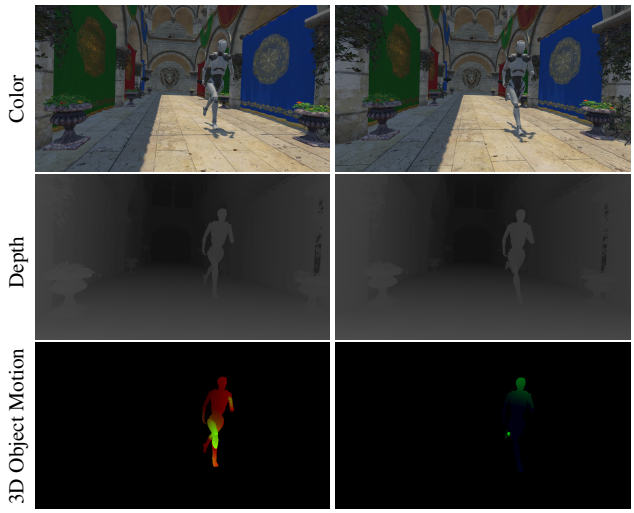
More recently, split rendering has emerged as a hybrid alternative. Hladky et al. [HSV*22] use an approximate geometric scene representation for client-side rendering; and XRGo [GLJ*25] dynamically balances workload between server and client. Compared to these, our approach targets more extreme interpolation (spatial and temporal), avoids G-buffers or geometry, and introduces techniques specifically for mobile VR scenarios, such as bidirectional reprojection and auxiliary views to handle disocclusion and dynamic shading.

3. Method

We propose an image-based split rendering method that reconstructs high-frame-rate client views from sparsely rendered server frames. By relying solely on image data, our approach avoids geometry transmission and client-side rasterization. Using bidirectional scene flow, auxiliary cameras, and dynamic shadow encoding, we achieve robust spatio-temporal interpolation, even under disocclusion and fast shading changes.

3.1. Preliminaries: Bidirectional Reprojection

Bidirectional image-based reprojection [YTS*11] relies on information from frames F_t and F_{t+1}, to generate intermediate frames $F_{t+\alpha}$, with $\alpha \in [0,1]$. Each frame provides *stationary* camera information $C_t = \{I_t, Z_t, P_t\}$, consisting of the color image I_t, z-buffer



(a) *Forward* **(b)** *Backward*

Figure 2: *Example of the forward and backward information needed to perform bidirectional reprojection.*

Z_t, and view-projection transform $P_t(\mathbf{x})$, which transforms a homogeneous world space coordinate $\mathbf{x}$ into C's normalized device coordinate (NDC) space (including perspective division). Reprojecting dynamic objects requires additional *transitional* information in the form of forward and backward 3D object motion $O_{t\to t+1}$ and $O_{t+1\to t}$, which hold 3D velocity for all visible surface points in C_t and C_{t+1} between time points t and $t+1$. Storing O as 3D *world space* motion, leads to sparse information when large parts of an image contain static objects, allowing for extensive compression. We show an example of these buffers for a forward/backward image pair in Fig. 2. The NDC scene flow at time point $t+\alpha$ for each pixel from camera C_t to another camera defined via the view-projection transform P_{to} can be computed from its 2D NDC pixel coordinates $\mathbf{p} = (p_x, p_y)$ as

$$V(C_t, O_{t\to t+1}, P_{\text{to}}, \alpha)[\mathbf{p}] = P_{\text{to}}(P_t^{-1}(\bar{\mathbf{p}}) + \alpha O_{t\to t+1}[\mathbf{p}]) - \bar{\mathbf{p}}, \quad \text{with} \quad \bar{\mathbf{p}} = (p_x, p_y, Z_t[\mathbf{p}], 1)^\top. \tag{1}$$

In their work, Yang et al. [YTS*11] compute the complete forward and backward scene flow fields V^f_{t+1}, V^b_{t+1}, which contain the flow of each visible surface point in C_t into the NDC space of C_{t+1}, and vice versa

$$V^f_{t+1} = V(C_t, O_{t\to t+1}, P_{t+1}, 1), \tag{2}$$
$$V^b_{t+1} = V(C_{t+1}, O_{t+1\to t}, P_t, 1). \tag{3}$$

While not explicitly computed in their work, we will denote the scene flow from C_t and C_{t+1} to the interpolated space at $t+\alpha$ as

$$V^f_{t+\alpha} = \alpha V^f_{t+1}, \quad V^b_{t+\alpha} = (1-\alpha) V^b_{t+1}. \tag{4}$$

To find the forward/backward correspondending pixels $\hat{\mathbf{p}}^{f/b}$ in F_t and F_{t+1} for a pixel $\mathbf{p}_{t+\alpha}$ in $F_{t+\alpha}$, they then perform fixed-point iteration (FPI) on these flow fields:

$$\mathbf{p}^{f/b}_i = \mathbf{p}_{t+\alpha} - V^{f/b}_{t+\alpha}[\mathbf{p}^{f/b}_{i-1}].xy, \text{ with } \quad \mathbf{p}^{f/b}_0 = \mathbf{p}_{t+\alpha}, \tag{5}$$
$$\hat{\mathbf{p}}^{f/b} = \mathbf{p}^{f/b}_{N_I}, \tag{6}$$

with $i \in \{1, \dots, N_I\}$ for N_I iterations, and the simplest approach initializing the FPI with $\mathbf{p}_{t+\alpha}$. This approach linearly interpolates between two NDC spaces, which leads to two pressing issues: (1) It only allows for temporal upsampling without the ability to react to a divergent client camera, as the concept of a camera projection $P_{t+\alpha}$ does not exist, making it undesirable for split rendering; (2) It assumes that a surface point that is visible in C_t and C_{t+1} maps to the same pixel in the reprojected image at $t+\alpha$, which fails due to the non-linearity of the projective transform.

Bidirectional Visibility and Blending. To determine the best correspondence between the forward and backward results $\hat{\mathbf{p}}^f$ and $\hat{\mathbf{p}}^b$, both have to be compared based on depth and projection error. Their respective reprojection errors $e^{f/b}$ and projected depths $z^{f/b}$ at $t+\alpha$ can be computed as

$$e^f = \|(\hat{\mathbf{p}}^f + V^f_{t+\alpha}[\hat{\mathbf{p}}^f].xy) - \mathbf{p}_{t+\alpha}\|, \tag{7}$$
$$z^f = Z_t[\hat{\mathbf{p}}^f] + V^f_{t+\alpha}[\hat{\mathbf{p}}^f].z, \tag{8}$$

and vice versa for e^b and z^b (using $V^b_{t+\alpha}$, $\hat{\mathbf{p}}^b$, and Z_{t+1}). Algorithm 1 shows how to determine the better corresponding match between $\hat{\mathbf{p}}^f$ and $\hat{\mathbf{p}}^b$, by using a reprojection error threshold ϵ_e and a depth similarity threshold ϵ_z, both given in NDC units. Finally, instead of simply using the color of the better match, the point can be projected into the next frame to find a better correspondence there (cf. Algorithm 2). The colors are then linearly blended based on α, which leads to a smoother shading transition.

ALGORITHM 1: Choosing the best match between the forward and backward reprojection results [YTS*11].

Data: e^b, e^f, z^b, z^f, Reprojection errors and depths
Result: Choice of best corresponding match
if $e^b < \epsilon_e \wedge e^f < \epsilon_e$ **then**
  if $|z^b - z^f| < \epsilon_z$ **then**
    `/* Correspond to same point */`
    ⇒ choose the match with the smaller error;
  else
    `/* Both correspond, but one occludes the other */`
    ⇒ choose the match closest to the camera;
  end
else if $e^b < \epsilon_e \vee e^f < \epsilon_e$ **then**
  `/* Only one match corresponds */`
  ⇒ choose the match which corresponds;
else
  `/* No match corresponds */`
  ⇒ choose either the match with the smaller error [YTS*11] or **none** (Ours);
end

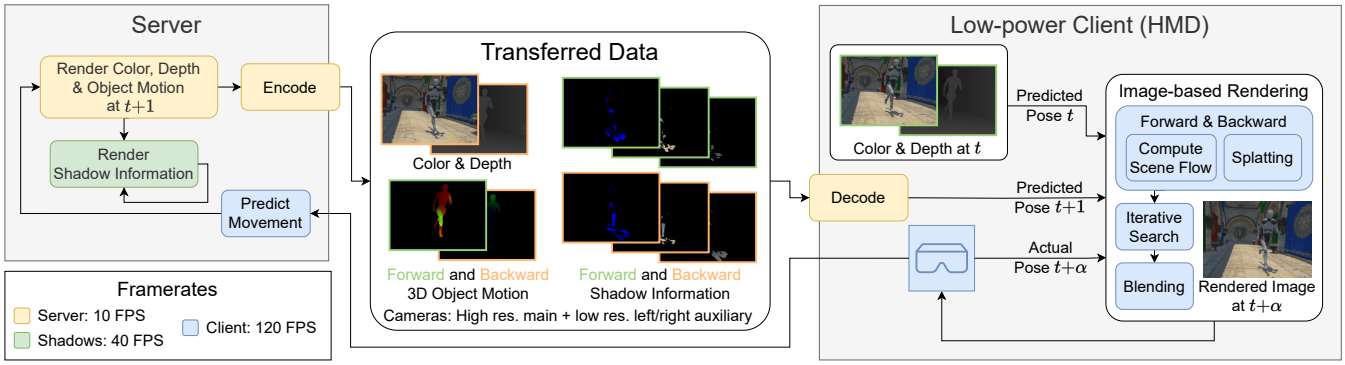


Figure 3: *Pipeline overview of our method. The **server** receives frequent pose updates from the client, and predicts the future camera pose at $t+1$. It then renders color, depth, object motion (10 FPS), and dynamic shadow information (40 FPS). To prevent disocclusions artifacts, we also render and transfer buffers for lower resolution auxiliary camera (left & right). All buffers are then efficiently encoded to fulfill the required bandwidth demands and transferred. The **client** decodes this data and performs image-based bidirectional reprojection at 120 FPS from the previous (t) and predicted next ($t+1$) camera frame to the actual pose ($t+\alpha$).*

ALGORITHM 2: Second chance reprojection and blending of a corresponding match.

Data: Matching point $\hat{\mathbf{p}}$ in source camera frame C,
Destination camera frame C',
Forward/backward 3D object motion O and O',
Blending weight α.
Result: The final blended RGB color $\mathbf{c}$.

$\hat{\mathbf{p}}' \leftarrow V(C,O,P',1)[\hat{\mathbf{p}}]$;
$\mathbf{c} \leftarrow$ `Color`$(C,\hat{\mathbf{p}})$; `// Simply` $I[\hat{p}]$ `for [YTS*11]`
if `in_range`$(\hat{\mathbf{p}}',C') \wedge |Z[\hat{\mathbf{p}}] - Z'[\hat{\mathbf{p}}']| < \epsilon_z$ **then**
  if $\|O[\hat{\mathbf{p}}] + O'[\hat{\mathbf{p}}']\| < \epsilon_O$ **then**
    `/* Check if forward and backward motion cancel out (not done in [YTS*11]) */`
    $\mathbf{c}' \leftarrow$ `Color`$(C',\hat{\mathbf{p}}')$;
    $\mathbf{c} \leftarrow$ `lerp`$(\mathbf{c},\mathbf{c}',\alpha)$;
  end
end

FPI Initialization. As multiple pixels can reproject to the same pixel $\mathbf{p}_{t+\alpha}$ at different depths, we want to find the match that is closest to the camera. The task of finding a good initialization of FPI is crucial, as it heavily influences if the algorithm will converge to the correct result. Multiple initialization strategies have been proposed: Yang et al. [YTS*11] offset the starting point for t by the scene flow of $t+1$, and exploit temporal coherence by reusing information from the previous interpolated frame, along with a small window search. We find that these strategies fail in challenging scenarios, *e.g.* for fast object or camera movements. Bowles et al. [BMS*12] subdivide the source image into pixel quads, and warp and rasterize those quads in the new camera frame, which leads to robust results, but is undesirable as it requires rasterization hardware on our client device. For non-temporal stereoscopic reprojection, a small fixed horizontal offset can also lead to a better initialization [BMS*12]. Lee et al. [LKE18] employ a stochastic search inside a tightly bounded window, where the quality ultimately depends on the number of samples taken. Other works initialize the search through forward point splatting and a small-window search [LRR*14], which we also found to produce the most robust results.

3.2. Pipeline Overview

We show an overview of our method's pipeline in Fig. 3. We assume a low-powered client device that does not need to contain any specialized rasterization or ray tracing hardware – *e.g.* a small form-factor HMD. The client is connected to a powerful server over a wireless connection, assuming a reasonably stable connection with a bandwidth between 10-100 Mbit/s. For a responsive experience, we assume a client frame rate of f_c=120 FPS. The server receives frequent pose updates from the client, allowing for reasonably accurate prediction of the future pose P_{t+1}, *e.g.* roughly 200 ms in advance. This enables the server to render a future frame F_{t+1} at the predicted pose, and to send the data such that the client receives the information of frame F_t and F_{t+1} in time to render a frame at time $t+\alpha$, with $\alpha > 0$. Our method can also extrapolate beyond $\alpha > 1$ to some extent, making it robust to slightly unstable connections with unpredictable latency. For example: If the server renders and transfers frames at f_s=10 FPS, the client temporally interpolates the eleven in-between frames – additionally, all frames are spatially interpolated for the client's actual current pose $P_{t+\alpha}$.

The transferred data from the server to the client for frame F_{t+1}, consists of the following buffers:

1. *Stationary* information C_{t+1}, containing the static render information at time $t+1$:
   a. View-projection transform P_{t+1}
   b. RGB image color I_{t+1}
   c. NDC z-buffer Z_{t+1}, transferred in reverse $[0,1]$ range
2. Forward and backward *transitional* information between time points t and $t+1$ for all visible surface points in C_t and C_{t+1}:
   a. 3D object motion buffers $O_{t\to t+1}$, and $O_{t+1\to t}$. We compute and transfer these buffers in the NDC space of C_t and C_{t+1} respectively, as this confines motion of almost all moving

points to the range $[-1,1]$, and make sure to apply the motion linearly in world space on the client side.

b. Dynamic shading information $S_{t\to t+1}$ and $S_{t+1\to t}$, storing information about high-frequency shading effects (cf. Sec. 3.3 for our encoding of dynamic soft shadows).

Additionally, to address disocclusions that inevitably happen during spatio-temporal interpolation, we propose to use additional auxiliary cameras, rendering at a lower resolution and with a larger field of view (FOV). As our setup is ultimately designed for a stereoscopic HMD setting, we choose to place two auxiliary cameras to the left and right of the main camera (see Sec. 4.1 for our exact experimentation setup).

3.3. Dynamic Soft Shadows

Fast changing shading effects, like shadows, are especially challenging for frame interpolation, as they require a distinction between object motion and optical flow. Linearly blending between two frames with dynamic shadows leads to undesired in/out-fading of shadows (see Fig. 4). We propose to render additional intermediate frames on the server, and sparsely encode information about dynamic shadows to reapply them on the client.

Server: We track shading changes for all visible surface points in C_t and C_{t+1} between time steps t and $t+1$, by rendering additional intermediate frames $C_{t+\beta_i}$ at regular intervals, and repeatedly projecting all points from C_t and C_{t+1} into $C_{t+\beta_i}$ using the known object motion and depth. We encode the dynamic shadows in $C_{t+\beta_i}$ by quantizing the screen space shadow map values for the scene's main light source, *e.g.* by using 2 bits b_i per intermediate frame $C_{t+\beta_i}$, and accumulating these values in a bitfield per pixel. Additionally, we store the RGB color values $S_{\min}/S_{\max}$ of the shaded pixels for the minimum/maximum encountered shadow values $b_{\min}/b_{\max}$ (cf. Fig. 5 for an example). For example: A server frame rate $f_s = 10$ FPS, and dynamic shadow frame rate $f_d = 40$ FPS results in the rendering of three additional frames at regular intervals $\beta_i \in \{0, 0.25, 0.5, 0.75\}$, a $4 \cdot 2 = 8$ bit bitfield, and accompanying color values for both forward and backward direction. **Sparsification & Compression:** We discard all pixels where the quantized shadow value is constant across all subframes (*i.e.* $b_{\min} = b_{\max}$), resulting in very sparse buffers, as usually only few pixels are effected by dynamic shadows in-between frames. Additionally, we set $S_{\min}$ to zero if the initial shadow value b_0 is equal to $b_{\min}$ (as the color is already present in I_t), and vice versa for $S_{\max}$ and $b_{\max}$. Importantly, the shadow information bitfield needs to be transmitted with a lossless encoding (cf. Sec. 3.4). **Client:** Before starting the projection for a frame at $t+\alpha$, we apply the dynamic shadow information to the images I_t and I_{t+1}. As shadow information is only present at fixed intervals $t+\beta_i$, and quantized shadow values b_i can differ from the minimum/maximum shadow values $b_{\min}/b_{\max}$, we need to interpolate: (1) temporally depending on the current α and its previous and next time points $\beta_i \le \alpha < \beta_{i+1}$ via $\tau_\beta \in [0,1]$:

$$\alpha = \mathrm{lerp}(\beta_i, \beta_{i+1}, \tau_\beta) \Rightarrow \tau_\beta = \frac{\alpha - \beta_{\min}}{\beta_{\max} - \beta_{\min}}, \tag{9}$$

and (2) the min/max shadow color based on the quantized shadow values $b_i, b_{i+1} \in [b_{\min}, b_{\max}]$ via $\tau_{(b,i)}, \tau_{(b,i+1)} \in [0,1]$:

$$b_i = \mathrm{lerp}(b_{\min}, b_{\max}, \tau_{(b,i)}) \Rightarrow \tau_{(b,i)} = \frac{b_i - b_{\min}}{b_{\max} - b_{\min}}, \tag{10}$$

to compute the final shaded color value $\mathbf{c}_S$ as

$$\mathbf{c}_S = \mathrm{lerp}(\mathrm{lerp}(S_{\min}, S_{\max}, \tau_{(b,i)}), \mathrm{lerp}(S_{\min}, S_{\max}, \tau_{(b,i+1)}), \tau_\beta). \tag{11}$$

3.4. Compression & Encoding

The combination of low-latency requirements and the substantial volume of auxiliary data produced by our method—which can reach up to 1.6 Gbit/s raw data—necessitates efficient compression to enable real-time streaming within practical bandwidth limits.

Color Images. Real-time streaming of 8-bit RGB color images is a well-established problem with mature hardware and software solutions. However, at high resolutions and frame rates, efficient compression remains critical. In our setup, we employ hardware-accelerated HEVC encoding via NVIDIA's NVENC to compress both the color and shadow color buffers.

Floating-Point Images. Transmitting floating-point buffers such as depth maps and optical flow fields poses a greater challenge, as standard video codecs are not directly designed for high-precision data. Prior work has explored temporally-aware float compression [JB20], general-purpose float compression [Lin14], and lossy quantization schemes [NSS14]. Our objective is to maximize visual and numerical fidelity while minimizing bandwidth usage. Through empirical evaluation, we found that linearly scaling 16-bit floating-point data to fit within 12-bit RGB channels, followed by HEVC encoding, achieves favorable results in low-bitrate scenarios.

Shadow Information. Our system encodes shadow information as a per-pixel bit-mask, where even minor perturbations can lead to significant visual artifacts or incorrect results. Consequently, it is crucial to preserve this data without any loss. To ensure bit-exact transmission, we apply lossless compression directly to the raw mask data using the Zstandard [Col16] algorithm, which provides an efficient trade-off between compression ratio and performance suitable for real-time applications.

3.5. Reprojection

Our client performs bidirectional image-based reprojection, which we adapt to our challenging spatio-temporal interpolation scenario.

Firstly, as we attempt to interpolate temporally for large time differences (*e.g.* 100-200ms), and where the actual camera pose $P_{t+\alpha}$ can deviate from the predicted interpolation path between start/end-poses P_t and P_{t+1}, our backward search has to be extraordinarily robust. In contrast to Yang et al. [YTS*11], who simply shift the forward and backward scene flow based on α (cf. Eqn. (4)), we explicitly construct the scene flow with respect to the actual client pose $P_{t+\alpha}$, *i.e.*

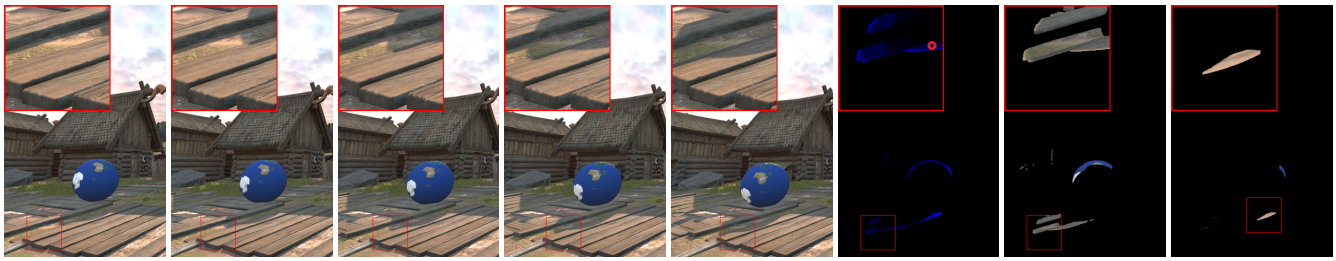
$$V^f_{t+\alpha} = V(C_t, O_{t\to t+1} P_{t+\alpha}, \alpha), \tag{12}$$

$$V^b_{t+\alpha} = V(C_{t+1}, O_{t+1\to t}, P_{t+\alpha}, (1-\alpha)). \tag{13}$$

For initialization of our backward search, we found that using

(a) *GT* (t) **(b)** *GT* $(t+1)$ **(c)** *GT* $(t+0.5)$ **(d)** *Linear Blending* $(t+0.5)$ **(e)** *Ours* $(t+0.5)$

Figure 4: *Example of a dynamic shadow in the Robot Lab scene, with ground truth (GT) frames at different time points, showing the quality difference between linear blending and our method, which encodes and tracks dynamic shadows at a higher frame rate.*



(a) t **(b)** $t+0.25$ **(c)** $t+0.5$ **(d)** $t+0.75$ **(e)** $t+1$ **(f)** *Bitfield* **(g)** *Min. color* **(h)** *Max. color*

Figure 5: *Example of the dynamic shadow from a rolling ball; The forward shadow information is extracted on the* ***server*** *from the screen space shadow map of three rendered intermediate frames at* $\{t+0.25, t+0.5, t+0.75\}$*. The information is encoded in a bitfield with 2 bits per frame (visualized in the blue channel): the example pixel (red dot) has the binary value* `11111100`*, meaning it is entirely out of shadow (*`00`*) at the initial frame* t *and entirely in shadow (*`11`*) for the remaining three. The two color buffers store the accompanying min./max. shadow color information (only if it is not present in the initial frame* t*). Note that the shadow information is also tracked on the moving ball itself.*

depth-aware forward point splatting based on these flow fields, combined with a 3×3 window search to find the point closest to the camera—as also employed by Lochmann et al. [LRR*14]—leads to a very robust initialization. As these initializations are very accurate, FPI only requires $\sim$3 iterations to converge. In the case that the next server frame arrives delayed (*i.e.* $\alpha > 1$), our method gracefully transitions to unidirectional reprojection using only $V^b_{t+\alpha}$, providing robustness when dealing with an unstable connection.

Secondly, as disocclusions are much more frequent and noticeable in our challenging scenario—especially for a stereoscopic client—we employ a different strategy for handling the case where neither the forward nor backward search found a corresponding match (see Algorithm 1). Instead of just choosing the better match, we attempt to fill these holes with the information from our low-resolution auxiliary cameras, which employ the same initialization and FPI algorithm as the main camera. Similarly to Algorithm 1, we decide which auxiliary camera delivers the better match for a pixel based on reprojection error and depth, and finally compare the best *auxiliary* match against the match of the *main* camera:

1. If *main* found no match, then choose the *auxiliary* match.
2. If *main* found a match, then we only choose the *auxiliary* match if it is neither visible in the forward nor the backward frame of the *main* camera, *i.e.* being inside the view frustum and not occluded (when checking the depth buffer).
3. If neither *main* nor *auxiliary* find a match, we do not try to color this pixel, and leave the color of the previous frame in there.

This process ensures that we choose the main camera's result whenever possible, as it contains higher-resolution information, but still make proper use of the auxiliary information.

Finally, we adapt the correspondence check between $\hat{\mathbf{p}}^f$ and $\hat{\mathbf{p}}^b$ in Algorithm 2: Instead of only checking correspondence purely based on depth, we additionally check if their 3D object motion cancels out (within a threshold ϵ_O), which should hold true if they correspond to the same surface point. This resolves wrong correspondences in certain edge-cases, *e.g.* when a dynamic object occludes a static object, but both have very similar depth values.

4. Results

We evaluate the effectiveness and efficiency of our split rendering approach across a range of challenging scenarios. Our evaluation focuses on both qualitative and quantitative aspects of the system, including reconstruction quality, compression performance, and computational cost. We test our method in a controlled modular setup and compare against several image-based reprojection methods.

4.1. Implementation

To assess our method, we use an experimental setup that captures the essential components of an end-to-end split rendering pipeline. The system is divided into three main components.

Server: All server-side buffers are generated using Unity's Universal Render Pipeline [Uni24]. Color data is represented as 8-bit RGB images, while floating-point data such as depth and optical flow is maintained in 16-bit half-precision format to preserve numerical accuracy.

Compression: The compression stage processes the server-side buffers and encodes them into continuous image streams suitable for real-time transmission. Color and float-derived buffers are compressed using the HEVC codec with low-latency settings to minimize delay and avoid intermediate buffering. Shadow bitmask data, which requires exact reconstruction, is compressed losslessly using Zstandard [Col16] at compression level 10. Under a fixed bandwidth budget, we allocate 60% of the available bandwidth to the main camera and 20% to each of the two auxiliary cameras. After subtracting the fixed-size shadow bitmask buffer—transmitted losslessly—the remaining bandwidth is distributed among the image buffers. Empirically, we found that the most effective split consists of 50% for color, 20% for depth, 20% for flow fields, and 10% for shadow color information.

Client: Client-side frame interpolation is implemented in C++ with CUDA to support efficient parallel execution. For our evaluation, we set the reprojection error threshold ϵ_e to 0.002 for the main camera, which approximately corresponds to a 2-pixel deviation in normalized device coordinates. The depth similarity threshold ϵ_z is configured to 0.001. To account for the lower resolution of the auxiliary cameras, both thresholds are scaled by a factor of four. Input frames are divided into 16×16 pixel tiles to enable tile-parallel processing and maintain performance at high resolutions.

4.2. Evaluation

We perform our evaluation on four different Unity scenes, with three scenes containing dynamic content: Viking Village [Unid] contains multiple moving objects and a fast moving rolling sphere [Sol], Sponza [Unia; McG] contains a fast moving camera following an animated running robot [Unic], and Robot Lab [Unib; Vik]. The evaluated camera paths are 8s long for the Sponza scene, and 15s for the other scenes, and contain animated characters, rotating and translating complex objects, and dynamic soft shadows from a main light source. We render the ground truth images in Unity and quantitatively evaluate our results using the SSIM, and FLIP [ANA*20] image metrics. Additionally, we evaluate using the VMAF [LAK*16] perceptual video quality metric, which can be interpreted as a linear mapping between human opinion scores ("bad", "poor", "fair", "good", and "excellent") from 0−100, with 100 being "excellent" and 20 considered "poor" [LBN*18].

Camera Setup. We use a multi-camera setup, with a single high-resolution, low-FOV main camera (2560×1440, 60°), and low-resolution, high-FOV auxiliary cameras (1280×720, 80°). If not stated otherwise, the server operates at 10 FPS, the client at 120 FPS, and the shadow information is rendered at 40 FPS. Our camera paths are pre-determined, with deterministic perturbations: cameras are attached to an "anchor" inside the "head box", with the "head box" following an animated path through the scene; the anchor simulates a HMD user behavior and deviates from this path in translation and rotation. The server frames are generated with a 100ms offset on the anchor's animation, simulating an imperfect prediction ("prediction delay") of the user's exact future pose (cf. supplementary video).

Methods. Our method (*Ours*) uses two auxiliary cameras, located 20 cm to the left and right of the main camera (all scenes are scaled to real-world scale), and rotated 10° outwards. *Ours (extrapolate)* is a purely unidirectional reprojection version of *Ours*. All other methods (except *Backward Bidir.*) utilize a single auxiliary camera with the same rotation and translation as the main camera. We incorporated a number of image-based reprojection methods for camparison to our method: *Timewarp* uses planar reprojection of the previous frame at the average z-depth; *Forward Splatting* employs forward projection through simple point splatting and a 3×3 neighborhood search for the fragment closest to the camera (equal to our FPI initialization); *Backward Bidir.* performs backward projection and is a re-implementation of bidirectional scene reprojection [YTS*11].

Quantitative Evaluation. We evaluate the baseline quality of all methods on the uncompressed data in Tab. 1, which should serve as an upper limit to each method's reconstruction ability. These numbers are also included as dotted lines in Fig. 6, which contains the rate distortion curve for all methods to ablate the quality/bandwidth trade-off of our compression and encoding. Additionally, we include metrics for *Timewarp* with higher server frame rates, which also operate on better pose predictions: frame rates of 10/30/60 FPS use prediction delays of $100/33.\dot{3}/16.\dot{6}$ ms respectively; *Video Streaming* assumes perfect predictions (0 ms delay), *i.e.* just the video-encoded ground truth images. *Ours* outperforms all other methods on the uncompressed data, and also delivers the best quality/bandwidth trade-off on the compressed data, achieving "good" to "excellent" quality at ∼40−60 Mb/s. *Ours (extrapolate)* delivers similar image metrics when large parts of the scene are static, however, it fails for challenging scenarios with fast dynamic objects: (1) extrapolating non-linear motion based purely on the previous motion leads to rough transitions once the next frame arrives; (2) shading changes or dynamic shadows cannot be handled at all, leading to non-smooth shading transitions and unrealistic shadows; (3) disocclusions due to moving objects cannot be filled with useful information (cf. the supplemental video). *Forward Splatting* shows the same artifacts, but additionally leads to noisy results due to the 3×3 pixel footprint of each point, which is required to properly solve occlusions; this noise even leads to better metrics on the slightly blurrier compressed data vs. the uncompressed data (dotted line). *Timewarp* is unable to handle camera translation and object motion properly, which leads to bad results for our challenging scenario. When increasing the server frame rate to 30 or even 60 FPS, the generated videos get noticeably smoother, but the slightly incorrect reprojection still leads to inferior metrics. *Backward Bidir.* fails entirely, as it can only interpolate linearly between the server frames, and cannot account for a diverging client camera pose. Unsurprisingly, the compressed client *Video Streaming* at 120 FPS compares best on frame-to-frame image and video metrics. However, this setup relies on the unrealistic assumptions of zero-latency and/or perfect head movement prediction. *Timewarp*

(60 FPS), which more closely resembles a realistic streaming setup, achieves noticeably inferior image metrics to our method.

Table 1: *Averaged per-image metrics for our qualitative evaluation on the uncompressed data. This serves as an upper limit of each method's reprojection capabilities.*

	Viking Village					
	Static			Dynamic		
Method	SSIM↑	FLIP↓	VMAF↑	SSIM↑	FLIP↓	VMAF↑
Ours	0.923	0.043	84.80	0.923	0.043	84.55
Ours (extrapolate)	0.916	0.044	83.27	0.914	0.045	82.19
Forward Splatting	0.704	0.105	34.57	0.705	0.105	34.07
Timewarp	0.556	0.193	17.92	0.555	0.195	17.39
Backward Bidir.	0.477	0.283	6.03	0.478	0.283	5.87

	Sponza			Robot Lab		
Method	SSIM↑	FLIP↓	VMAF↑	SSIM↑	FLIP↓	VMAF↑
Ours	0.938	0.038	80.01	0.944	0.032	84.28
Ours (extrapolate)	0.922	0.044	74.83	0.932	0.036	80.64
Forward Splatting	0.714	0.094	31.19	0.766	0.077	37.75
Timewarp	0.522	0.200	5.58	0.615	0.150	11.70
Backward Bidir.	0.506	0.218	2.78	0.492	0.250	0.84

We also ablate our method's robustness to unstable connections in Tab. 2, by introducing an artificial latency/delay to the received server data. During this delay—*i.e.* until the data for frame F_{t+1} arrives—our method temporarily switches to unidirectional projection and extrapolates using the information from the previous server frame F_t. Small latencies are almost unnoticeable, both quantitatively and qualitatively (cf. supplemental video), especially for static scenes, or scenes with slowly moving objects. However, larger delays become noticeable for more challenging scenarios, *e.g.* fast, non-linearly moving objects, large shading changes, and dynamic shadows. Finally, Tab. 3 includes an evaluation for a stereo client setup (pupillary distance of 7 cm), where our method maintains high quality, and is able to fill disocclusions from the information in the left and right auxiliary cameras.

Table 2: *Ablation of our method's robustness to an unstable connection by introducing an artificial delay to the received server frames, i.e. the information for frame F_{t+1} arrives l client frames too late ($f_c = 120$ FPS, $f_s = 10$ FPS). For $l = 12$, our method degenerates to complete unidirectional reprojection (extrapolation). Rendered using the compressed data, with a target bit rate of 40 Mb/s.*

	Sponza		Robot Lab	
Method	SSIM↑	FLIP↓	SSIM↑	FLIP↓
Ours	0.876	0.064	0.924	0.044
Ours $l = 1$	0.876	0.064	0.924	0.044
Ours $l = 2$	0.876	0.064	0.924	0.044
Ours $l = 4$	0.874	0.064	0.923	0.044
Ours $l = 8$	0.867	0.066	0.918	0.046
Ours (extrapolate)	0.859	0.070	0.912	0.049

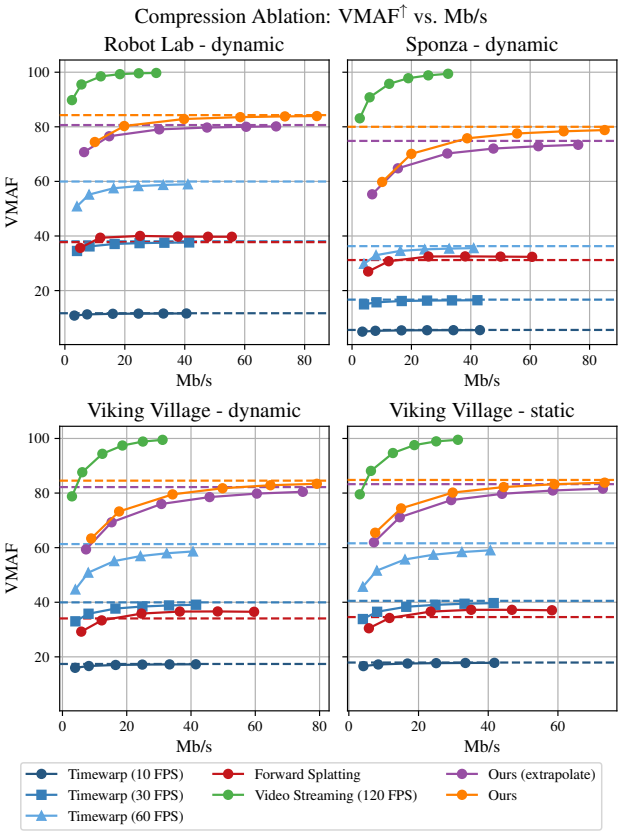


Figure 6: *Rate distortion curves VMAF (higher is better) vs. Mb/s, exploring the quality/bandwidth trade-off for our evaluated methods. Note that* Video Streaming *assumes a perfect prediction of the head movement and a zero-latency setup (0 ms);* Timewarp *follows a more traditional streaming setup, with higher framerates and a prediction window of one frame: 60 FPS, 16.7 ms; 30 FPS, 33.3 ms; 10 FPS, 100 ms. The dotted lines visualize the quality result for each method on the uncompressed data.*

Qualitative Evaluation. We provide a visual comparison in Fig. 7, as well as the supplemental video. Compression naturally leads to blurrier image buffers, but also to slight ghosting, as the color of dynamic foreground objects can bleed into the background, and vice versa (see "Earth sphere" in *Ours*). Excessive compression on the 3D object motion and depth buffers can lead to fuzzy edges at depth discontinuities; for the shown compression rate (target bit rate of 40 Mb/s), these artifacts are barely noticeable. *Forward Splatting* and *Ours (extrapolate)* struggle with dynamic shadows and disocclusions, revealed by moving objects; *Forward Splatting* additionally produces much noisier images. *Timewarp* is only suited for simple spatio-temporal frame generation tasks, and completely fails for our challenging scenario, as our scene contains a considerable divergence from the predicted camera position, fast moving objects, and large temporal differences between server frames. *Backward Bidir.* also fails for a divergent client camera, and while it should be able to handle moving objects, their employed FPI initialization

Table 3: *Comparison of all methods on the compressed data (target bit rate 40 Mb/s) for a stereoscopic client setup, with the averaged metrics from the left and right stereo cameras.*

	Sponza			
	Mono		Stereo	
Method	SSIM↑	FLIP↓	SSIM↑	FLIP↓
Ours	0.876	0.064	0.872	0.065
Ours (extrapolate)	0.859	0.070	0.862	0.069
Forward Splatting	0.715	0.096	0.720	0.098
Timewarp	0.542	0.194	0.537	0.200
Backward Bidir.	0.522	0.216	0.518	0.225
	Robot Lab			
	Mono		Stereo	
Method	SSIM↑	FLIP↓	SSIM↑	FLIP↓
Ours	0.924	0.044	0.922	0.045
Ours (extrapolate)	0.912	0.049	0.910	0.050
Forward Splatting	0.771	0.077	0.767	0.080
Timewarp	0.625	0.149	0.557	0.198
Backward Bidir.	0.498	0.250	0.490	0.258

fails to find good correspondences for thin or fast moving objects. We refer to our supplemental video for a more detailed qualitative comparison.

4.3. Computational Analysis

Our current prototype implementation is divided into three separate components. The primary objective of our work has been to develop a robust reprojection method that delivers high-quality and correct results, even under challenging conditions. To support this goal, we designed an efficient encoding scheme that minimizes bandwidth requirements. None of the components are currently optimized for runtime performance, as we prioritized experimental flexibility over execution speed. Still, the core steps of our image-based reprojection algorithm are lightweight, and similar techniques have previously been demonstrated to run efficiently on mobile hardware [YZZ*24].

Client. Our image-based reprojection algorithm runs in approximately 2 ms per frame (500 FPS) on a high-end consumer GPU (RTX 4090) for a 2560×1440 camera setup. The runtime is roughly split between two stages: (1) Computing flow fields, applying dynamic shadow encoding to the color images, and forward-projecting the original surface points to initialize FPI. Currently, each forward and backward image pair from the main, left, and right cameras performs these operations in separate, sequential kernel calls, revealing clear potential for parallelization. This stage is also heavily memory-bound, as flow and depth fields are read and written in 32-bit, despite our encoding using only 12-bit values—effectively tripling the bandwidth requirements in our prototype. Another bottleneck arises from the use of 64-bit `atomicMin` operations during depth-aware splatting, which combine 32-bit depth and 32-bit pixel ID into a single payload. Considering the limited depth range of our encoded format and bounded pixel movement, 32-bit atomics would be sufficient in practice—but would restrict experimental flexibility. (2) The actual FPI search is implemented as a single kernel, executed sequentially for all cameras. Due to the fixed and low iteration count per camera, and the absence of explicit geometry processing, this step is highly efficient and well suited for mobile deployment, as demonstrated in prior work [YZZ*24]. Exploiting parallelization across all camera views and reducing memory bandwidth can increase the performance of these steps by at least $4\times$, according to our experiments. By relying on dedicated hardware, it may be possible to eliminate the need to split the process into two steps, computing the input for FPI on the fly while utilizing caches to keep the data in fast memory. This could potentially result in an additional performance boost of $2\times$ or more.

Compression. Our compression pipeline is designed for real-time performance, targeting high compression ratios with minimal latency. Color images are encoded using HEVC via NVENC, leveraging dedicated hardware encoders for high throughput. While NVENC is highly efficient, streaming multiple buffers in parallel approaches the limits of available encoder bandwidth and hardware resources. Nonetheless, for our target frame rate of 5–10 FPS on professional GPUs, this setup remains feasible. For floating-point buffers (*i.e.* depth and object motion), we use 12-bit HEVC encoding to preserve fine detail; however, this requires slower, software-based encoding on consumer GPUs. In cloud-based deployments or with specialized hardware, dedicated 12-bit encoders can provide significantly higher throughput. In contrast, shadow masks are compressed using Zstandard [Col16], which offers fast, lightweight performance with real-time capabilities—without relying on specialized hardware.

5. Conclusion, Limitations & Future Work

We presented an image-based spatio-temporal interpolation method designed for split rendering on low-powered devices, such as mobile HMDs. Our image-based approach operates without geometry, minimizes bandwidth, and remains robust under motion, viewpoint changes, and dynamic shading—particularly shadows. Through bidirectional reprojection, iterative search, auxiliary views, and encoded shadow dynamics, our approach achieves high visual fidelity and interactive performance even under constrained network and compute conditions.

Our evaluations across multiple scenes demonstrate that our method outperforms prior image-based and G-buffer-based frame generation strategies, especially in handling disocclusions, dynamic content, and challenging prediction errors in user motion. The combination of low-latency reprojection and efficient compression provides a viable path toward responsive and visually coherent cloud-assisted rendering for mobile XR. While promising, our method currently relies on relatively heavy preprocessing and client-side decoding, which we plan to optimize for real-world deployment on embedded GPUs. Additionally, our system does not yet handle all forms of shading effects—such as caustics or volumetric lighting—and assumes moderate prediction quality for future viewpoints. Overall, we believe this work contributes a practical step toward bridging the gap between high-end rendering and mobile form-factor devices while keeping the experience smooth, immersive, and light on bandwidth.

(a) *Viking Village* **(b)** *Robot Lab* **(c)** *Sponza*

Figure 7: *Qualitative comparison of example views from our evaluated dynamic scenes for a compression target bit rate of 40Mb/s, with their corresponding FLIP error images. Extrapolation methods—Forward Splatting & Ours (extrapolate)—struggle with dynamic shadows, shading changes, and disocclusions behind moving objects. Timewarp & Backward Bidir. cannot handle a diverging client camera properly. Ours is able to handle these challenging scenarios, and can be streamed using efficient encoding and compression, with only a minor loss in quality compared to the uncompressed result.*

References

[ANA*20] ANDERSSON, PONTUS, NILSSON, JIM, AKENINE-MÖLLER, TOMAS, et al. "FLIP: A Difference Evaluator for Alternating Images". *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3.2 (2020) 7.

[BCC16] BOOS, KEVIN, CHU, DAVID, and CUERVO, EDUARDO. "FlashBack: Immersive Virtual Reality on Mobile Devices via Rendering Memoization". *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. 2016, 291–304 2.

[BDM*21] BRIEDIS, KARLIS MARTINS, DJELOUAH, ABDELAZIZ, MEYER, MARK, et al. "Neural Frame Interpolation for Rendered Content". *ACM Trans. Graph.* 40.6 (2021) 2.

[BMS*12] BOWLES, HUW, MITCHELL, KENNY, SUMNER, ROBERT W., et al. "Iterative Image Warping". *Computer Graphics Forum* 31.2 (2012), 237–246 2, 4.

[Col16] COLLET, YANN. *Zstandard - Real-time data compression algorithm*. https://facebook.github.io/zstd/. Accessed: 2025-04-09. 2016 5, 7, 9.

[CW93] CHEN, SHENCHANG ERIC and WILLIAMS, LANCE. "View Interpolation for Image Synthesis". *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. 1993, 279–288 2.

[dAWA*23] Dos ANJOS, RAFAEL KUFFNER, WALTON, DAVID, AKŞIT, KAAN, et al. "Metameric Inpainting for Image Warping". *IEEE Trans. Vis. Comput. Graph.* 29.12 (2023), 5511–5522 2.

[DBZD12] DO, LUAT, BRAVO, GERMAN, ZINGER, SVITLANA, and DE WITH, PETER HN. "GPU-accelerated Real-time Free-viewpoint DIBR for 3DTV". *IEEE Transactions on Consumer Electronics* 58.2 (2012), 633–640 2.

[DER*10] DIDYK, PIOTR, EISEMANN, ELMAR, RITSCHEL, TOBIAS, et al. "Perceptually-motivated Real-time Temporal Upsampling of 3D Content for High-refresh-rate Displays". *Computer Graphics Forum* 29.2 (2010), 713–722 2.

[DRE*10] DIDYK, PIOTR, RITSCHEL, TOBIAS, EISEMANN, ELMAR, et al. "Adaptive Image-space Stereo View Synthesis". *Vision, Modeling, and Visualization*. 2010 2.

[GGSC96] GORTLER, STEVEN J., GRZESZCZUK, RADEK, SZELISKI, RICHARD, and COHEN, MICHAEL F. "The Lumigraph". *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. 1996, 43–54 2.

[GHZ*24] GUO, SHUAI, HU, JINGCHUAN, ZHOU, KAI, et al. "Real-Time Free Viewpoint Video Synthesis System Based on DIBR and A Depth Estimation Network". *IEEE Transactions on Multimedia* (2024) 2.

[GLJ*25] GAO, STEVEN, LIU, JEFFREY, JIANG, QINJUN, et al. "XRgo: Design and Evaluation of Rendering Offload for Low-Power Extended Reality Devices". *Proceedings of the 16th ACM Multimedia Systems Conference*. 2025, 124–135 2.

[HSV*22] HLADKY, JOZEF, STENGEL, MICHAEL, VINING, NICHOLAS, et al. "QuadStream: A Quad-Based Scene Streaming Architecture for Novel Viewpoint Reconstruction". *ACM Trans. Graph.* 41.6 (2022) 1, 2.

[JB20] JUN, HANSEUL and BAILENSON, JEREMY. "Temporal RVL: A Depth Stream Compression Method". *IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops*. 2020, 664–665 5.

[KSEY18] KÄMÄRÄINEN, TEEMU, SIEKKINEN, MATTI, EERIKÄINEN, JUKKA, and YLÄ-JÄÄSKI, ANTTI. "CloudVR: Cloud Accelerated Interactive Mobile Virtual Reality". *Proceedings of the 26th ACM International Conference on Multimedia*. 2018, 1181–1189 2.

[LAK*16] LI, ZHI, AARON, ANNE, KATSAVOUNIDIS, IOANNIS, et al. *Toward A Practical Perceptual Video Quality Metric*. 2016. URL: https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652 7.

[LBN*18] LI, ZHI, BAMPIS, CHRISTOS, NOVAK, JULIE, et al. *VMAF: The Journey Continues*. 2018. URL: https://netflixtechblog.com/vmaf-the-journey-continues-44b51ee9ed12 7.

[LBR*18] LALL, PUNEET, BORAC, SILVIU, RICHARDSON, DAVE, et al. "View-Region Optimized Image-Based Scene Simplification". *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1.2 (2018) 2.

[LCC*15] LEE, KYUNGMIN, CHU, DAVID, CUERVO, EDUARDO, et al. "Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming". *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. 2015, 151–165 2.

[LHC*17] LAI, ZEQI, HU, Y. CHARLIE, CUI, YONG, et al. "Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices". *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. 2017, 409–421 2.

[Lin14] LINDSTROM, PETER. "Fixed-Rate Compressed Floating-Point Arrays". *IEEE Trans. Vis. Comput. Graph.* 20.12 (2014), 2674–2683 5.

[LKC08] LEE, SUNGKIL, KIM, GERARD JOUNGHYUN, and CHOI, SEUNGMOON. "Real-Time Depth-of-Field Rendering Using Point Splatting on Per-Pixel Layers". *Computer Graphics Forum* 27.7 (2008), 1955–1962 2.

[LKE18] LEE, SUNGKIL, KIM, YOUNGUK, and EISEMANN, ELMAR. "Iterative Depth Warping". *ACM Trans. Graph.* 37.5 (2018) 2, 4.

[LRR*14] LOCHMANN, GERRIT, REINERT, BERNHARD, RITSCHEL, TOBIAS, et al. "Real-time Reflective and Refractive Novel-view Synthesis". *Vision, Modeling and Visualization*. 2014 2, 4, 6.

[LTV*16] LE DINH, MINH, TUNG, LONG VUONG, VAN, XIEM HOANG, et al. "Improving 3D-TV View Synthesis Using Motion Compensated Temporal Interpolation". *2016 International Conference on Advanced Technologies for Communications (ATC)*. 2016, 312–317 2.

[McG] MCGUIRE COMPUTE GRAPHICS ARCHIVE. *Crytek Sponza*. The Atrium Sponza Palace in Dubrovnik, re-modeled by Frank Meinl at Crytek. Licensed under CC BY 3.0. URL: https://casual-effects.com/data 7.

[MFL21] MISIAK, MARTIN, FUHRMANN, ARNULPH, and LATOSCHIK, MARC ERICH. "Impostor-based Rendering Acceleration for Virtual, Augmented, and Mixed Reality". *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*. 2021 2.

[MFY*09] MORI, YUJI, FUKUSHIMA, NORISHIGE, YENDO, TOMOHIRO, et al. "View Generation with 3D Warping Using Depth Information for FTV". *Signal Processing: Image Communication* 24.1-2 (2009), 65–72 2.

[MMB97] MARK, WILLIAM R., MCMILLAN, LEONARD, and BISHOP, GARY. "Post-Rendering 3D warping". *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 1997 2.

[MVD*18] MUELLER, JOERG H, VOGLREITER, PHILIP, DOKTER, MARK, et al. "Shading atlas streaming". *ACM Transactions on Graphics (TOG)* 37.6 (2018), 1–16 1, 2.

[NKD*11] NDJIKI-NYA, PATRICK, KOPPEL, MARTIN, DOSHKOV, DIMITAR, et al. "Depth Image-Based Rendering With Advanced Texture Synthesis for 3-D Video". *IEEE Transactions on Multimedia* 13.3 (2011), 453–465 2.

[NKDW08] NDJIKI-NYA, PATRICK, KÖPPEL, MARTIN, DOSHKOV, DIMITAR, and WIEGAND, THOMAS. "Automatic Structure-Aware Inpainting for Complex Image Content". *Proceedings of the 4th International Symposium on Advances in Visual Computing*. 2008, 1144–1156 2.

[NSL*07] NEHAB, DIEGO, SANDER, PEDRO V., LAWRENCE, JASON, et al. "Accelerating Real-Time Shading with Reverse Reprojection Caching". *Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware*. 2007 2.

[NSS14] NENCI, FABRIZIO, SPINELLO, LUCIANO, and STACHNISS, CYRILL. "Effective compression of range data streams for remote robot operations using H.264". *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, 3794–3799 5.

[PGB03] PÉREZ, PATRICK, GANGNET, MICHEL, and BLAKE, ANDREW. "Poisson Image Editing". *SIGGRAPH*. 2003, 313–318 2.

[RKR*16] REINERT, BERNHARD, KOPF, JOHANNES, RITSCHEL, TOBIAS, et al. "Proxy-guided Image-based Rendering for Mobile Devices". *Computer Graphics Forum* 35.7 (2016), 353–362 2.

[SA12] SOLH, MASHHOUR and ALREGIB, GHASSAN. "Hierarchical Hole-Filling For Depth-Based View Synthesis in FTV and 3D Video". *IEEE Journal of Selected Topics in Signal Processing* 6.5 (2012), 495–504 2.

[SGHS98] SHADE, JONATHAN, GORTLER, STEVEN, HE, LI-WEI, and SZELISKI, RICHARD. "Layered Depth Images". *Proceedings of the ACM on Computer Graphics and Interactive Techniques*. 1998 2.

[SNC12] SHI, SHU, NAHRSTEDT, KLARA, and CAMPBELL, ROY. "A Real-Time Remote Rendering System for Interactive Mobile Graphics". *ACM Trans. Multimedia Comput. Commun. Appl.* 8.3s (2012) 2.

[Sol] SOLAR SYSTEM SCOPE. *Solar Textures*. Distributed under Attribution 4.0 International license. URL: https://www.solarsystemscope.com/textures 7.

[Unia] UNITY. *Classic Sponza - Unity Remaster*. URL: https://github.com/Unity-Technologies/Classic-Sponza 7.

[Unib] UNITY. *Robot Lab (Unity 4x)*. Asset no longer available. URL: https://assetstore.unity.com/packages/essentials/tutorial-projects/robot-lab-unity-4x-7006 7.

[Unic] UNITY. *Starter Assets - ThirdPerson*. Licensed under the Unity Companion License: https://unity.com/legal/licenses/unity-companion-license. URL: https://assetstore.unity.com/packages/essentials/starter-assets-thirdperson-updates-in-new-charactercontroller-pa-196526 7.

[Unid] UNITY. *Viking Village URP*. Licensed under the Standard Unity Asset Store EULA: https://unity.com/legal/as-terms. URL: https://assetstore.unity.com/packages/essentials/tutorial-projects/viking-village-urp-29140 7.

[Uni24] UNITY TECHNOLOGIES. *Unity*. https://unity.com/. Accessed: 2025-04-09. 2024 7.

[Vik] VIKTOROV, DMITRY. *Github - Robot Lab*. Robot Lab from the Unity Asset Store adopted for Unity 2018.3. URL: https://github.com/dmitry1100/Robot-Lab 7.

[WKZ*23] WU, SONGYIN, KIM, SUNGYE, ZENG, ZHENG, et al. "ExtraSS: A Framework for Joint Spatial Super Sampling and Frame Extrapolation". *SIGGRAPH*. 2023 2.

[WVS*24] WU, SONGYIN, VEMBAR, DEEPAK, SOCHENOV, ANTON, et al. "GFFE: G-buffer Free Frame Extrapolation for Low-latency Real-time Rendering". *ACM Trans. Graph.* 43.6 (2024) 2.

[YTS*11] YANG, LEI, TSE, YU-CHIU, SANDER, PEDRO V., et al. "Image-Based Bidirectional Scene Reprojection". *ACM Trans. Graph.* 30.6 (2011) 2–5, 7.

[YWY10] YU, XUAN, WANG, RUI, and YU, JINGYI. "Real-time Depth of Field Rendering via Dynamic Light Field Generation and Filtering". *Computer Graphics Forum* (2010) 2.

[YZZ*24] YANG, SIPENG, ZHU, QINGCHUAN, ZHUGE, JUNHAO, et al. "Mob-FGSR: Frame Generation and Super Resolution for Mobile Real-Time Rendering". *SIGGRAPH*. 2024 2, 9.

[ZDD10] ZINGER, SVETA, DO, LUAT, and DE WITH, PHN. "Free-viewpoint depth image based rendering". *Journal of Visual Communication and Image Representation* 21.5-6 (2010), 533–541 2.