ITUJournal
Future and evolving technologies

ITU Journal on Future and Evolving Technologies, Volume 6, Issue 2, June 2025

Article

Edge-assisted user-centric real-time 3D remote near-eye rendering for AR/MR headsets

***Bishakha Rani Biswas*[1], *Xueyu Hou*[1], *Yongjie Guan*[1]**

[1] *The University of Maine, USA*

Corresponding author: Yongjie Guan, yongjie.guan@maine.edu

Augmented Reality and Mixed Reality (AR/MR) headsets are transforming computing by enabling immersive 3D experiences, yet inherent size and power limitations prevent them from matching desktop systems in delivering complex graphics. As a result, many graphics-intensive applications cannot run natively on these devices. Remote rendering offers a promising alternative by offloading heavy 3D graphics computations to a server and streaming the rendered results to AR/MR headsets. However, conventional remote rendering approaches often suffer from considerable interaction latency over wireless networks, making them unsuitable for latency-sensitive applications. This paper introduces a novel low-latency remote rendering system that enables real-time 3D graphics on AR/MR headsets. By leveraging image-based rendering with advanced 3D image warping techniques, our system synthesizes headset displays from server-generated depth images. Experimental results demonstrate that our approach significantly reduces interaction latency while maintaining high rendering quality, achieved through the careful optimization of multiple-depth image generation strategies.

Keywords: 3D rendering, augmented reality, edge assisted, human-computer interaction

1. INTRODUCTION

The growing adoption of AR/MR devices is fundamentally reshaping how we interact with digital content [1, 2, 3]. However, despite their increasing popularity, these devices still face significant computational and bandwidth limitations [4, 5, 6]. As a result, many advanced 3D graphics applications, ranging from cutting-edge video games and sophisticated 3D model visualizations to immersive telepresence systems powered by multiple depth cameras, remain out of reach for these platforms. Such applications demand not only dynamic user interactions with real-time viewpoint updates, but also robust GPU performance to support high-fidelity rendering, highlighting the growing complexity and data intensity of modern 3D graphics content.

Even with significant advances in mobile GPU architectures and energy-efficient optimizations [7, 8, 9, 10], AR/MR devices continue to lag behind desktop systems in 3D rendering capabilities, largely constrained by their physical form factors and strict power consumption limits. Additionally, the challenge is exacerbated by wireless network bandwidth limitations, where the transmission of large 3D datasets can quickly saturate available resources, further restricting the potential of mobile immersive experiences.

Remote rendering has re-emerged as a promising solution by leveraging the computational power of resource-rich workstations or cloud servers [11, 12, 13, 14] to perform the intensive 3D rendering tasks. The resulting rendered images are streamed to mobile clients in near-real time, allowing lightweight AR/MR devices to deliver visually rich experiences without incurring heavy local computational loads. Although the concept of remote rendering originated decades ago, when even personal computers struggled with 3D graphics, it has gained renewed importance today to address the escalating performance demands of next-generation AR/MR systems.

2. RELATED WORK

2.1 Mobile clients and remote sharing

Mobile client frameworks like SLIM [15] and MobileC [16], along with remote desktop sharing systems such as VNC [17] and RDP [18], empower users to access applications hosted remotely and share computational resources. Although these systems allow client devices to interact with server-based applications, they differ fundamentally from the remote rendering systems explored in this paper.

Early mobile client and remote sharing solutions were developed before the advent of widespread 3D graphics rendering and were primarily focused on sharing desktop components and 2D content. The shift toward supporting 3D graphics emerged in later systems, as seen in MobileC [16] and TurboVNC [19]. Additionally, while 2D application sharing emphasizes protocols that efficiently update only the modified regions of the display, i.e., taking advantage of the relatively light computational demands of 2D rendering, this strategy does not translate well to 3D graphics, where rendering complexity increases dramatically. Moreover, 3D applications, such as video games, typically require full-screen updates and thus call for specialized compression and streaming techniques that are distinct from those used in conventional 2D mobile client systems.

2.2 Remote system methods and models

In the literature, server-client remote rendering and visualization systems are typically classified into two main categories based on the type of data generated by the server: model-based methods and image-based methods. In this section, we analyze these approaches within the context of the remote rendering model described earlier and provide a comparative evaluation of their respective strengths and weaknesses.

Systems that transmit 3D models (whether as triangle meshes or point clouds) to the client are classified as model-based methods. In this configuration, the server handles the computation involved in creating and manipulating the source data, which in turn requires the client to have sufficiently robust hardware to render the 3D graphics.

2.2.1 Model-based approach

Original model: In this approach, the entire application runs on the server, with all 3D models transmitted to the client for rendering. This method is ideal for applications that demand significant computational resources for model creation rather than for rendering. By leveraging this strategy, system libraries can seamlessly port any graphics application to a server-client remote visualization framework without source code modifications. Prominent examples include GLX [20], Game@Large [21], and THiNC [21].

Partial model: A major drawback of the original model is the considerable network bandwidth and time required to transmit complete 3D models, especially in scenes with complex geometries and textures, before rendering can commence on the client. In many cases, however, only a subset of the full model is necessary. For example, systems described in Engel for remote visualization of medical volume data transmit just a slice of data based on user focus, while Schmalstieg [22] employs a similar strategy for rendering virtual environments in remote walkthroughs. Although this method adds extra server-side computation to partition models into subsets, it helps alleviate network congestion and reduces client start-up time.

Simplified model: Tailored for "thin" client scenarios, this methodology (outlined in Levoy [23]) involves sending a simplified version of the model along with an image that captures the difference between the simplified and original models. The client renders the simplified model and then integrates the differential image into the final display. While this approach minimizes client-side processing and reduces network bandwidth for model streaming while preserving high rendering quality, it imposes a heavier computational load on the server, which must generate the simplified model, render both versions, and compute the differential image.

Point cloud: This variant of the simplified model approach focuses on generating a point cloud representation of the original model, as introduced by Duguet and Drettakis [24]. Designed specifically for mobile clients with smaller display screens, it addresses the issue that the polygon count of detailed meshes can far exceed the number of available display pixels. In contrast, the server-generated point cloud is scaled to the resolution of the display, making it significantly more manageable for streaming and rendering [25].

2.2.2 Image-based approach

Image-based remote rendering systems operate by rendering all 3D models on the server and transmitting the resulting images to the client. Unlike model-based approaches, this strategy eliminates the need for the client to possess dedicated 3D graphics hardware.

Image impostor. The image impostor technique is widely adopted in remote rendering systems. In this method,

the server renders all 3D content and sends the outcomes as 2D images to the client. The client's role is simply to display these images and forward user interactions back to the server. Because this approach only requires a resolution that matches the client display, the network bandwidth needed for streaming is relatively low, regardless of the complexity of the original 3D models. Furthermore, advanced image and video compression techniques can be applied to further reduce bandwidth usage. The primary drawback of this method is interaction latency.

Environment map. Environment mapping, extensively used in 3D game development to streamline the rendering of distant background elements, can also be leveraged to reduce interaction latency in virtual environment rendering. In this approach, the server generates a panoramic environment map, i.e., a 360-degree view of the scene, based on the current viewpoint. The client then projects this map into a standard view, allowing users to pan the virtual camera without immediate server involvement. Consequently, the latency for panning is limited to the client-side projection time. However, if the user changes the viewpoint position significantly, the client must wait for the server to deliver an updated environment map.

Depth image. In the depth image method, the server renders the 3D models and produces a depth image comprising both a color map and a depth map. When the rendering viewpoint remains unchanged, the client can directly display the color map. If the viewpoint shifts, the client employs 3D image warping (McMillan 1997) [26] on the received depth image to instantly synthesize a new view. This approach can be seen as a simplified version of the point cloud method, where each pixel in the depth image represents a 3D point. The key advantage is that 3D image warping requires significantly less computational power than rendering a full point cloud via a comprehensive 3D graphics pipeline. This technique has been applied in various systems, including those by Chang and Ger (2002) [27], Bao and Gourlay (2004) [28], Smit et al. (2009) [29], and Zhu et al. (2011) [30].

3. SYSTEM DESIGN

In this section, we detail the architecture of our proposed real-time remote rendering system (Fig. 1), meticulously engineered to deliver low latency. Building on our previous analysis, the 'Depth Image' approach stands out as particularly advantageous compared to alternative methods. However, its average response time is largely determined by the ability of Vsingle to accurately capture and represent all user movements.

3.1 3D image warping

3D image warping is a well-established image-based rendering technique first introduced by McMillan and Bishop [31]. This algorithm takes as input a depth image, which comprises both a color map and a depth map, along with the original viewpoint and a desired target viewpoint. It then generates a color image corresponding to the target viewpoint by calculating, for each pixel (u_1, v_1) in the input, its new coordinates (u_2, v_2) and reassigning the pixel's color accordingly.

A common side effect of this process is the creation of "hole" artifacts, since the output image can only incorporate pixels that exist in the input image. These artifacts generally fall into two categories. The first, occlusion exposure, occurs when objects that were previously hidden or outside the initial view become visible in the warped image. The second type arises from inadequate sampling, typically on surfaces with varying gradients, resulting in gaps in the synthesized view.

To address these challenges, numerous hole-filling techniques have been developed. Methods such as depth filter and 'Splats' effectively fill small gaps caused by undersampling, while approaches like super-view warping and wide field-of-view warping are beneficial for handling occlusion-related artifacts. Moreover, strategies such as multiple references, view compensation, and Layered Depth Image (LDI) utilize images from different viewpoints to correct distortions introduced during warping, thereby addressing both categories of hole artifacts.

In our remote rendering system, we have adopted the multiple references approach to mitigate warping artifacts. This technique was chosen because it avoids the offline processing required by LDI and circumvents the need for continuous interactive communication, as seen in viewpoint compensation methods.

3.2 Multi-depth image

We present our multi-depth image design, a remote rendering system engineered to generate multiple depth images that enable mobile clients to render interactive graphics in real time. As illustrated in the accompanying diagram, our system operates as follows:

On the server side, a reference viewpoint selection module identifies a set of viewpoints, denoted as $\{ref_k\}$. In addition to rendering the source 3D data at the primary reference viewpoint (ref_0), the rendering engine produces renderings for every viewpoint in the set $\{ref_k\}$, yielding a collection of depth images.

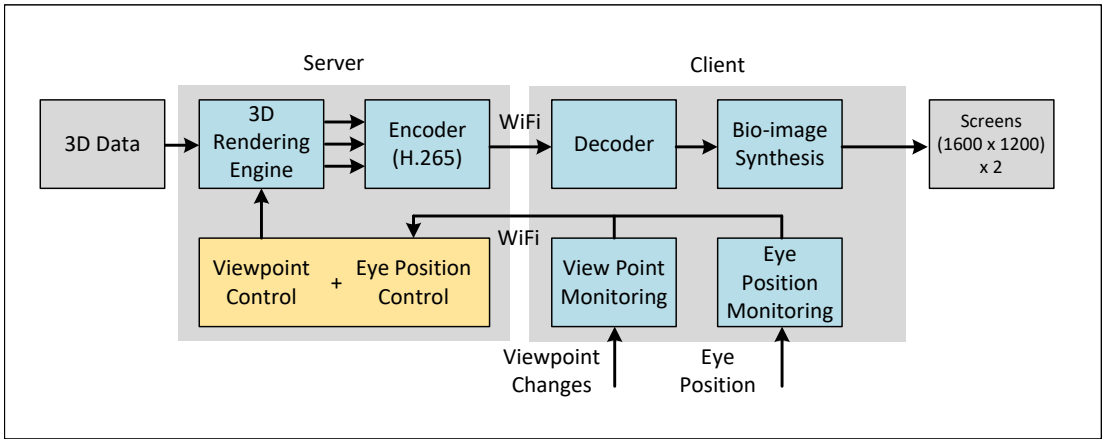

Figure 1 – System architecture.

On the client side, when there is no user interaction or when the current viewpoint (V_C) coincides with ref_0, the corresponding image (I) is displayed directly. However, when the user changes the viewpoint, the client warps each received depth image to align with the new viewpoint and then combines these warped images to generate the final display image. The effectiveness of this warping process, particularly its ability to minimize gaps or inconsistencies, depends critically on the appropriate selection of reference viewpoints. Our model analysis confirms that the proposed system meets the design requirements, maintaining a quality metric (Q) of 0. Although warping multiple depth images introduces extra computation on the client side, the overall processing time (T) scales with the display resolution rather than with the complexity of the source 3D models.

The probability p associated with V_{multi}, where V denotes the cardinality of the viewpoint set, depends largely on how the reference viewpoints $\{ref_k\}$ are selected. Thus, minimizing p to reduce latency transforms into the challenge of maximizing the effective coverage provided by these viewpoints. Unlike previous systems that use multiple reference viewpoints to replace full 3D graphics rendering, selecting these viewpoints after observing changes in the user's perspective to maximize warping quality, our system is designed to minimize interaction latency by proactively choosing reference viewpoints before any changes occur.

Our selection criteria are designed to maximize coverage across the viewing space while ensuring sufficient warping quality for the covered viewpoints. Minor artifacts are acceptable since the warped image is displayed only briefly until updated renderings from the server are received.

Compared to other image-based remote rendering approaches, our system requires greater network bandwidth due to the transmission of multiple depth images. While this introduces a higher data load than single-image methods such as image impostor, we mitigate this overhead by applying standard compression algorithms, JPEG for color images and ZLIB for depth data. The trade-off, however, is deliberate: the system is optimized for significantly reduced interaction latency, which is critical for AR/MR applications requiring real-time feedback. We further discuss this trade-off and provide quantitative comparisons in the evaluation section.

4. REFERENCE VIEWPOINT SELECTION

The main challenge in our system is the strategic selection of reference viewpoints to maximize coverage. A rendering viewpoint is defined by three vectors: the camera's position, its viewing direction, and its upward orientation. Instead of considering every possible combination of these vectors, we concentrate only on the viewpoints that mobile users are likely to adopt. Based on our application analysis, we introduce two simplifying assumptions: (1) changes in the rendering viewpoint due to user movement follow specific, predictable patterns, and (2) these changes occur in discrete steps and are limited in distance. These assumptions significantly reduce the search space, transforming the reference selection problem into a unidimensional task. Rather than selecting a large number of viewpoints to cover all possibilities, we choose a single reference viewpoint for each motion pattern, ensuring complete coverage along each trajectory.

In one case, two reference viewpoints are chosen to cover the range of potential viewpoints along an orbital path. It is important to note that even motions with the same general pattern but different directions are treated as distinct, necessitating separate reference viewpoints. In contrast, another scenario depicts a video game where the player maneuvers a tank on a battlefield to engage adversaries.

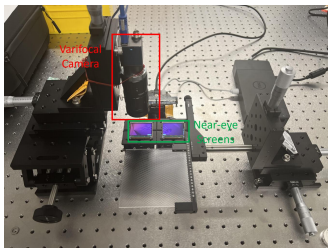

Figure 2 – Hardware design of prototype.

5. EVALUATION

5.1 Experiment setup

To evaluate the proposed remote rendering system, we implemented our design across three platforms: an Android tablet, a Meta Quest 3 headset, and a custom-built prototype AR device. The **server** runs on a workstation equipped with an AMD 5950X CPU, 32 GB of RAM, an Nvidia RTX 3060Ti GPU, and a 5 GHz WiFi connection. It supports three rendering modes:

- **Image impostor based:** Only a 2D color image of the current viewpoint is generated and sent to the client.
- **Depth image based:** A depth image, consisting of both color and depth maps, is rendered for the current viewpoint.
- **Multi-depth image based (ours):** The server generates depth images for multiple reference viewpoints. These are selected via either the full search algorithm or the reference prediction algorithm to enable real-time client-side warping.

Client-side deployments span the following platforms:

- **3D tablet:** We use a Lume Pad 2 [32] in our evaluation, which features a Qualcomm Snapdragon 888 SoC, 8 GB of RAM, and a 12.4-inch display. The tablet supports both conventional 2D and immersive 3D viewing through Leia's proprietary Diffractive Lightfield Backlighting (DLB) technology. In 3D mode, the effective per-eye resolution is approximately 1280×800. For network performance analysis, the device is connected to the server via both 5 GHz WiFi.
- **Meta Quest 3 [33]:** A standalone AR/MR headset built on the Qualcomm Snapdragon XR2 Gen 2 platform. The client application is integrated into the Unity-based Android runtime and communicates with the server over 5 GHz WiFi to assess immersive interaction performance.
- **Prototype device:** A purpose-built AR headset powered by an NVIDIA Jetson Orin Nano module [34]. It features dual displays, each with a resolution of 1600×1200 pixels. The device connects wirelessly via 5 GHz WiFi and is used to evaluate high-resolution stereoscopic rendering on embedded hardware.

The 3D test models are selected from the Stanford 3D scanning repository, including: **Bunny** (35,947 vertices, 69,451 triangles), **Happy Buddha** (543,652 vertices, 1,087,716 triangles), and **Dragon** (566,098 vertices, 1,132,830 triangles). Users perform leftward and rightward orbital motions, which prompt the generation of two reference depth images under the multi-depth mode.

Rendering is conducted at three resolutions **2560×800 pixels** (Lume Pad 2), **3200×1200 pixels** (Prototype), and **4128×2208 pixels** (Meta Quest 3). Color images are compressed using the **JPEG** algorithm, and depth maps are compressed using **ZLIB**. Communication between the server and clients is handled via the **TCP** protocol to ensure reliable data delivery.

5.2 Evaluation metrics

To assess the performance and effectiveness of the proposed remote rendering system, we utilize the following evaluation metrics:

- **Frames Per Second (FPS):** This metric quantifies the rendering throughput on the client side. FPS reflects the number of complete frames the system can render and display per second, which directly impacts the visual fluidity of the AR/MR experience. We report both average FPS and its variations under static and interactive conditions.
- **End-to-end latency:** This measures the total delay between a user's interaction (e.g., head movement) and the corresponding visual update on the display. It includes server processing, image transmission, decoding, and client-side rendering. Latency is recorded using synchronized timestamps embedded in the client-server communication cycle.
- **Structural Similarity Index (SSIM):** SSIM is a perceptual metric that evaluates the visual similarity between the rendered image on the client and the reference ground-truth image rendered on the server. It jointly considers luminance, contrast, and structural information. The SSIM between two images x and y is computed as:

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (1)$$

  where μ_x and μ_y are the mean intensities, σ_x^2 and σ_y^2 are the variances, σ_{xy} is the covariance of the two images, and C_1, C_2 are small constants to stabilize the division. We use $L = 255$, $K_1 = 0.01$, and $K_2 = 0.03$ in our implementation.
- **Bandwidth usage:** This metric captures the total amount of data transmitted from the server to the client, including both color and depth images. We measure average bandwidth using different methods.

5.3 Performance with different rendering modes

We evaluate system performance under two user interaction scenarios: (1) a *stationary viewpoint*, where the user occasionally adjusts the viewing angle, and (2) *dynamic viewpoint movement*, where the user actively explores the scene. Experiments are conducted across three platforms using three rendering modes. The evaluation metrics include Frames Per Second (FPS), end-to-end latency, structural similarity index (SSIM), and bandwidth usage.

Abbreviations used in tables:

- **Platforms:**
  - **LP2** – Lume Pad 2 tablet (Snapdragon 888, 12.4" DLB display)
  - **MQ3** – Meta Quest 3 headset (Snapdragon XR2 Gen 2)
  - **PRO** – Custom prototype device (Jetson Orin Nano, dual 1600×1200 displays)
- **Rendering modes:**
  - **IMP** – Image impostor
  - **DEP** – Depth image
  - **MDEP (ours)** – Multi-depth image

Table 1 – Performance across platforms (stationary viewpoint)

Platform	Mode	FPS	Latency (ms)	SSIM	BW (MB/s)
LP2	IMP	27	155	0.82	1.6
	DEP	23	100	0.88	3.9
	MDEP (Ours)	25	65	0.93	4.5
MQ3	IMP	29	145	0.84	1.4
	DEP	25	90	0.90	3.6
	MDEP (Ours)	26	58	0.95	4.4
PRO	IMP	24	160	0.83	1.7
	DEP	22	98	0.87	4.0
	MDEP (Ours)	25	62	0.94	4.7

Table 2 – Performance across platforms (dynamic viewpoint movement)

Platform	Mode	FPS	Latency (ms)	SSIM	BW (MB/s)
LP2	IMP	14	250	0.77	2.2
	DEP	17	130	0.84	4.3
	MDEP (Ours)	21	80	0.91	5.1
MQ3	IMP	16	230	0.79	2.0
	DEP	19	115	0.86	4.1
	MDEP (Ours)	23	70	0.93	5.0
PRO	IMP	15	240	0.78	2.1
	DEP	18	122	0.85	4.5
	MDEP (Ours)	22	75	0.92	5.3

Tables 1 and 2 present the system's performance across platforms under stationary and dynamic interaction scenarios. In the stationary case, all modes maintain acceptable frame rates, with Image impostor (IMP) achieving the highest FPS due to minimal client-side processing. However, IMP suffers from high latency, as every viewpoint change requires a server round-trip. Depth image (DEP) improves latency via local warping but consumes more bandwidth. Multi-Depth image (MDEP) delivers the best trade-off: latency as low as 58 ms, SSIM above 0.93, and FPS comparable to other modes across all platforms.

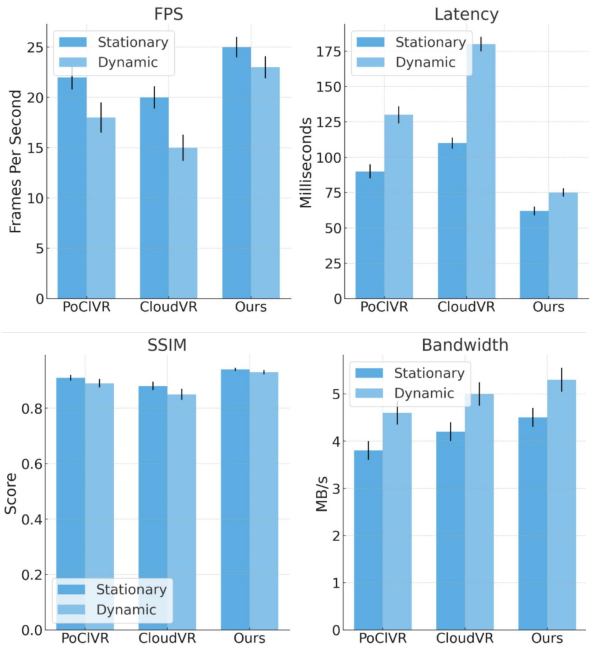

Figure 3 – Comparison with SOTA method across FPS, latency, SSIM, and bandwidth under stationary and dynamic conditions.

In dynamic scenarios, the benefits of MDEP are more prominent. IMP's latency sharply increases (up to 250 ms on LP2), while DEP improves performance but still suffers under rapid viewpoint changes. MDEP maintains interactive frame rates (21–23 FPS), reduced latency (70–80 ms), and high SSIM (≥ 0.91) across all devices. These results confirm that MDEP is the most effective strategy for high-quality, responsive rendering in real-time AR/MR systems under both light and intensive user interaction.

5.4 Visualization of method comparison

To enhance the interpretability of performance differences among rendering techniques, we compares three representative methods: **PoCIVR** [35], **CloudVR** [36], and **multi-depth (ours)**. The comparison includes four key performance metrics, Frames Per Second, end-to-end latency, structural similarity index (SSIM), and bandwidth usage, evaluated under both *stationary* and *dynamic* user interaction scenarios.

As shown in Fig. 3, our multi-depth rendering method consistently outperforms the two baselines across all four metrics. In the **FPS** plot, our method achieves the highest frame rate, especially under dynamic conditions, while maintaining low variance. In terms of **latency**, it shows a significant reduction, approximately 30–100 ms lower than competitors, demonstrating its suitability for real-time interaction.

The **SSIM** results reveal superior visual quality, with our method reaching the highest average similarity scores and the least degradation under movement. Finally,

although our approach requires moderately higher **bandwidth**, it remains within a practical range for modern wireless systems, and this trade-off is justified by its improvements in responsiveness and rendering quality.

Overall, our method offers the most favorable balance of visual fidelity, latency, and performance robustness across varying conditions and platforms.

6. CONCLUSION

We presented a real-time edge-assisted rendering system for mobile AR/MR platforms, introducing a novel multi-depth image approach to reduce interaction latency while maintaining high rendering quality. Unlike traditional remote rendering methods, our system pre-renders multiple depth images and enables client-side 3D warping, significantly reducing reliance on server feedback. Our design includes a warping-aware reference viewpoint selection strategy and is evaluated across multiple devices and scenarios. Experimental results demonstrate substantial improvements in latency, FPS, and SSIM compared to both conventional baselines and state-of-the-art methods. This system provides a practical foundation for future deployment in collaborative multi-user AR environments and scalable edge-cloud rendering frameworks.

REFERENCES

[1] Jacob Chakareski. "VR/AR immersive communication: Caching, edge computing, and transmission trade-offs". In: *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. 2017, pp. 36–41.

[2] Mingrui Yin, Sohom Sen, Yongjie Guan, Xueyu Hou, Tao Han, and Nirwan Ansari. "Towards Immersive Metaverse Experience: A Wireless Adaptive 3D Human Modeling System". In: *IEEE Network* (2025).

[3] Mingrui Yin, Sohom Sen, Yongjie Guan, Jing Du, Xueyu Hou, and Tao Han. "Health-MR: A Mixed Reality-Based Patient Registration and Monitor Medical System". In: *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 2024, pp. 2113–2119.

[4] George Papagiannakis, Gurminder Singh, and Nadia Magnenat-Thalmann. "A survey of mobile and wireless technologies for augmented reality systems". In: *Computer Animation and Virtual Worlds* 19.1 (2008), pp. 3–22.

[5] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. "DeepMix: mobility-aware, lightweight, and hybrid 3D object detection for headsets". In: *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 2022, pp. 28–41.

[6] Xiuquan Qiao, Pei Ren, Schahram Dustdar, Ling Liu, Huadong Ma, and Junliang Chen. "Web AR: A promising future for mobile augmented reality—State of the art, challenges, and insights". In: *Proceedings of the IEEE* 107.4 (2019), pp. 651–666.

[7] Xueyu Hou, Yongjie Guan, Tao Han, and Ning Zhang. "Distredge: Speeding up convolutional neural network inference on distributed edge devices". In: *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2022, pp. 1097–1107.

[8] Xueyu Hou, Yongjie Guan, and Tao Han. "NeuLens: spatial-based dynamic acceleration of convolutional neural networks on edge". In: *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 2022, pp. 186–199.

[9] Zenan Li, Xiaoxing Ma, Chang Xu, Chun Cao, Jingwei Xu, and Jian Lü. "Boosting operational dnn testing efficiency through conditioning". In: *Proceedings of the 2019 27th ACM Joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 2019, pp. 499–509.

[10] Xueyu Hou, Yongjie Guan, Tao Han, and Cong Wang. "Towards real-time embodied AI agent: A bionic visual encoding framework for mobile robotics". In: *International Journal of Intelligent Robotics and Applications* (2024), pp. 1–19.

[11] Xueyu Hou and Tao Han. "TrustServing: A quality inspection sampling approach for remote DNN services". In: *2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE. 2020, pp. 1–9.

[12] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. "Cloud computing: An overview". In: *IEEE international conference on cloud computing*. Springer. 2009, pp. 626–631.

[13] Xueyu Hou, Yongjie Guan, and Tao Han. "Dystri: A dynamic inference based distributed dnn service framework on edge". In: *Proceedings of the 52nd International Conference on Parallel Processing*. 2023, pp. 625–634.

[14] Xueyu Hou, Yongjie Guan, Nakjung Choi, and Tao Han. "BPS: Batching, Pipelining, Surgeon of Continuous Deep Inference on Collaborative Edge Intelligence". In: *IEEE Transactions on Cloud Computing* (2024).

[15] Zhuo Wang, Larry Millet, Mustafa Mir, Huafeng Ding, Sakulsuk Unarunotai, John Rogers, Martha U Gillette, and Gabriel Popescu. "Spatial light interference microscopy (SLIM)". In: *Optics express* 19.2 (2011), pp. 1016–1026.

[16] Bo Chen, Harry H Cheng, and Joe Palen. "Mobile-C: a mobile agent platform for mobile C/C++ agents". In: *Software: Practice and Experience* 36.15 (2006), pp. 1711–1733.

[17] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R Wood, and Andy Hopper. "Virtual network computing". In: *IEEE Internet Computing* 2.1 (1998), pp. 33–38.

[18] Bonnie L Maidak, James R Cole, Timothy G Lilburn, Charles T Parker Jr, Paul R Saxman, Jason M Stredwick, George M Garrity, Bing Li, Gary J Olsen, Sakti Pramanik, et al. "The RDP (ribosomal database project) continues". In: *Nucleic acids research* 28.1 (2000), pp. 173–174.

[19] Lien Deboosere, Jeroen De Wachter, Pieter Simoens, Filip De Turck, Bart Dhoedt, and Piet Demeester. "Thin client computing solutions in low-and high-motion scenarios". In: *International Conference on Networking and Services (ICNS'07)*. IEEE. 2007, pp. 38–38.

[20] S Myriokefalitakis, K Tsigaridis, N Mihalopoulos, J Sciare, Athanasios Nenes, K Kawamura, A Segers, and M Kanakidou. "In-cloud oxalate formation in the global troposphere: a 3-D modeling study". In: *Atmospheric Chemistry and Physics* 11.12 (2011), pp. 5761–5782.

[21] Yana Dimova, Gunes Acar, Lukasz Olejnik, Wouter Joosen, and Tom Van Goethem. "The cname of the game: Large-scale analysis of dns-based tracking evasion". In: *arXiv preprint arXiv:2102.09301* (2021).

[22] Leonel Merino, Magdalena Schwarzl, Matthias Kraus, Michael Sedlmair, Dieter Schmalstieg, and Daniel Weiskopf. "Evaluating mixed and augmented reality: A systematic literature review (2009-2019)". In: *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2020, pp. 438–451.

[23] Marc Levoy. "Display of surfaces from volume data". In: *IEEE Computer graphics and Applications* 8.3 (1988), pp. 29–37.

[24] F. Duguet and G. Drettakis. "Flexible point-based rendering on mobile devices". In: *IEEE Computer Graphics and Applications* 24.4 (2004), pp. 57–63. DOI: 10.1109/MCG.2004.5.

[25] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. "Metastream: Live volumetric content capture, creation, delivery, and rendering in real time". In: *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 2023, pp. 1–15.

[26] William R. Mark, Leonard McMillan, and Gary Bishop. "Post-rendering 3D warping". In: *Proceedings of the 1997 Symposium on Interactive 3D Graphics*. I3D '97. Providence, Rhode Island, USA: Association for Computing Machinery, 1997, 7–ff. ISBN: 0897918843. DOI: 10.1145/253284.253292. URL: https://doi.org/10.1145/253284.253292.

[27] Chun-Fa Chang and Shyh-Haur Ger. "Enhancing 3D Graphics on Mobile Devices by Image-Based Rendering". In: *Advances in Multimedia Information Processing — PCM 2002*. Ed. by Yung-Chang Chen, Long-Wen Chang, and Chiou-Ting Hsu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 1105–1111. ISBN: 978-3-540-36228-9.

[28] P. Bao and D. Gourlay. "A framework for remote rendering of 3-D scenes on limited mobile devices". In: *IEEE Transactions on Multimedia* 8.2 (2006), pp. 382–389. DOI: 10.1109/TMM.2005.864337.

[29] L. Scheunemann, D. Balzani, D. Brands, and J. Schröder. "Design of 3D statistically similar Representative Volume Elements based on Minkowski functionals". In: *Mechanics of Materials* 90 (2015). Proceedings of the IUTAM Symposium on Micromechanics of Defects in Solids, pp. 185–201. ISSN: 0167-6636. DOI: https://doi.org/10.1016/j.mechmat.2015.03.005. URL: https://www.sciencedirect.com/science/article/pii/S0167663615000745.

[30] Cheng Zhu, T Yong-Jin Han, Eric B Duoss, Alexandra M Golobic, Joshua D Kuntz, Christopher M Spadaccini, and Marcus A Worsley. "Highly compressible 3D periodic graphene aerogel microlattices". In: *Nature communications* 6.1 (2015), p. 6962.

[31] Leonard McMillan and Gary Bishop. "Plenoptic modeling: An image-based rendering system". In: (2023), pp. 433–440.

[32] Leia Inc. *Lume Pad 2 — World's First 3D Lightfield Tablet*. https://www.leiainc.com/lume-pad-2. Accessed: 2025-04-23. 2023.

[33] Meta. *Meta Quest 3 – Mixed Reality Headset*. https://www.meta.com/quest/quest-3. Accessed: 2025-04-23. 2023.

[34] NVIDIA. *Jetson Orin Nano Module*. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin-nano/. Accessed: 2025-04-23. 2023.

[35] Ximing Wu, Kongyange Zhao, Xu Chen, and Teng Liang. "Edge-assisted Real-time Dynamic 3D Point Cloud Rendering for Multi-party Mobile Virtual Reality". In: *Proceedings of the 32nd ACM International Conference on Multimedia*. 2024, pp. 2824–2832.

[36] Teemu Kämäräinen, Matti Siekkinen, Jukka Eerikäinen, and Antti Ylä-Jääski. "CloudVR: Cloud accelerated interactive mobile virtual reality". In: *Proceedings of the 26th ACM international conference on Multimedia*. 2018, pp. 1181–1189.

AUTHORS

Bishakha Rani Biswas is a Ph.D. student in the Department of Electrical and Computer Engineering at the University of Maine. Her current research interests lie in real-time, human-interactive systems powered by artificial intelligence, including large language models and neural network–based 3D rendering.

Xueyu Hou is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Maine. Her research interests include distributed computing, efficient artificial intelligence, and real-time systems. Her current work focuses on real-time text streaming using large language models, diffusion-based robotic control, and resource-efficient computing in cyber-physical systems.

Yongjie Guan is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Maine. His research interests include augmented and virtual reality, edge computing, cloud systems, and wireless communication. He has authored over 15 publications in top-tier conferences and journals. His current work focuses on multi-user AR/VR systems and the development of efficient edge-based infrastructure for real-time sensing, communication, and rendering in immersive environments.