InstantXR: Instant XR Environment on the Web Using Hybrid Rendering of Cloud-based NeRF with 3D Assets

Moonsik Park*
ESTsoft Corporation
Seoul, Korea
moonsik.park@estsoft.com

Byounghyun Yoo†
Korea Institute of Science and Technology
Seoul, Korea
University of Science and Technology
Seoul, Korea
yoo@byoo.net

Jee Young Moon
Korea Institute of Science and Technology
Seoul, Korea
jeeyoung.moon@wrl.onl

Ji Hyun Seo
Korea Institute of Science and Technology
Seoul, Korea
Korea University
Seoul, Korea
jihyun.seo@wrl.onl

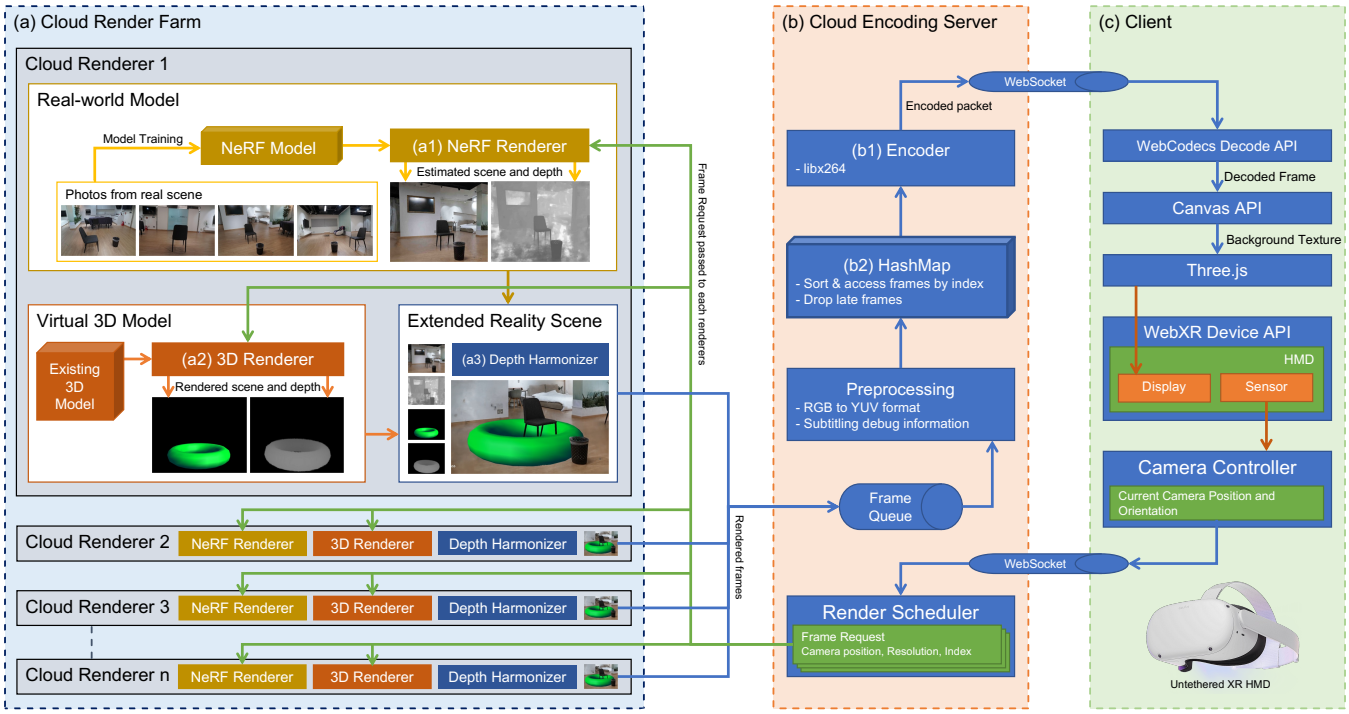

Figure 1: Overview of the proposed instant XR environment.

ABSTRACT

For an XR environment to be used on a real-life task, it is crucial all the contents are created and delivered when we want, where we want, and most importantly, on time. To deliver an XR environment faster and correctly, the time spent on modeling should be considerably reduced or eliminated. In this paper, we propose a hybrid method that fuses the conventional method of rendering 3D assets with the Neural Radiance Fields (NeRF) technology, which uses photographs to create and display an instantly generated XR environment in real-time, without a modeling process. While NeRF

*Work done primarily as an intern at the Korea Institute of Science and Technology.
†Corresponding author: Byounghyun Yoo (yoo@byoo.net).

Web3D '22, November 2–4, 2022, Evry-Courcouronnes, France

ACM ISBN 978-1-4503-9914-2/22/11.
https://doi.org/10.1145/3564533.3564565

can generate a relatively realistic space without human supervision, it has disadvantages owing to its high computational complexity. We propose a cloud-based distributed acceleration architecture to reduce computational latency. Furthermore, we implemented an XR streaming structure that can process the input from an XR device in real-time. Consequently, our proposed hybrid method for real-time XR generation using NeRF and 3D graphics is available for light-weight mobile XR clients, such as untethered HMDs. The proposed technology makes it possible to quickly virtualize one location and deliver it to another remote location, thus making virtual sightseeing and remote collaboration more accessible to the public. The implementation of our proposed architecture along with the demo video is available at https://moonsikpark.github.io/instantxr/.

CCS CONCEPTS

• **Human-centered computing** → *Collaborative and social computing systems and tools*; **Mixed / augmented reality**; **Virtual reality**; **Web-based interaction**.

KEYWORDS

Extended reality, XR, Neural rendering, Neural radiance fields, NeRF, Deep learning, Cloud computing, Virtual reality, Web-based XR

ACM Reference Format:
Moonsik Park, Byounghyun Yoo, Jee Young Moon, and Ji Hyun Seo. 2022. InstantXR: Instant XR Environment on the Web Using Hybrid Rendering of Cloud-based NeRF with 3D Assets. In *The 27th International Conference on 3D Web Technology (Web3D '22), November 2–4, 2022, Evry-Courcouronnes, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3564533.3564565

1 INTRODUCTION

The success of VR, AR, and MR technologies has broken the boundaries between each area, resulting in increased demand for extended reality (XR) technologies. In recent years, the COVID-19 pandemic isolation, together with the excitement of realization of the Metaverse, has increased the public's desire to experience various locations remotely from a remote area. The demand for immersive XR technologies for remote collaboration is especially increasing. XR technology for remote collaboration between a worksite and remote location requires the provision of the virtual environment of the worksite to the remote user. Although preliminary modeling is imperative for creating a mirror world for an actual worksite, modeling is notorious for its time-consuming process. In other words, a method for reducing time consumption is essential to instantly provide an XR environment in a mirrored space.

Therefore, in creating a mirror world, this study replaces the conventional preliminary modeling method with Neural Radiance Fields (NeRF), which can generate images from an arbitrary perspective by applying a deep learning model that processes several photos of the actual work site. The results generated by the NeRF were extremely photorealistic [Mildenhall et al. 2020]. Despite its exceptional quality in visualizing a static environment, NeRF is not appropriate for creating a dynamic 3D object that is necessary for an XR environment. Our method overcomes this shortcoming by incorporating a conventional 3D graphics rendering pipeline for rendering dynamic 3D assets. Simply, this paper presents a hybrid method that generates the static part of the XR environment using NeRF while generating the dynamic part using the conventional 3D graphics rendering pipeline. We removed the previously required preliminary modeling process for the XR environment composition using the architecture shown in Figure 1. To the best of our knowledge, this is the first attempt to apply NeRF to untethered XR HMD. The contributions of this study are as follows:

(1) Eliminating preliminary modeling process for the XR environment using Neural Radiance Fields implementation [Müller et al. 2022]
(2) Implementing real-time cloud-based NeRF scene generation
(3) Overcoming the limitation of NeRF through harmonization with conventional 3D graphics rendering through the depth test of the 3D scene
(4) Real-time cloud-based XR streaming to untethered HMD

2 RELATED WORK

The implementation of the XR environment requires the following representative elements: content, platform, network, and device. Continuous content supply requires simplification and automation of the modeling process. Particularly, in an application domain that virtually mirrors the environment in reality to the XR environment, such as remote tourism and collaboration [Lee and Yoo 2021], the transformation time for digitalizing the real space into a virtual model is of utmost importance. Although conventional XR content production pipelines scan the real-world environment or reconstruct a 3D model using photogrammetry, they are difficult to use in instant collaboration because of the long preparation time.

To reduce the time required for the 3D reconstruction of the XR environment, an approach that skips the reconstruction process and uses point cloud rendering after real-time scanning of the site with LiDAR has been investigated [Lee et al. 2021]. However, the image quality of the point cloud is lower than that of the conventional mesh-based reconstructed 3D model with texture because of the space between the point clouds, which lowers the user's situational awareness.

With the popularity of machine learning, attempts have been made to map 3D spatial locations and the implicit representation of an object's shape to the weight of a fully connected multilayer perceptron model [Curless and Levoy 1996]. However, these attempts worked only with objects with simple geometries, and the results were of poor quality compared with methods that did not leverage machine learning [Mildenhall et al. 2020].

NeRF [Mildenhall et al. 2020] used not only a 3D spatial location but also the viewing direction of a scene. The location and direction data are mapped to the volume density and view-dependent emitted radiance. The process can be differentiated because it is a fully connected function. The loss of the renderer can be propagated back to the input and applied to train the network. NeRF has become a novel view synthesis method comparable to handcrafted algorithms with adjustments and optimization. Being based on machine learning, NeRF has shown potential, which rapidly gained the interest of vision researchers.

While NeRF's view synthesis quality is far superior to that of other methods, the time and computing resources required to train

the model and render a scene are issues that block its application to the real-world use cases. There has been significant development in both the application and implementation of NeRF [Tewari et al. 2022]. Research began aiming to speed up and lower computing power requirements. FastNeRF [Garbin et al. 2021] can compute a scene thousands of times faster than the original NeRF by caching a deep radiance map at each position in space, and efficiently query the map using ray directions to estimate pixel values in the rendered image. NVLabs [Müller et al. 2022] was able to speed up NeRF using a multiresolution hash table of trainable feature vectors, whose values are optimized through stochastic gradient descent.

Some research focused on using NeRF in domains other than novel-view synthesis. BlockNeRF [Tancik et al. 2022] applies NeRF to a real-world environment as large as the city blocks. HyperNeRF [Park et al. 2021] attempts to render a novel view of a moving object by adding another dimension representing time to a model. Nerf in the Dark [Mildenhall et al. 2022] uses NeRF to denoise images captured in low-light environments.

The major difference between structure-from-motion (SfM) and NeRF is that NeRF can generate volumetric representations while SfM generates a triangulated mesh of the scene. While there is research done to map a large location with SfM, [Schönberger et al. 2016] in the current state the triangulated mesh of SfM cannot match the volumetric output of NeRF in terms of photographic realism. The problem with NeRF is that although we can get the scene as an RGB image and depth, it cannot reconstruct objects in the scene. There is potential, though, that if we run SfM with NeRF's output, we could possibly get both the photorealistic output and the 3D reconstruction of the scene [Condorelli et al. 2021].

We define cloud-based real-time rendering as doing compute-intensive workloads on a remote server and sending the result artifacts to the client as a video stream. The difficulty in designing these systems lies in the management of latency. In cloud-based real-time rendering, the latency can occur at four points. User input should be sent to the server when rendering a scene. The server should render a scene based on input. The scene needs to be transported back to the client. The client has to decode the video stream and render it to the user's screen. It is nearly impossible to implement cloud-based real-time rendering because it is difficult to manage the latency to the level compared to local rendering. However, recent advances in Internet speed and quality have boosted commercial efforts to create a remote gaming service. Sony Corporation released PS Now in 2014, followed by GeForce Now by NVIDIA, and Stadia by Google [Xu and Claypool 2021].

3 PROBLEM STATEMENT

The original NeRF implementation was built using Tensorflow in Python, which slowed down the implementation. It took 1-2 days to learn a scene and approximately 30 s to render a novel view of 800×800 resolution [Mildenhall et al. 2020], which is not ideal for real-time rendering.

NVlabs successfully sped up NeRF using multiresolution hash encoding, which achieved several orders of magnitude faster training and rendering speed [Müller et al. 2022]. We chose this implementation (henceforth referred to as instant-ngp) for our method because of its fast speed based on its implementation in C++. By using this implementation, it took less than a second to train the network and less than 30 milliseconds (ms) to render a novel view, which is a significant improvement compared to the original NeRF implementation, which takes two days to train the network and 30 s to render a novel view with the same dataset.

While instant-ngp is faster than most NeRF implementations, it is not sufficient to be used for our purpose, which renders NeRF in a real-time immersive XR environment. Research shows that users find the motion-to-photon (MTP) latency disturbance when the latency is over 23 ms, while in a situation where the viewpoint is violently moving at 80 °/s [Yang et al. 2019]. To provide a smoother experience to the user navigating the mirror world generated by NeRF, we should have the capability to render at least 44 frames per second per eye (a total of 88 frames per second for stereographic display; 11 ms per frame) without considering the sensor, transport, and any other latency.

DIBR is a method to render a different view for a scene with RGB image and depth [Schmeing 2011]. It is commonly used to render a 'target view' image for the other eye when only one input 'reference view' image is available. Our approach, in contrast to DIBR, is a two-pass rendering method where the renderer renders two distinct images for both eyes using their own position of view. Using DIBR would be faster because it only requires one source image as a reference view to be rendered from the network and rendering the target view for another eye with DIBR is faster than rendering another scene from NeRF. However, its fundamental limitation of not knowing the areas not visible in the reference view creates a loss in the target view. There have been methods to overcome this limitation [Azzari et al. 2010], but its implementation is outside the scope of our architecture. Because recent devices that target 3D use two pass rendering for each eye and we expect NeRF rendering to get significantly faster, we did not use DIBR.

To replace the 3D modeling task of a mirror world as an XR environment with a novel view rendered by NeRF, conventional 3D assets should be harmonized with the scene. Because NeRF can estimate the volume density along with the view-dependent emitted radiance, we used this density estimation to place conventional 3D assets inside a view rendered by NeRF.

An overview of the designed system is in Figure 1. We implemented a cloud renderer (a) that can provide a view upon request from the cloud encoding server (b). Client (c) is a web application that fully complies with web standards.

The client uses the WebXR device API [Jones et al. 2022] to continuously send the user's viewpoint to the cloud-encoding server. Given the user's viewpoint and the chosen resolution, the cloud encoding server sends a frame request to the cloud renderer. The renderer creates an extended reality scene using not only the scene but also the depth data, which are from both the volumetric rendering artifacts rendered by NeRF and artifact from a rendered 3D model. The cloud encoding server receives rendered extended reality scenes and places them in a frame queue. The server then preprocesses the frames by subtitling the debug information and converting their pixel format to be suitable for the encoder. Because the order of the frames should be strictly honored, despite its inefficiency, we used a hash map in sorting and selecting frames by index in a constant time. The frames were then fed to the encoder to generate an encoded video-packed stream. The cloud encoding

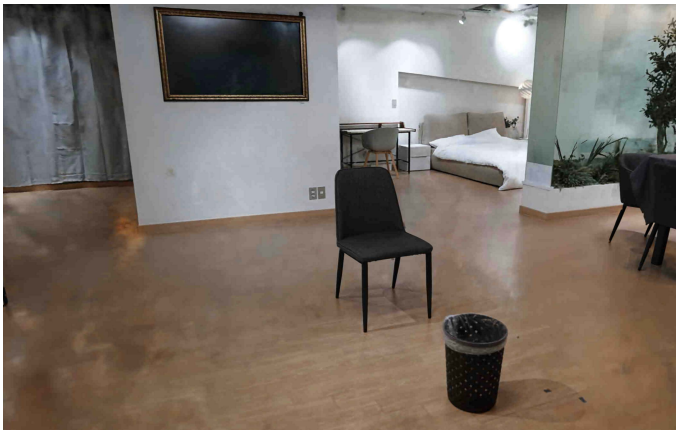

Figure 2: Output of instant-ngp after 10 min (100,000 iterations) of training the neural network.

Figure 3: One of the photographs used to train the network.

server can also use multiple cloud renderers to achieve a better frame rate and more continuous scenes.

Sending live processed frames in an untethered HMD is not a trivial task. We need to use a streaming protocol that can send packets as quickly as possible without introducing any additional delay. After much consideration, we implemented a novel video streaming method that sends raw packets to the client using WebSockets [WHATWG 2022]. This method is explained in detail in Section 4. The client then receives and decodes the video stream sent by the server. The video was rendered on a canvas and converted into a background texture. The texture is used by Three.js to display the video on each HMD display.

4 IMPLEMENTATION

4.1 Neural radiance fields

To create a mirror world with NeRF, we chose a laboratory that resembled an average household, called the Living Lab, located inside the Korea Institute of Science and Technology. This is because the Living Lab has a simple but realistic geometry and has previously been modeled in 3D, allowing us to compare NeRF and 3D reconstructed view in possible future work.

For the training dataset, we prepared 60 photos with a resolution of 4240 × 2832 taken from Sony α7S II at the Living Lab. Figure 3 shows an example of a photograph taken in the laboratory. We used COLMAP [Schönberger and Frahm 2016; Schönberger et al. 2016], an open-source structure-from-motion program to determine the viewing position and orientation of each photograph. Using this dataset, we trained the instant-ngp for 10 min. We terminated the training and used the trained model after 10 min and 100,000 iterations, after confirming that the loss rate did not improve. With the trained model, we could render Figure 2, a 1280 × 720 image in 80 ms on a computer with an NVIDIA A100 discrete GPU.

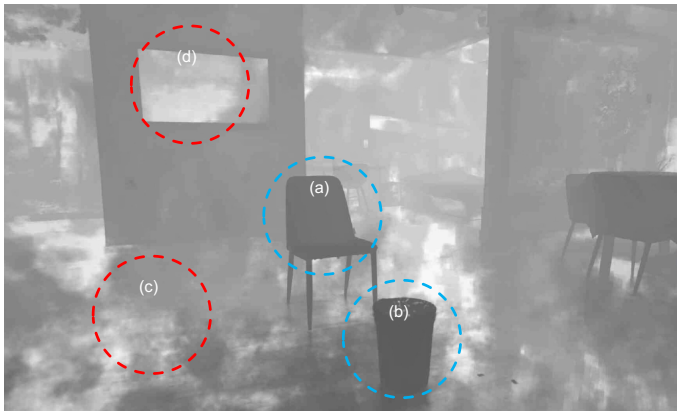

Figure 4: Depth estimation using NeRF.

NeRF estimates not only the scene but also the volume density of the view. Figure 4 shows a visualization of the volume density estimated using the NeRF. NeRF estimates the volume density for objects with a matte surface with irregular patterns and various features, such as regions (a) and (b). However, NeRF struggles to estimate the volume density of the floor, such as region (c), or the screen, such as region (d), which has fewer features and diffuse reflections.

4.2 Cloud-based NeRF

We set a goal in which the user should feel like they are inside a mirror world generated by NeRF. The entire technology should be web-based and web standards compliant such that the user does not need to install any program or add-ons. To achieve this, we should receive the viewing position and orientation from the client, render the view on a cloud rendering system, and send the rendered view back to the user's device. It is not sufficient to render only one view for a given viewpoint, because of the wide adaptation of stereoscopic displays in XR devices. Thus, it is important to render and stream two scenes that reflect parallax between the eyes.

For a smoother experience for the user inside the mirror world generated by the NeRF, it is better to generate as many frames as

possible. Although the rendering speed of the NeRF implementation we chose was relatively fast compared to other NeRF implementations, it was not sufficient to create the level of smoothness necessary for our purpose.

The limitation of computing power from a single unit will continue to exist despite the rapid increase in the computing power of discrete graphics processing units. If it is possible to distribute the rendering process across multiple machines, we can create a solution that is more powerful and cost-effective than using a single powerful unit. Alongside the readily available, time-sharable, and scalable resources provided by cloud computing vendors, the importance of distributed-rendering is increasing.

To accelerate the frame generation process, we built a cloud-based distributed NeRF rendering system. The system consisted of two components. A renderer that runs on multiple machines and an encoder that requests scenes based on user motion, then gather the scenes rendered by multiple renderers and encodes the scenes into a video stream.

To work properly, two factors should be considered for a cloud-rendering system based on neural rendering. First, the encoding speed between renderers should be as uniform as possible. The order of each frame is important for the video to be played in real-time. For instance, when the user is moving left, the frames should also move left until the user stops or changes direction. However, if the encoding speeds among the renderers are different, the encoder might have to wait for the frames to arrive or drop them entirely. If we wait for a frame, the frame generated by a fast encoder can become useless. If we ignore the order, there will be cases in which the user is moving to the left, but the video stream is unexpectedly transported to the right. Second, each frame should not be associated with others. If the frames are associated with one another, it is not possible to parallelize the rendering process because renderers need to share the past frames to draw the current frame.

Multiple web standards were used for the system. To sample the user's current viewing position and orientation, we used the WebXR device API. The API provides the viewing position and orientation of the user as Euler angles. Because the NeRF implementation receives the camera input as an affine transformation matrix, we convert the 3D position and viewing direction from the API to an affine transformation matrix, and then send the matrix to the rendering server using WebSockets. We had to sample the user's position and orientation frequently to provide a smooth transition of the resulting video stream. Using the WebSockets API, we did not experience any problem sending data every 10 ms. It was important for the protocol to maintain the order of the viewing position and orientation data. It is better to receive motion data slowly than to receive them out of order because the resulting motion of the video stream will be affected by the order. WebSockets guarantee the order of the data because they use the TCP protocol.

4.2.1 Renderer. Although the renderer was forked from instant-ngp, the implementation was modified in several ways. It was modified to run as a daemon on the server, serving the frame request upon arrival. This makes it possible to separate the renderers from an encoder because we can always assume that the renderers will be ready to serve a frame request. This also prevents the encoder from being affected by the failure of the renderer.

Also, the renderer has a limitation in that it can output only the scene or depth in a single rendering cycle, although it uses depth information internally to render the scene. We modified the source of the renderer to render both the scene and depth information simultaneously, as we planned to harmonize the scene with existing 3D assets. We also added a thread that renders a legacy 3D asset using OpenGL.

The renderer then receives a frame request from the encoder containing the index, width, height, and viewpoint of the frame to be generated. The renderer renders a frame, harmonizes the frame with legacy 3D assets, and responds to a request with the rendered frame. Because there are no relations between frames, a single renderer can serve the frame requests from multiple clients.

4.2.2 Encoder.

Requirements. The requirements for the encoder and streaming protocol in this system are unique. We were not able to use existing solutions because we had different requirements compared with popular video rendering and streaming methods. First, the frames collected from the renderers were not associated with each other. Second, the specific timestamp of each frame was not important because we only have to provide the latest rendered frames. Third, there was no need for audio/video synchronization. Finally, for our purpose, the video stream needs to be playable without any add-ons to a web standards-compliant browser.

Design choices. We selected the H.264 codec implemented by libx264 because it is widely supported and most devices have hardware acceleration support. Because HMDs have a stereoscopic display, we should provide a separate frame for each eye by considering the parallax between the eyes. There are two methods for rendering a scene while considering the parallax between the eyes: spatial image warping and two-pass rendering. We chose the latter method of rendering the same scene twice with the parallax in consideration.

Structure. The encoder serves as a client for the renderer daemon. One encoder can be connected to multiple renderer daemons to parallelize and speed up the rendering process. Once the encoder connects to all renderer daemons provided by a command line argument, the program begins and starts a loop that receives the viewpoint from the client HMD using WebSockets. To facilitate an immersive experience of a user, the viewpoints should arrive on time and in the correct order. With WebSockets, we managed to send the viewpoints once every 10 ms, which is less than the time consumed for rendering each scene. Because frame rendering is parallelized, it is crucial to frequently update the coordinates.

Once the viewpoint update loop is running, the encoder sends the renderer a frame request containing the index, resolution, and viewpoint when a request is generated. The request is generated shortly after each encoder serves the frame request.

The computing power requirements for each frame are different for the NeRF renderer. If the space we render has fewer features, or if the training dataset does not contain much information about the space, the renderer takes more time to render. In other words, the rendering time for each frame is different. For instance, when the

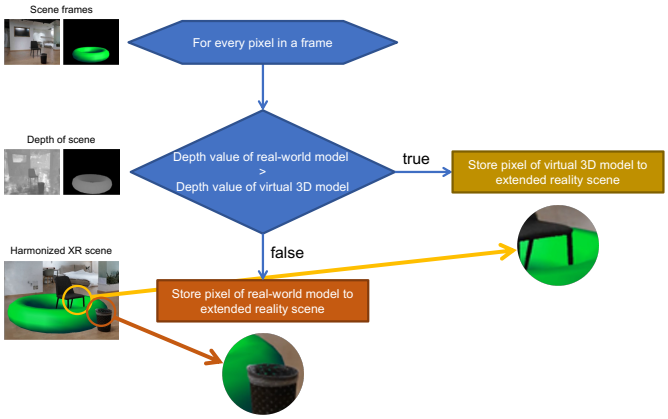

Figure 5: Harmonization process of NeRF output and 3D assets.

encoder requests four frames based on the user's current moving viewpoint, the first frame can be rendered after the second, third, and fourth frames. If we honor the frame's index, the video stream will show the first, second, and third frames almost simultaneously, and then move to the fourth frame. If we do not honor the index, the user's movement is not correctly represented by the video stream. Therefore, we designed the system to strictly honor the frame index while dropping the frames if it takes too long to render.

The rendered frames arriving at the encoder were placed in a queue. A thread preprocessed the frames. The frame was in RGB format; however, to encode it efficiently, we transformed it into the YUV format. We also added subtitles to the frames for debugging. When preprocessing was completed, the frames were sorted by index and fed into the H.264 encoder. The encoder receives the framed and generated an encoded packet.

Theoretically, this structure is not limited by the number of cloud renderers. Based on this structure, we managed to run eight cloud renderers concurrently to generate a single stream with an eight times increase in frame rate.

4.3 Harmonization with legacy 3D assets

NeRF uses volume rendering technology and outputs a scene as an RGB image, making it difficult to place 3D assets naturally. However, NeRF outputs the apparent color and volume density of the scene. The density information can be used to estimate the depth of a certain view. Using this information, we can perform depth testing between the depth buffer of legacy 3D assets and the estimated depth from NeRF.

We compared the depth information from the NeRF renderer with the depth buffer of an object rendered through the conventional 3D renderer, using the method described in Figure 5. The results are shown in Figure 6.

Although NeRF can estimate depth from images, it is not as accurate as the depth from the 3D renderers because it is the result of volume rendering. As shown in Figure 7, there are areas where the NeRF cannot correctly estimate depth. For example, Figure 7 (a) shows a region where NeRF depth estimation is accurate, and

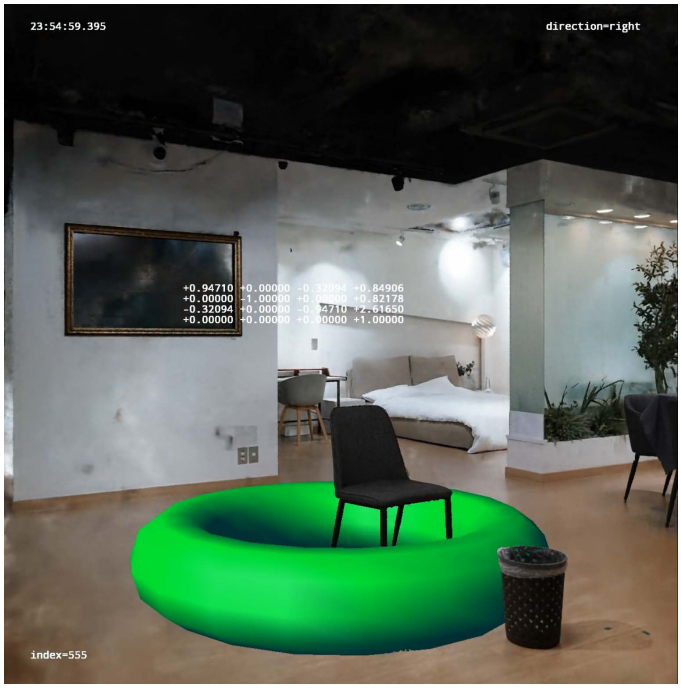

Figure 6: Harmonization using the NeRF volume density output.

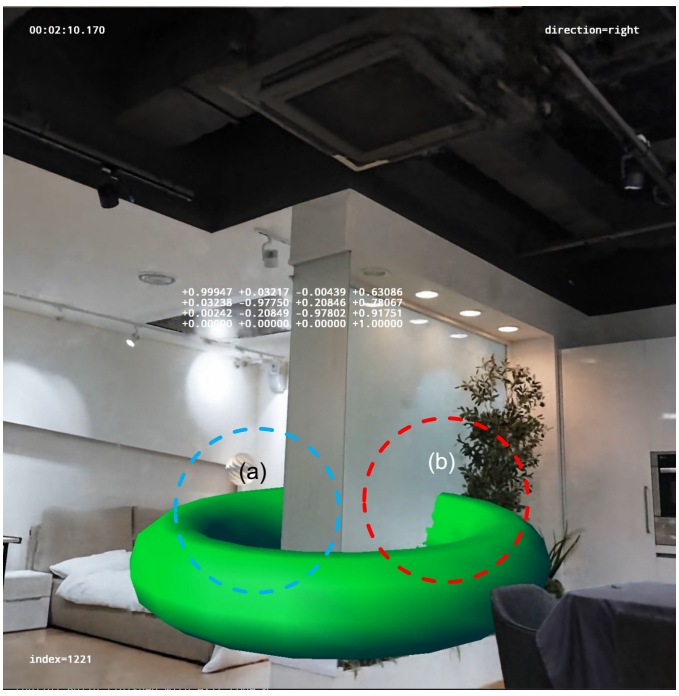

Figure 7: Estimation of depth on surfaces with diffused reflections.

Figure 7 (b) shows a region where NeRF has trouble estimating depth.

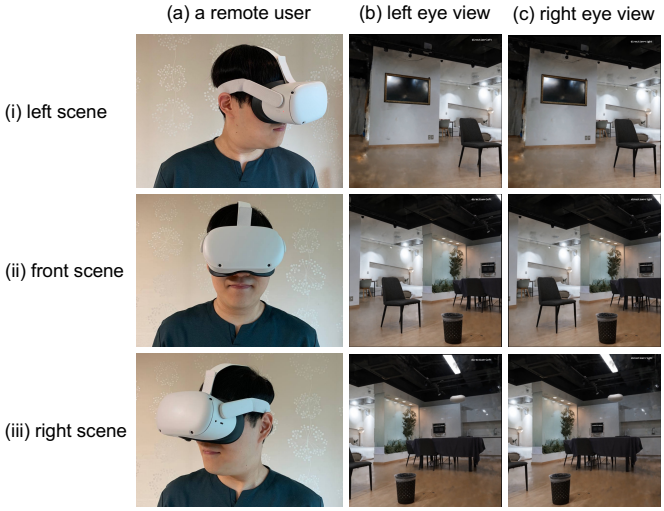

Figure 8: Remote user wearing an untethered HMD viewing a streamed XR scene generated by NeRF.

4.4 Instant XR streaming

For packets generated from the encoder, a streaming protocol is necessary to deliver them to the client as quickly as possible. However, we faced difficulties in choosing an appropriate streaming protocol to transport the stream to the client. Streaming protocols typically focus on latency, reliability, or scalability. Examples of protocols that choose reliability and scalability include HLS [Pantos and May 2017] and MPEG-DASH [MPEG 2019]. However, these protocols were not appropriate for our case, for our goal was to only send the stream to a single client, which, in turn, emphasizes diminishing latency. Although protocols prioritizing latency are favored, protocols such as RTSP [Rao et al. 1998] and SRT [Sharabayko et al. 2021] were not considered because they were not compliant with web standards.

Streaming protocols often generate delays for a multitude of reasons. HLS or MPEG-DASH delays the stream by tens of seconds to create a chunk that can be instantly downloaded by the client. This enables the protocol to scale the stream and conserve computing power by not sending the stream to clients too often. RTSP and WebRTC [Jennings et al. 2022] introduce delays in matching video and audio synchronization. These delays, which are useful in most streaming cases, are a critical disadvantage in real-time XR streaming.

We devised a novel method to rapidly stream video packets to a client without introducing any delay. Although this method is not the best to implement instant streaming, it is by far the easiest way to achieve promising quality. WebCodecs API [Cunningham et al. 2022] is a web standard proposed by Google, Mozilla and Microsoft. This standard aims to provide low-level media processing APIs to JavaScript web applications. Chrome browser added support for WebCodecs standard in 2021 [Cunningham and Sanders 2022]. Firefox browser plans to support the API in the near future [Chang 2022]. We embraced this API due to its roadmap as a standard and the ability to render videos without any artificially introduced delay. We created a system that sends raw encoded packets to a client using the WebSockets and decodes them using WebCodecs API.

This method aligns with our purpose and the cloud-based rendering. However, there are severe limitations to this method for cases other than ours. This method completely disregards the timestamp information for each packet. For every packet, once it arrives and is decoded, it is instantly displayed to the user. Audios were not available with this method because it does not support the synchronization of the timestamp from the audio and video streams. Additionally, scaling the stream to multiple users is computationally intensive because the frames should be sent to every user simultaneously and in real-time.

For our implementation to access an XR device and be web standard compliant, we used the WebXR device API. API enables web applications to render a 3D scene to the chosen device at an appropriate frame rate. The API has been implemented in many web 3D graphics frameworks, including Three.js [Cabello 2022], Babylon.js [Microsoft 2022], and A-Frame [Marcos et al. 2022]. We chose Three.js because it is highly flexible and does not abstract the low-level controls required for our purposes. Using Three.js, we played a video on each eye of the HMD.

We tested our system using eight cloud renderers, each equipped with an NVIDIA A100 discrete GPU. After numerous tests and considerations between rendering speed and quality, we decided to use 1300×1300 as the optimal resolution for each eye. Each cloud renderer was able to output 10 frames per second with resolution. The system was able to generate eighty 1300×1300 frames per second in total, with 40 frames per second for each eye. Notwithstanding other factors of latency remaining, we were able to deliver a new frame every 25 ms, creating an immersive experience inside the mirror world generated by the NeRF successfully. Figure 8 shows a remote user wearing an untethered HMD. Two scenes were rendered for each eye regarding the parallax between eyes. The scene moved along smoothly from the viewpoint of the user.

5 CONCLUSION

Our proposed method can provide a mirror world as an immersive XR environment without modeling process by capturing several pictures of the actual environment. No problem will arise when using 3D assets, as they are combined with the conventional XR production pipeline.

Unfortunately, the high computational power requirements of the current NeRF structure and implementation makes it difficult to design a scalable solution for our proposed system. To generate an immersive environment for one user, we need at least two NVIDIA A100 discrete GPUs. To serve more concurrent users, we need to linearly increase the processing power. We do not believe our current structure could scale to serving more than ten users. However, computational demand of NeRF has been rapidly decreasing [Tewari et al. 2022]. The growing research and development in the neural rendering field makes it possible to envision the not-too-distant future where even the GPUs in mobile devices will be able to generate an immersive environment with NeRF with a fraction of the total computational power. By then, we will be able to aggressively tune our structure to be more scalable. We expect the experience

and errors faced in developing our current proposed system and structure will provide a milestone for the future.

In future work, we will find a suitable method for instant XR services by evaluating the usability and realism based on a comparison among various real-time XR environment generation methods. For example, we will compare the XR environment of an identical space generated by the following three methods:

(1) Mesh model: a method to reconstruct a 3D environment using photogrammetry at a rapid speed.
(2) Point cloud: a method for quickly creating and rendering a point cloud using a 3D scanner such as LiDAR.
(3) NeRF: a method to skip modeling using NeRF.

Herein, the expected artifacts from each method are listed below:

(1) Mesh model: no model exists when the user approaches the unmodeled space.
(2) Point cloud: a cavity exists in an unscanned space or a point cloud of a coarsely scanned region.
(3) NeRF: The NeRF model is inaccurate for the part where the input image is insufficient.

We will compare the required time and quality of the output of each method instantly providing an XR environment from an existing space through comparative experiments and user evaluations. We will continue our research on improving user experience by finding how to overcome the lack of data in advance.

Moreover, there is a need for further work in improving the user experience. To prevent users from feeling motion sickness, we need to lower the Motion-to-Photon(MTP) latency in the current system. There has been numerous research [van Waveren 2016; Xiong and Peri 2021] to reduce MTP latency in rendering 3D mesh models and recent developments in mobile devices have made it possible to aggressively adopt them [Aksoy and Beeler 2015; Antonov 2015]. However, reducing MTP latency in volume rendering methods has not been studied often. We plan to explore ways to reduce MTP latency, in particular where volume rendering is used based on remote user input.

ACKNOWLEDGMENTS

This work was supported by the Industrial Technology Innovation Program (20012462) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea), the National Research Foundation of Korea (NRF) grant (NRF-2021R1A2C2093065) funded by the Korea government (MSIT) and the KIST under the Institutional Program (Grant No. 2E31561).

REFERENCES

Volga Aksoy and Dean Beeler. 2015. INTRODUCING ASW 2.0: BETTER ACCURACY, LOWER LATENCY. Retrieved September 29, 2022 from https://www.oculus.com/blog/introducing-asw-2-point-0-better-accuracy-lower-latency/

Michael Antonov. 2015. Asynchronous Timewarp Examined. Retrieved September 29, 2022 from https://developer.oculus.com/blog/asynchronous-timewarp-examined/

Lucio Azzari, Federica Battisti, and Atanas Gotchev. 2010. Comparative Analysis of Occlusion-Filling Techniques in Depth Image-Based Rendering for 3D Videos. In *Proceedings of the 3rd Workshop on Mobile Video Delivery* (Firenze, Italy) *(MoViD '10)*. Association for Computing Machinery, New York, NY, USA, 57–62. https://doi.org/10.1145/1878022.1878037

Ricardo Cabello. 2022. Three.js. Retrieved July 30, 2022 from https://github.com/mrdoob/three.js/

Chun-Min Chang. 2022. Firefox Bugzilla: [meta] Tracking bug for WebCodecs API implementation. Retrieved September 29, 2022 from https://bugzilla.mozilla.org/show_bug.cgi?id=WebCodecs

F Condorelli, F Rinaudo, F Salvadore, and S Tagliaventi. 2021. A Comparison Between 3d Reconstruction Using Nerf Neural Networks and Mvs Algorithms on Cultural Heritage Images. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 43 (2021), 565–570. https://doi.org/10.5194/isprs-archives-XLIII-B2-2021-565-2021

Chris Cunningham, Paul Adenot, and Bernard Aboba. 2022. WebCodecs. Retrieved July 31, 2022 from https://www.w3.org/TR/webcodecs/

Chris Cunningham and Dan Sanders. 2022. Chrome Platform Status Feature: WebCodecs. Retrieved September 29, 2022 from https://chromestatus.com/feature/5669293909868544

Brian Curless and Marc Levoy. 1996. A Volumetric Method for Building Complex Models from Range Images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, New York, NY, USA, 303–312. https://doi.org/10.1145/237170.237269

Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. 2021. FastNeRF: High-Fidelity Neural Rendering at 200FPS. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 14346–14355.

Cullen Jennings, Henrik Boström, and Jan-Ivar Bruaroey. 2022. WebRTC 1.0: Real-Time Communication Between Browsers. Retrieved July 30, 2022 from https://www.w3.org/TR/webrtc/

Brandon Jones, Manish Goregaokar, and Rik Cabanier. 2022. WebXR Device API. Retrieved July 30, 2022 from https://www.w3.org/TR/webxr/

Yongjae Lee and Byounghyun Yoo. 2021. XR collaboration beyond virtual reality: work in the real world. *Journal of Computational Design and Engineering* 8, 2 (2021), 756–772. https://doi.org/10.1093/jcde/qwab012

Yongjae Lee, Byounghyun Yoo, and Soo-Hong Lee. 2021. Sharing Ambient Objects Using Real-Time Point Cloud Streaming in Web-Based XR Remote Collaboration. In *The 26th International Conference on 3D Web Technology* (Pisa, Italy) *(Web3D '21)*. Association for Computing Machinery, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/3485444.3487642

Diego Marcos, Don McCurdy, and Kevin Ngo. 2022. A-Frame. Retrieved July 30, 2022 from https://github.com/aframevr/aframe

Microsoft. 2022. Babylon.js. Retrieved July 30, 2022 from https://github.com/BabylonJS/Babylon.js/

Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul P. Srinivasan, and Jonathan T. Barron. 2022. NeRF in the Dark: High Dynamic Range View Synthesis From Noisy Raw Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 16190–16199. https://openaccess.thecvf.com/content/CVPR2022/papers/Mildenhall_NeRF_in_the_Dark_High_Dynamic_Range_View_Synthesis_From_CVPR_2022_paper.pdf

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Computer Vision – ECCV 2020*. Springer International Publishing, Cham, 405–421. https://doi.org/10.1007/978-3-030-58452-8_24

MPEG. 2019. *Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats*. Standard ISO/IEC 23009-1:2019. International Organization for Standardization, Geneva, CH. https://www.iso.org/standard/79329.html

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics* 41, 4 (jul 2022), 1–15. https://doi.org/10.1145/3528223.3530127

Roger Pantos and William May. 2017. HTTP Live Streaming. RFC 8216. https://doi.org/10.17487/RFC8216

Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. 2021. HyperNeRF: A Higher-Dimensional Representation for Topologically Varying Neural Radiance Fields. *ACM Trans. Graph.* 40, 6, Article 238 (dec 2021). https://doi.org/10.48550/arXiv.2106.13228

Anup Rao, Rob Lanphier, and Henning Schulzrinne. 1998. Real Time Streaming Protocol (RTSP). RFC 2326. https://doi.org/10.17487/RFC2326

Michael Schmeing. 2011. *Depth Image Based Rendering*. 279–310. https://doi.org/10.1007/978-3-642-22407-2_12

Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4104–4113. https://doi.org/10.1109/CVPR.2016.445

Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. 2016. Pixelwise View Selection for Unstructured Multi-View Stereo. In *European Conference on Computer Vision (ECCV)*. https://doi.org/10.1007/978-3-319-46487-9_31

Maria Sharabayko, Maxim Sharabayko, Jean Dube, Joonwoong Kim, and Jeongseok Kim. 2021. *The SRT Protocol*. Internet-Draft draft-sharabayko-srt-01. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-sharabayko-srt/01/ Work in Progress.

Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. 2022.

Block-NeRF: Scalable Large Scene Neural View Synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8248–8258. https://openaccess.thecvf.com/content/CVPR2022/papers/Tancik_Block-NeRF_Scalable_Large_Scene_Neural_View_Synthesis_CVPR_2022_paper.pdf

A. Tewari, J. Thies, B. Mildenhall, P. Srinivasan, E. Tretschk, W. Yifan, C. Lassner, V. Sitzmann, R. Martin-Brualla, S. Lombardi, T. Simon, C. Theobalt, M. Nießner, J. T. Barron, G. Wetzstein, M. Zollhöfer, and V. Golyanik. 2022. Advances in Neural Rendering. *Computer Graphics Forum* 41, 2 (2022), 703–735. https://doi.org/10.1111/cgf.14507 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14507

J. M. P. van Waveren. 2016. The Asynchronous Time Warp for Virtual Reality on Consumer Hardware. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology* (Munich, Germany) *(VRST '16)*. Association for Computing Machinery, New York, NY, USA, 37–46. https://doi.org/10.1145/2993369.2993375

WHATWG. 2022. WebSockets. Retrieved July 30, 2022 from https://websockets.spec.whatwg.org/

Yingen Xiong and Christopher Peri. 2021. Space-Warp with Depth Propagation in XR Applications. In *2021 IEEE International Symposium on Multimedia (ISM)*. 50–57. https://doi.org/10.1109/ISM52913.2021.00017

Xiaokun Xu and Mark Claypool. 2021. A First Look at the Network Turbulence for Google Stadia Cloud-based Game Streaming. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 1–5. https://doi.org/10.1109/INFOCOMWKSHPS51825.2021.9484481

Minxia Yang, Jiaqi Zhang, and Lu Yu. 2019. Perceptual Tolerance to Motion-To-Photon Latency with Head Movement in Virtual Reality. In *2019 Picture Coding Symposium (PCS)*. 1–5. https://doi.org/10.1109/PCS48520.2019.8954518