RUN: A Case for Cross-Layer Networked Virtual Reality

Yufeng Chen
Purdue University
West Lafayette, IN, USA
chen4044@purdue.edu

Umakant Kulkarni
Purdue University
West Lafayette, IN, USA
ukulkarn@purdue.edu

Voicu Popescu
Purdue University
West Lafayette, IN, USA
popescu@purdue.edu

Sonia Fahmy
Purdue University
West Lafayette, IN, USA
fahmy@purdue.edu

Abstract

Immersive virtual reality (VR) experiences require transmission and rendering of large-scale 3D content, often represented as point clouds or polygon meshes. Unfortunately, existing networked VR systems often fail to fully exploit the flexibility of VR data representations. To address this problem, we propose a cross-layer design that elevates a network data unit to a usable rendering unit for VR applications. Our aim is to bridge the gap between networks and applications in order to enhance visual quality, especially over constrained and variable networks. Our approach, Rendering Unit that is Network-aware (RUN), with two variants, RUN-Packet and RUN-Hybrid, includes mechanisms to effectively utilize network data units when encoding, transmitting, decoding, and rendering. Specifically, we develop additive detail refinement mechanisms and address streaming challenges such as head-of-line (HoL) blocking. We prototype our system in Unity 3D and evaluate it using synthetic network environments and real network traces. Our results with both static and dynamic point clouds demonstrate that RUN significantly reduces stalls and delivers smoother frame updates, enhancing visual quality.

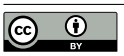
CCS Concepts

• **Networks** → **Network protocol design**; **Cross-layer protocols**; • **Computing methodologies** → *Rendering*; **Virtual reality**.

Keywords

Virtual reality, immersive applications, transport protocols, QUIC

ACM Reference Format:
Yufeng Chen, Umakant Kulkarni, Voicu Popescu, and Sonia Fahmy. 2025. RUN: A Case for Cross-Layer Networked Virtual Reality. In *Proceedings of the 33rd ACM International Conference on Multimedia (MM '25), October 27–31, 2025, Dublin, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3746027.3754800



This work is licensed under a Creative Commons Attribution 4.0 International License.
MM '25, Dublin, Ireland
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2035-2/2025/10
https://doi.org/10.1145/3746027.3754800

1 Introduction

Virtual reality (VR), extended reality, mixed reality, and augmented reality have important applications well beyond gaming and entertainment. VR supports immersive and realistic simulations that allow for safe practice in high-risk scenarios such as driving, aviation, and military training [13]. In the education sector, VR enhances student engagement and learning by providing interactive experiences [2, 9]. In healthcare, VR aids in medical training and patient care, offering a controlled environment for practice and treatment [40]. VR also supports mental health through therapy, and fosters social interaction in realistic virtual environments [52].

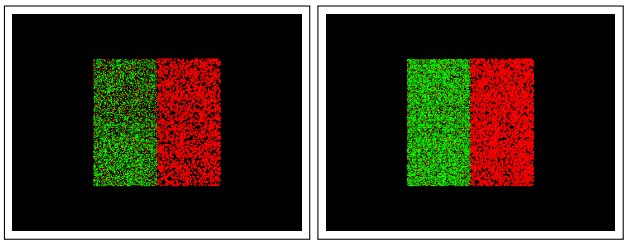
In this paper, we design a cross-layer approach for networked VR applications, and apply it to the case of streaming VR environments represented as point clouds. Tremendous efforts have been made in streaming point clouds over networks, both for dynamic clouds or volumetric videos (e.g., ViVo [14], GROOT [24], YuZu [53], DTP [56]) and static point clouds (e.g., Potree [41], Cesium [6]). A *point* in this context is the minimal rendering unit. Point clouds are typically organized into data structures to facilitate data management and reduce overhead. The most commonly used data structure is an *octree* [24, 41], though 3D tiles are also used [6, 14]. A data "chunk" such as an octree *node* is considered to be the smallest operational unit for rendering.

Most prior systems [6, 14, 24, 30, 41, 53] transmit each data chunk over a network using HTTP running over reliable transport protocols (e.g., TCP or reliable QUIC streams [23]). Transport protocols run on top of the Internet Protocol, which divides data into *network packets*. A common packet size is 1460 Bytes of payload, corresponding to a typical 1500-Byte Maximum Transmission Unit (MTU). A data chunk such as an octree node can contain as many as 10K to 50K points [42]. For uncompressed data, assuming 27 Bytes to represent one point (LAS format [1]), and 10K points per chunk, each HTTP response would be divided into approximately 185 packets, where each packet encodes approximately 54 points.

Our work is founded on the following key observation: *reliable transmission of data chunks is not required.* This is because rendering can often tolerate a certain number of missing points. As with 2D video streaming, where dropping certain frames is tolerable [34], losing some points can also be acceptable. In fact, point cloud transmission over a wireless network is often lossy, and retransmissions delay rendering on the client device, e.g., headset, potentially causing cybersickness. Reliable transport protocols such as TCP and reliable QUIC streams [23] enforce *in-order delivery*

of data to the application layer. This creates a *head-of-line (HoL) blocking* problem: points contained in packets that arrive after a packet loss remain blocked in the receiver buffer until missing packets are retransmitted and received, preventing their use by the application and causing stalls and visual quality degradation. Additionally, broadcast or multicast transmission, which can be beneficial in multi-user VR scenarios [30, 55] is not easy to use with reliable transport.

Figures 1a and 1b illustrate the HoL blocking problem by comparing how a rendered point cloud appears to the user at the exact same moment in time in the case of HoL blocking (Figure 1a), versus without HoL blocking (Figure 1b), where clearly more of the received points have been rendered. In this experiment, we detect HoL blocking by monitoring each packet arrival time and position.



(a) With HoL blocking **(b) Without HoL blocking**

Figure 1: Sample visualization of HoL blocking effect: 10330 points (left) and 16000 points (right).[1]

Our goal in this work is to bridge the gap between the network and the VR application. We propose a cross-layer approach, *Rendering Unit that is Network-aware (RUN)*, where the application on the client device utilizes network data units immediately when they arrive.

The paper makes the following contributions: (1) We propose RUN for alleviating the mismatch between the fine-granularity data transmission unit and the coarse-granularity of the typical VR application operational unit, elevating the transmission unit to serve as the visual progressive refinement unit. We develop two variants of RUN, RUN-Packet and RUN-Hybrid, where RUN-Hybrid balances the data size versus rendering smoothness tradeoff, (2) We demonstrate how using multiple concurrent transmission streams and decoupling reliability from in-order delivery can mitigate the head-of-line blocking problem in reliable transmission, and (3) We prototype our proposed approach in Unity 3D, the standard software platform for developing and deploying VR applications, and demonstrate that it enhances streaming point clouds and volumetric videos under real and synthetic network conditions.

2 Background and Related Work

In this section, we give the necessary background and discuss related work.

2.1 Virtual Reality Streaming Systems

VR and video streaming systems improve user experience through several strategies, including carefully selecting data or quality levels to stream [14, 15, 41, 49], applying compression or super-resolution techniques [24, 25, 46, 53], and optimizing network communication, such as through mmWave transmission [54, 55], Forward Error Correction (FEC) [35, 36], and other techniques [15, 20, 21]. Our work aims to increase the usefulness of each arriving network data unit by enabling immediate rendering and progressive refinement. Thus, our cross-layer approach, RUN, is complementary to existing systems and can serve as an enhancement to further improve streaming performance.

2.2 Static and Dynamic Point Clouds

Despite the recent emergence of new VR representations (e.g., Neural Radiance Fields [31], Gaussian Splatting [18]), point clouds remain a widely-adopted format for representing 3D environments. Our approach, RUN, uses point cloud streaming as a case study, but it can be adapted to other VR data formats.

Streaming pre-recorded point clouds has been extensively studied, both for static point clouds (e.g., Potree [41], Cesium [6]) and dynamic point clouds or volumetric videos (e.g., ViVo [14], GROOT [24]). Static point cloud datasets represent unchanging 3D scenes and objects. These datasets are usually large but remain constant over time, allowing users to explore a scene freely without temporal constraints. In contrast, dynamic point clouds capture time-varying scenes, such as moving objects or changing environments. Our work is agnostic to the streaming type: we demonstrate that RUN can enhance streaming for both static and dynamic point clouds.

2.3 Point Cloud Encoding and Compression

We use point clouds in LAS file format (type 3) [1]. Each point occupies 27 bytes, with coordinates stored as 32-bit integers, color channels as 16-bit unsigned integers, and additional information such as intensity, classification, and point source. We transmit all point cloud data since we do not wish to make any assumptions about the reconstruction algorithm on the client and do not know which specific point attributes it may use.

Point cloud compression methods such as LasZip [16], Draco [12], MPEG G-PCC/V-PCC [7, 27–29], and GeoCNNv2 [47] either do not support point clouds with color and lossless compression, or are not fully compatible with the Unity 3D pipeline. Moreover, these methods do not always yield significant improvements in compression efficiency. Therefore, in this paper, we adopt a simple yet effective solution, Brotli [11] and Morton coding, that is easy to integrate with Unity and provides competitive compression ratios.

2.4 Data Representation

Our starting point is the publicly available PotreeConverter 2.0 [42] which constructs an octree to store any input set of 3D points. We then use our own custom scripts to shuffle and compress points as we will describe in Section 3.1.

Multi-resolution data structures, such as octrees, can be organized additively or non-additively. With the additive approach, points at a coarse level of detail (LoD), i.e., higher in the octree hierarchy, are added to those at finer LoD, i.e., lower in the octree, during the refinement process [10, 43], instead of being replaced by them [45]. The key advantage of the additive approach is that it

[1] Additional examples can be found at https://github.com/yufchen/MM25_RUN

eliminates redundancy, but its disadvantages are that it introduces additional management overhead and requires in order tree traversal. Note that spatially adjacent points may belong to different octree levels. PotreeConverter employs an additive approach.

PotreeConverter uses a Poisson disk sampling algorithm. The algorithm selects points by checking if the distance between a new point and already selected points exceeds the smallest allowable distance (spacing), which prevents points from clustering too closely together and creates a distribution with desirable blue-noise properties [8, 41, 51]. Points within a node have a level-dependent minimum distance (spacing) equal to $\frac{S_0}{2^L}$, where S_0 is the root spacing (discussed below) and L is the level of the node in the octree.

According to Nyquist's sampling theorem, the ideal spacing between projected points is 0.5 pixels for the signal, i.e., the image, to be reconstructed from two samples, i.e., points, along each dimension. We estimate the desired target spacing at any given distance d between the viewer and points and we determine the LoD needed at different distances to be $\frac{d \cdot tan(HFoV/2)}{DisW}$, where $DisW$ is the screen display width and $HFoV$ is the horizontal field of view (FoV). The horizontal FoV controls the width of the visible scene along the horizontal axis in the 3D projection. Most displays use square pixels and the pixel aspect ratio is fixed, so the vertical field of view is not a free parameter and can be computed given $HFoV$ and the display resolution $DisW \times DisH$. The root spacing parameter S_0 can be empirically tuned. We set S_0 to $\frac{D}{\alpha}$, where the object size D is the length of the diagonal of the object's 3D axis-aligned bounding box, and α is the point density.

3 Design

We now describe our cross-layer approach, RUN, that makes each network packet an independently decodable and renderable unit. RUN has two variants: RUN-Packet and RUN-Hybrid.

3.1 Data Representation and Rendering

In RUN, point distribution in octree nodes and the point size used for rendering must be carefully managed to maintain visual coherence as packets arrive and the points contained in each packet are rendered on the client device. We develop two mechanisms: *adaptive point size* and *uniform intra-node distribution* for effective rendering.

3.1.1 Adaptive Point Size. Figures 2a–2c compare three point sizing strategies, each rendering the same number of points per quadrant: (2a) fixed size, (2b) node-wise adaptive size, and (2c) point-wise adaptive size. In the figure, the points are white, and the top-left quadrant is 1% full, bottom-left is 10%, top-right is 40%, and bottom-right is 80%.

Compared to the fixed-size strategy in Figure 2a, the node-wise approach used in prior work [41] (Figure 2b) reduces visual gaps. However, it still leaves noticeable artifacts. In this approach, the point size within a node's bounding box is uniformly set according to the node's spacing value, regardless of the actual fraction of received data.

To address this limitation, we introduce a *point-wise* adaptive point size rendering approach, to allow defining the size of each point progressively and individually as data is received, and not just once per node. As seen in Figure 2c, all four quadrants are almost solid white because the size of the white points is based on the fraction of received points in a node.

Equation (1) shows how the spacing for node n, S_n is computed.

$$S_n = \begin{cases} S_0 \cdot (R \cdot 2^{2L} + \sum_{i=0}^{L-1} 2^{2i})^{-\frac{1}{2}} & L \geq 1 \\ S_0 \cdot R^{-\frac{1}{2}} & L = 0 \end{cases} \quad (1)$$

where L is the node level, and R is the fraction of node n's points that have already been received.

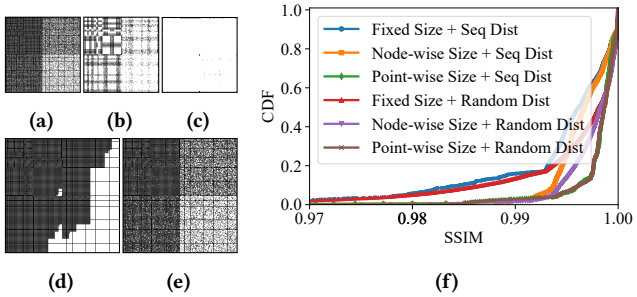


Figure 2: a, b and c show fixed, node-wise and point-wise point size. d and e show original and shuffled point distribution. f shows the CDF of SSIM values over rendered points for the "Longdress" dataset.

This resizing step introduces extra computation by the GPU for every point to determine its current spacing value. We follow Potree's [41] strategy and construct a 2D RGBA32 format texture in the CPU. RGB channels are used to locate the current node region of every point. The alpha channel is used to record the received ratio of each node and is updated every frame.

3.1.2 Random Intra-node Point Distribution. Points in an octree node are typically ordered according to how they were scanned and captured. To ensure that representative points are obtained when only partial octree node data has been received, we randomly shuffle the points in a node as a preprocessing step. This way, each packet contains a mix of points that provide more visual information while waiting for the remaining points of the node. Figure 2d shows the original sequential ordering and Figure 2e shows the order after adding the shuffling step. In these figures, all points are rendered at the same fixed point size. We observe that shuffling ensures that arriving points are more evenly distributed, thereby covering a larger portion of the view.

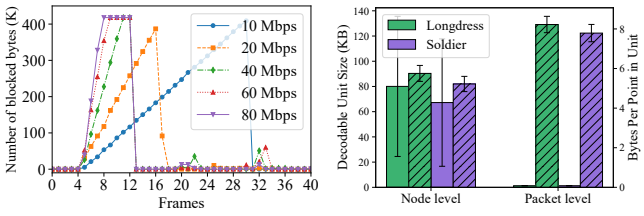
3.1.3 Evaluation. Figure 2f demonstrates that the two methods discussed above (Sections 3.1.1 and 3.1.2) indeed improve the quality of the rendered image for the "Longdress" dataset [19]. Specifically, the CDF curve for the combination of point-wise sizes and random point distribution lies furthest to the right, indicating that higher rendering quality is achieved with fewer rendered points. In other words, this approach efficiently utilizes the set of points received early on, resulting in higher SSIM values even at lower rendering densities. We conducted similar experiments on other datasets and observed consistent trends.

It is worth noting that when points are occluded, increasing the rendered point size may sometimes degrade quality due to overlapping points. In this paper, we assume there is little or no occlusion and leave this issue for future study. In the experiments presented in the remainder of this paper, we adopt randomly shuffled point clouds and point-wise size rendering.

3.2 Data Transmission and Delivery

Prior systems for streaming static point clouds (e.g., Potree [41]) or dynamic point clouds (e.g., ViVo [14] and GROOT [24]) use reliable streams to transport VR data to clients. Unfortunately, the "stream" abstraction implemented by transport protocols such as TCP or QUIC [23] couples reliability with *in-order delivery*, creating a *head-of-line blocking (HoL)* problem. Specifically, lost packets prevent points in packets that arrive later from being delivered to the application and rendered.

Figure 3a shows how a single packet loss blocks subsequent data when using a single reliable stream and in-order delivery at different link rates (round trip time (RTT)=50 ms, frames per second=60 in this experiment). Points are delayed by multiple frame durations waiting for loss recovery of earlier packets.



(a) Blocked data **(b) Size/Rendering tradeoff**

Figure 3: a shows the impact of a single packet loss at different link rates. b shows the size and rendering flexibility tradeoff for two datasets (solid bars represent decodable and rendering unit size, where lower is better; hatched bars represent number of Bytes per point in this unit, i.e., how well we can compress points, where lower is better).

Two strategies, namely out-of-order delivery and multi-stream transmission, can alleviate the HoL blocking problem. Out-of-order delivery allows the transport layer to immediately pass newly arriving packets on to the application, regardless of gaps caused by earlier losses, thus eliminating the need to wait for retransmissions of lost packets. Multi-stream transmission mitigates HoL blocking by distributing data across multiple concurrent streams; packet loss in one stream only delays data within that stream, while other streams continue to progress.

Table 1 compares TCP, QUIC (streams and datagrams), UDP, and SCTP (Stream Control Transmission Protocol [44]) in terms of whether they provide reliability and whether they support out-of-order (unordered) delivery to the application.

We evaluate SCTP, TCP and QUIC-stream performance in Section 5.2.1. Although SCTP with the out-of-order feature avoids HoL blocking, it is not widely deployed, especially on non-Linux platforms. Additionally, SCTP lacks native support for modern congestion control algorithms such as BBRv2 [5] which dictate loss recovery methods. In contrast, QUIC and TCP are widely utilized in the Internet. We use *multi-stream transmission with QUIC* to mitigate the HoL blocking problem in RUN.

Table 1: Protocol comparison.

Protocol	Reliability	Unordered Delivery
TCP	✓	×
SCTP	✓	✓
QUIC-streams	✓	×
QUIC-datagram, UDP	×	✓

3.3 Variants of RUN

Prior work [24, 41, 43] encodes, compresses, and transmits complete data chunks, in the format of octree nodes or 3D tiles, later split into multiple network packets. We implement a similar approach, which we name the *Node* method, serving as a baseline in our evaluation.

Packets provide more flexibility and impose no reliability requirements in terms of being able to immediately decode and render points in each packet as they arrive, regardless of packet loss. We therefore utilize units of packets in an approach we name *RUN-Packet*.

Using packets, however, prevents compression of larger data units and hence consumes more bandwidth, compared to the traditional approach of compressing and transmitting entire nodes reliably. Figure 3b validates the observation that the node-level method (on the left) is better for compression (yields smaller sizes as shown by the hatched bars), whereas the packet-level method (on the right) allows finer-grained rendering (very small decodable unit size as shown by the solid bars) which gives higher visual performance. Nodes achieve better compression (hatched bars), with decodable units in the range of 60 ∼ 80 KB, whereas packets enable much finer-grained decoding and rendering (solid bars) with 1.5 KB as decodable unit size.

To balance this tradeoff, we design a hybrid approach, *RUN-Hybrid*, given in Algorithm 1, that may use packets or nodes based on priority. In RUN-Hybrid, we send high priority points in the point cloud as independently decodable and renderable *packets*, and the remaining points as nodes. We follow prior work [41] and assign each node a priority based on the projected size on the screen. Without occlusions, nodes with larger projected size will contribute more to the rendered image quality. We use a parameter τ as a threshold on the priority to determine whether to use packets or octree nodes as independent units. Although τ is currently a static input parameter, we will explore setting it dynamically based on network conditions in our future work. Our evaluation will show that RUN-Hybrid outperforms simply using all nodes (Node) or all packets (RUN-Packet), especially under lossy conditions.

3.4 Priority Merging

We use the calculated projected size of each node, s_i as computed in line 4 of Algorithm 1, to assign transmission priorities, in addition to using it to decide on using nodes or packets as decodable units. The QUIC protocol uses a strict priority scheduler to schedule packets from different streams. When the sending window is smaller than the node size, a single stream is being transmitted at a time. Lower-priority streams must wait until all data in the highest-priority

Algorithm 1: RUN-Hybrid

Input : • Current camera pose: $\mathbf{T}_c$
• Octree nodes: $\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$
• Hybrid threshold: $\tau \in [0, 1]$
Output : • Packet priority queue $Q_p = \{(n_i, s_i)\}$
• Node priority queue $Q_n = \{(n_j, s_j)\}$

begin
  Initialize max-priority queue $Q \leftarrow \emptyset$
  foreach *node* $n_i \in \mathcal{N}$ **do**
    Compute projected size s_i from $\mathbf{T}_c, n_i$
    Q.insert(n_i, s_i)
  $Q_p \leftarrow \emptyset, Q_n \leftarrow \emptyset$
  Get maximum projected size $s_{\max} \leftarrow Q[0].s$
  Initialize partition index $k \leftarrow 0$
  while $k < |Q|$ ***and*** $Q[k].s \geq \tau \cdot s_{max}$ **do**
    $k \leftarrow k + 1$
  $Q_p \leftarrow Q[0:k]$, $Q_n \leftarrow Q[k:|Q|]$ **return** (Q_p, Q_n)

stream has been sent. In this case, HoL blocking occurs within a stream as discussed earlier. Additionally, certain regions of the point cloud may receive a larger number of points, creating spatial imbalance.

To mitigate this problem, we group nodes with similar projected sizes into priority categories. Specifically, we first identify the current maximum projected size s'_{max} among nodes that have not yet been assigned to a category. Then, we assign the same priority to all nodes n_i whose projected size satisfies $s_i \geq s'_{max} \cdot \theta$. This process is repeated until all nodes have been assigned priorities.

We use the parameter θ to control how aggressively nodes with similar projected sizes are grouped into the same priority class. For static point clouds, where content remains useful over time, more aggressive merging tends to perform better. In contrast, dynamic point clouds change continuously, and finer-grained prioritization helps ensure that the most relevant content is delivered. In our experiments, we empirically set parameter $\theta = 0.75, 0.92$ for static and dynamic point clouds, respectively. This parameter can be adapted based on received data and network conditions, which we plan to further investigate in our future work.

In the case when each packet is independently decodable, increasing the number of transmission streams always improves performance. In contrast, when nodes serve as decodable units, using a large number of streams may delay the complete delivery of individual nodes, leaving partial data unusable. Therefore, we add an upper bound (set to 5 in our experiments) on the number of concurrent streams used when transmitting nodes to maintain a balance between parallelism and data completeness. Our evaluation in Section 5.2.3 will show that priority merging is beneficial.

4 Implementation

As with any streaming system (e.g., DASH [17]), two critical decisions need to be made in the design: (1) *what to request* and (2) *when to request it*. Making these decisions on the client side or on the server side each has advantages and disadvantages. Server-side intelligence offers centralized control for optimized resource allocation and scheduling when the virtual reality application supports multiple concurrent clients, and reduces client battery consumption. In contrast, client-side intelligence is easier to implement and the server scales better with an increasing number of clients, but it consumes scarce client resources. In this work, we focus on single client use cases, so we use client-side intelligence (as in DASH) in RUN and use standard servers. We will explore server-side intelligence in future work.

We prototype our RUN client in Unity 3D, making it easily adaptable to VR headsets. We have three main modules at the client: (1) rendering, (2) request, and (3) network interface. The design of our rendering module is inspired by prior work [14, 33, 34, 41, 50]. Our implementation consists of approximately 6,373 lines of C# code, 232 lines of HLSL shaders, and Unity integration.

Rendering module. Unity 3D does not provide native support for point clouds, but it allows rendering point cloud data as mesh vertices without connecting them [50], which we utilize.

Request module. The client makes decisions on which nodes to request, in what order, and how many requests can be sent in parallel. These decisions are made after each traversal of the octree structure. We use a Breadth-First Search (BFS) tree traversal. We first examine if the node's bounding box is within the view frustum; if not, we stop exploring its children. Then, we calculate the projected size of the selected node. If it is smaller than the specified minimum size, we stop exploring its children. After completing the traversal, the request module sends node requests to the network interface module, along with their priorities and a possible limit on the number of concurrent streams.

Network interface module. The network interface module is divided into a sending component and a receiving component. This module supports the HTTP/1.1 (TCP), SCTP, and QUIC protocols. The module sends requests to the server (with requested priorities suggested by the request module). When receiving packets, if packets are treated as rendering units, the module will instantly attempt to decode any newly received data; otherwise, it will wait until a complete node is received before decoding.

5 Evaluation

Our evaluation aims to answer the following questions: (1) How well do out-of-order delivery and multi-stream transmission mitigate the HoL blocking problem? (2) How does RUN-Hybrid compare to only node units and only packet units (RUN-Packet) for both static point clouds and dynamic point clouds with real and synthetic network traces? and (3) How does priority merging affect performance?

5.1 Testbed, Datasets, and Networks

Our experimental setup consists of one server, one client, and one router. The server runs on a System76 Meerkat equipped with an Intel Core i7 1260P 12-core processor and 64 GB of RAM. The client runs on a Windows 10 desktop machine equipped with an AMD Ryzen 5 5600X 6-core processor, 32 GB of RAM, and an NVIDIA GeForce RTX 3060 Ti. For rendering, we use a similar setting as the Meta Quest 3, with the resolution set to 2064×2208p, horizontal FoV set to 110° and target frame rate set to 72 Hz. We use a Banana Pi BPI-R3 router running OpenWrt 23.05.2.

Datasets. Table 2 lists the point cloud datasets we use. Unless otherwise specified, we show results with the "Plane" dataset for

static point clouds and the "Longdress" dataset for dynamic point clouds. We have validated our findings on all datasets.

Table 2: Datasets used in experiments.

Name	Source	Point Cloud Type
Tikal	Sketchfab [37]	Static
CA21	OpenTopography [3]	Static
Plane	Open3D	Static
Guanyin	Sketchfab [38]	Static
Longdress	8iVSLF [19]	Dynamic
Soldier	8iVSLF [19]	Dynamic

Network conditions. We connect the client, router, and server through Ethernet cables and use *tc* and *netem* on the router or *mahimahi* [26] to emulate different network conditions. We first conduct experiments under controlled networks, where we vary the link rate (bandwidth) and introduce random loss at certain rates. Then, we replay real network traces, as summarized in Table 3, by selecting multiple 10 s segments from each trace. For traces without round trip time (RTT) information, we set RTT to 10 ms. We select publicly available bandwidth traces representing the most common types of networks: Wi-Fi, 4G cellular, and 5G cellular. We choose three recent traces from each type of network. The Wi-Fi traces [32] were collected in January 2021 using NemFi, a trace-driven Wi-Fi emulator. The Orange 4G cellular network traces [22] were obtained in January 2022 in Lannion, France, and the 5G traces [39] were collected from a major Irish mobile operator in 2020.

Table 3: Traces used in experiments.

Network	Average Downlink Bandwidth (Mbps)	Coefficient of Variation (1 s interval)
Wi-Fi (Ideal)	117.32	0.04
4G (Trace B)	78.92	0.48
5G (Static)	121.02	0.73

5.2 Experimental Results

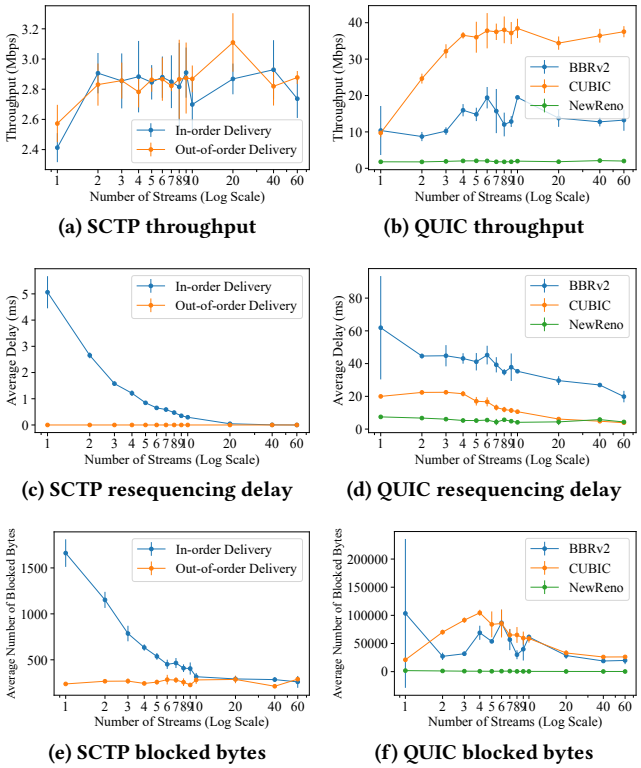
We now discuss our key experimental results. Each experiment is repeated at least five times. All error bars in the figures represent the standard deviation of the results of the experiments.

5.2.1 HoL Blocking. As discussed earlier, when packets are lost, packets that are subsequently received are not delivered to the application unless out-of-order delivery or multiple streams are used. We quantify the impact of this HoL blocking by investigating *throughput* and *resequencing delay*. Resequencing delay measures how long a packet needs to wait at the receiver to be delivered to the application. We also measure the number of bytes that are blocked from delivery to the application every frame.

In this set of results, we use dataset "Plane" with *Node* as the rendering unit. We emulate a network with a round trip time (RTT) of 50 ms, random loss rate of 1% and a link rate of 50 Mbps.

We experiment with SCTP (Figures 4a, 4c, 4e) and with QUIC (Figures 4b, 4d, and 4f).

SCTP supports both in-order and unordered (out-of-order) delivery and a Reno-like congestion control algorithm. We observe



(a) SCTP throughput (b) QUIC throughput

(c) SCTP resequencing delay (d) QUIC resequencing delay

(e) SCTP blocked bytes (f) QUIC blocked bytes

Figure 4: Comparison of SCTP and QUIC under HoL blocking scenarios at 50 Mbps link rate and 50 ms RTT.

that throughput is mostly similar for in-order and out-of-order delivery under different numbers of SCTP streams. When data is delivered in order to the application, the resequencing delay decreases as the number of streams increases. This is because packet loss only blocks a subset of the streams, demonstrating the benefit of using multiple streams. When data is delivered in order, with a smaller resequencing delay, the number of blocked bytes is also lower. When data is delivered out of order, ideally, the number of blocked bytes should be zero. We observe a small value, though, since there is a small delay between receiving data and delivering it to the application, even without resequencing.

QUIC streams only support in-order delivery, and support different congestion control algorithms. We experiment with three algorithms: NewReno, CUBIC, and BBRv2.

As expected, NewReno results are closer to the SCTP results (which use a Reno-like algorithm) than the results with CUBIC and BBRv2, which exhibit significantly higher throughput values. The throughput results with NewReno and BBRv2 do not significantly increase as the number of streams increases. In contrast, the throughput with CUBIC quickly increases when the number of streams increases, especially between 1 and 4. This is because the application may not fully utilize the available bandwidth when there is a small number of streams. Depending on the point cloud dataset, some nodes can be sparse and contain only a few hundred points, resulting in small request sizes. When the request size is smaller

than the congestion window, it becomes difficult to efficiently use bandwidth, and using multiple streams is highly beneficial.

The resequencing delay decreases with more streams. With CUBIC, initially the delay slightly increases and the number of blocked bytes increases as we add more streams, as bandwidth is not fully utilized. As the number of streams increases beyond four or so, the resequencing delay and the number of blocked bytes decrease.

As a comparison baseline, we also experimented with TCP. Since our client runs on Windows 10, the non-disablable auto-tuning receive window throttles throughput under high latency networks with HTTP/1.1. Therefore, we do not use HTTP/1.1 and directly connect to the server with TCP. The responses are returned sequentially on a single TCP connection, which is equivalent to the case when the number of concurrent streams is equal to one. Thus, throughput does not increase with the number of concurrent requests. Using CUBIC, the throughput is 3.71 Mbps, while with NewReno, the throughput is 3.12 Mbps. The corresponding throughput values with QUIC are higher: 9.72 Mbps and 1.78 Mbps, respectively.

In summary, our results show that: (1) HoL blocking can bottleneck point cloud streaming under packet loss, (2) out-of-order delivery eliminates HoL blocking, while multi-stream transport mitigates its impact by distributing data across multiple independent streams, confining blocking effects to individual streams, (3) congestion control algorithms play an important role: both BBRv2 and CUBIC maintain high throughput, and (4) multi-stream QUIC with BBR or CUBIC performs well, and we therefore adopt it in our RUN implementation.

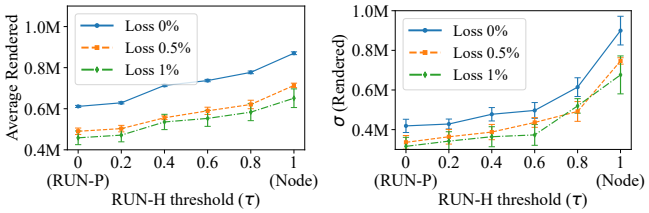
5.2.2 Nodes, Packets, and Hybrids. We now evaluate how RUN-Packet and RUN-Hybrid compare to using nodes as rendering units. We begin by evaluating performance under random loss patterns for both static and dynamic point clouds. Then, we examine performance under different link rates and across various datasets of static point clouds. Finally, we validate our findings using real network traces.

For static point clouds, we make a single decision regarding what data to transmit under packet and node granularities, whereas dynamic point clouds require per-frame decisions and therefore involve multiple transmission decisions over time.

We use the RUN data representation and rendering methods discussed in Section 3.1 in the static point cloud case. Rendering new points always improves visual quality in this case. Since later points contribute little to perceived quality, we use the *number of newly rendered points* as a direct metric for performance (abbreviated as "Rendered" in the figures). Specifically, we consider both the average and the standard deviation of newly rendered points per second. The average indicates how quickly quality improves, whereas the standard deviation reflects the smoothness of frame updates. We use the value for the *Node* method as the baseline and normalize the values for the other two methods accordingly.

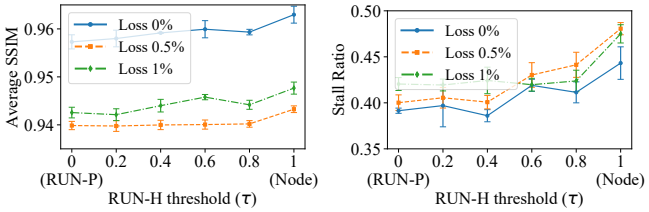
For dynamic point cloud streaming, we assess quality and smoothness using the average MS-SSIM [48] and the stall ratio. The average MS-SSIM reflects the visual quality across the entire sequence, whereas the stall ratio, defined as the number of stalled frames divided by the total frame count, reflects playback smoothness.

Figures 5a and 5b (link rate=50 Mbps, RTT=10 ms) show static point cloud quality and smoothness under no loss, or 0.5% or 1%



(a) Quality **(b) Smoothness**

Figure 5: Static point clouds with loss.



(a) Quality **(b) Smoothness**

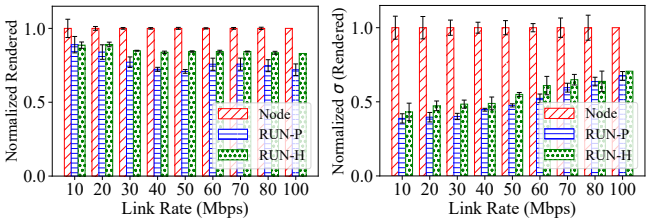
Figure 6: Dynamic point clouds with loss.

random loss rates, while Figures 6a and 6b (link rate=150 Mbps, RTT=10 ms) present the corresponding results for dynamic point clouds. The x-axis represents the hybrid threshold τ: when $\tau = 0$, the method corresponds to sending independently decodable *Packets* (RUN-Packet), whereas $\tau = 1$ corresponds to decodable *Nodes*.

The figures show that increasing τ improves *quality* for the no loss and random loss cases. The improvements are modest since the compression ratio difference between *Node* and RUN-Packet is relatively small. For *smoothness*, as τ increases, the standard deviation for static point clouds (Figure 5b) and the stall ratio for dynamic point clouds (Figure 6b) increase noticeably. This suggests that using packets as rendering units indeed improves smoothness.

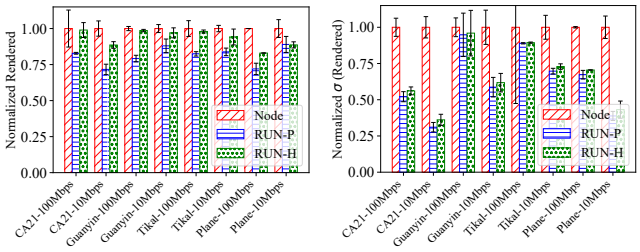
In the following experiments, we select $\tau = 0.6$ to represent RUN-Hybrid as it provides a fair balance between early rendering and compression efficiency.

Figures 7a and 7b (RTT=10 ms, no emulated loss) illustrate the performance for static point clouds under various link rates. RUN-Packet and RUN-Hybrid achieve high smoothness (lower standard deviation), especially at low link rates, at the cost of slightly reduced quality (fewer newly rendered points per second). As the link rate increases, the difference diminishes because higher bandwidth enables the application to receive complete nodes more frequently, reducing the impact of granularity.



(a) Quality **(b) Smoothness**

Figure 7: Static point clouds at several link rates.



(a) Quality **(b) Smoothness**

Figure 8: Static point clouds for several datasets.

Figures 8a and 8b (link rate=50 Mbps, RTT= 10 ms, no emulated loss) further confirm these observations across multiple datasets. However, we find that datasets with less uniformly distributed points in the scene (e.g., Tikal and Guanyin) show reduced benefits for RUN-Packet and RUN-Hybrid at higher link rates. This is because these datasets feature relatively smaller octree node sizes and higher variance, making it more likely for the *Node* method to receive complete nodes within a frame time. Quantifying the impact of point cloud characteristics remains an important question, which we leave for future work.

Finally, as shown in Table 4, we compare the performance of the *Node* method with RUN-Packet and RUN-Hybrid on real Wi-Fi, 4G, and 5G traces. The results are consistent with our previous findings: RUN-Hybrid achieves a good balance between quality and smoothness. We make two additional observations. First, for 4G traces, the *Node* method exhibits poor smoothness. This is because the available bandwidth cannot meet the demands of point cloud streaming. In this case, RUN-Packet and RUN-Hybrid allow the application to proceed without waiting for an entire node, leading to significant improvements in smoothness. Second, although 5G traces exhibit high bandwidth variance, which we expected would amplify the benefits of RUN-Packet and RUN-Hybrid, we do not observe significant improvements with dynamic point clouds. We attribute this to the congestion control algorithm used (BBRv2) which estimates bottleneck bandwidth and RTT to adjust its sending rate, rather than relying solely on packet loss as a congestion signal. This design helps BBRv2 maintain high throughput and low latency in networks with fluctuating bandwidth, by preventing excessive queue buildup and reducing the likelihood of bufferbloat [4, 5]. We still observe that, in trace segments where packet loss indeed occurs (i.e., bandwidth drops to zero), RUN-Packet and RUN-Hybrid achieve significant improvements in smoothness.

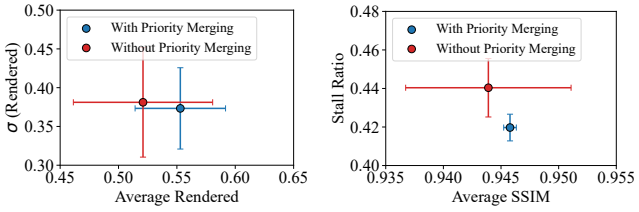
In summary, RUN-Packet and RUN-Hybrid consistently achieve high smoothness while maintaining good quality for both static and dynamic point cloud streaming.

5.2.3 Priority Merging. We evaluate the impact of the priority merging feature in terms of both quality and smoothness for static and dynamic point clouds. In our experiments, when priority merging is disabled, nodes are assigned different priorities, and due to the strict priority scheduling, a single active stream is being transmitted most of the time.

Figures 9a and 9b demonstrate that quality and smoothness improve with priority merging for both the static and dynamic point cloud cases. However, the gains are modest, and the results show high variance, as evidenced by the error bars in both directions. This is because the severity of HoL blocking depends on packet loss location. If a loss occurs early on, more packets are likely to be blocked, increasing the impact. Conversely, if the loss happens towards the end, fewer packets are affected.

Table 4: Performance of *Node*, RUN-Packet, and RUN-Hybrid ($\tau = 0.6$) with wireless traces. Each cell shows Quality / Smoothness (Average Rendered / σ(Rendered) in static point clouds; Average SSIM / Stall Ratio in dynamic point clouds).

Trace	Static Point Cloud (M point/s)		
	Node	**RUN-Packet**	**RUN-Hybrid**
Wi-Fi (Ideal)	1.676 / 0.903	1.172 (-30.08%) / 0.625 (-30.74%)	1.420 (-15.31%) / 0.721 (-20.13%)
4G (Trace B)	1.372 / 0.853	0.987 (-28.08%) / 0.568 (-33.39%)	1.141 (-16.84%) / 0.365 (-44.54%)
5G (Static)	1.450 / 0.913	1.067 (-26.41%) / 0.600 (-34.24%)	1.270 (-12.41%) / 0.655 (-28.31%)
Trace	**Dynamic Point Cloud**		
	Node	**RUN-Packet**	**RUN-Hybrid**
Wi-Fi (Ideal)	0.970 / 0.337	0.964 (-0.58%) / 0.295 (-12.41%)	0.963 (-0.66%) / 0.310 (-8.04%)
4G (Trace B)	0.941 / 0.513	0.940 (-0.04%) / 0.367 (-28.45%)	0.940 (-0.04%) / 0.365 (-28.97%)
5G (Static)	0.944 / 0.470	0.943 (-0.09%) / 0.395 (-15.88%)	0.946 (+0.25%) / 0.415 (-11.71%)



(a) Static point cloud. **(b) Dynamic point cloud.**

Figure 9: RUN-Hybrid ($\tau = 0.6$). RTT=10 ms, loss rate=1%, link rate=50 Mbps (static) or 150 Mbps (dynamic).

6 Conclusions and Future Work

We have shown how a cross-layer approach, RUN, that elevates a network data unit to a unit usable by a point cloud streaming application, can enhance user experience under constrained and variable networks. Our experimental results demonstrate how rendering smoothness can be increased and the head-of-line blocking problem can be mitigated by using finer-grained units for transmitting important data and using multiple concurrent streams.

We plan to extend our work as follows. In terms of design, we plan to add methods to dynamically tune the parameters of RUN (i.e., τ and θ) to adapt to network conditions. In terms of evaluation, we plan to conduct experiments with wirelessly-connected VR headsets and experiments with changing user positions and view directions. Finally, beyond point clouds, we plan to extend our approach to other virtual environment representation methods.

Acknowledgments

This work has been supported in part by NSF grant 2212200 and a gift from Meta.

References

[1] ASPRS (American Society for Photogrammetry and Remote Sensing). 2019. *LAS Specification Version 1.4 - R13.* https://www.asprs.org/wp-content/uploads/2019/03/LAS_1_4_r14.pdf

[2] Kabir Batra, Zirui Zhang, Shuwen Yang, Arnima Agrawal, Yiyin Gu, Bedrich Benes, Alejandra Magana, and Voicu Popescu. 2025. XRXL: A System for Immersive Visualization in Large Lectures. In *2025 IEEE Conference Virtual Reality and 3D User Interfaces (VR).* IEEE.

[3] M.P. Bunds, C. Scott, B. Whitney, and V.J.C. Lee. 2021. High Resolution Topography of the Southern San Andreas Fault from Painted Canyon to Bombay Beach, USA. Distributed by OpenTopography. https://doi.org/10.5069/G94M92RG

[4] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue* 14, 5 (2016). https://doi.org/10.1145/3012426.3022184

[5] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Kevin Yang, Ian Swett, Victor Vasiliev, Bin Wu, Luke Hsiao, et al. 2019. BBRv2: A model-based congestion control performance optimization. In *Proc. IETF 106th Meeting*. 1–32.

[6] Cesium GS, Inc. 2024. CesiumJS. https://cesium.com/cesiumjs/.

[7] P. A. Chou, O. Nakagami, and E. S. Jang. 2017. Point Cloud Compression Test Model for Category 1 V0. document N17223, ISO/IEC JTC1/SC29/WG11 MPEG.

[8] Robert L Cook. 1986. Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG)* 5, 1 (1986), 51–72.

[9] Hong Gao, Efe Bozkir, Lisa Hasenbein, Jens-Uwe Hahn, Richard Göllner, and Enkelejda Kasneci. 2021. Digital Transformations of Classrooms in Virtual Reality. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. ACM, Article 483, 10 pages. https://doi.org/10.1145/3411764.3445596

[10] Enrico Gobbetti and Fabio Marton. 2004. Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics* 28, 6 (2004), 815–826.

[11] Google. [n.d.]. Brotli compression format. https://github.com/google/brotli.

[12] Google. [n.d.]. Draco 3D Graphics Compression. https://github.com/google/draco.

[13] Ayah Hamad and Bochen Jia. 2022. How Virtual Reality Technology Has Changed Our Lives: An Overview of the Current and Potential Applications and Limitations. *International Journal of Environmental Research and Public Health* 19, 18 (2022). https://doi.org/10.3390/ijerph191811278

[14] Bo Han, Yu Liu, and Feng Qian. 2020. ViVo: Visibility-Aware Mobile Volumetric Video Streaming. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (London, United Kingdom) *(MobiCom '20)*. Article 11, 13 pages. https://doi.org/10.1145/3372224.3380888

[15] Yakun Huang, Yuanwei Zhu, Xiuquan Qiao, Zhijie Tan, and Boyuan Bai. 2021. AITransfer: Progressive AI-powered Transmission for Real-Time Point Cloud Video Streaming. In *Proceedings of the 29th ACM International Conference on Multimedia* (Virtual Event, China) *(MM '21)*. ACM, 3989–3997. https://doi.org/10.1145/3474085.3475624

[16] Martin Isenburg. [n.d.]. LASzip: lossless LiDAR compression. https://laszip.org/.

[17] ISO. 2019. Dynamic adaptive streaming over HTTP (DASH). https://www.iso.org/standard/75485.html.

[18] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.* 42, 4 (2023), 139–1.

[19] Maja Krivokuca, Philip A Chou, and Patrick Savill. 2018. 8i voxelized surface light field (8iVSLF) dataset. *ISO/IEC JTC1/SC29/WG11 MPEG, input document m42914* (2018).

[20] Umakant Kulkarni, Khaled Diab, Shivang Aggarwal, Lianjie Cao, Faraz Ahmed, Puneet Sharma, and Sonia Fahmy. 2023. Understanding the Impact of Wi-Fi Configuration on Volumetric Video Streaming Applications. In *Proceedings of the 2023 Workshop on EMS* (New York, NY, USA). ACM, 41–47. https://doi.org/10.1145/3609395.3610599

[21] Umakant Kulkarni, Khaled Diab, Lianjie Cao, Faraz Ahmed, Shivang Aggarwal, Puneet Sharma, and Sonia Fahmy. 2025. Maestro: QoE-Aware Dynamic Resource Allocation in Wi-Fi Networks. *Proc. ACM Netw.* 3, CoNEXT1, Article 4 (March 2025), 24 pages. https://doi.org/10.1145/3709371

[22] Joël Roman Ky, Bertrand Mathieu, Abdelkader Lahmadi, and Raouf Boutaba. 2022. Characterization and troubleshooting of cloud gaming applications on mobile networks. Network Traffic Measurement and Analysis Conference (TMA 2022). https://hal.science/hal-03878114

[23] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. 183–196. https://doi.org/10.1145/3098822.3098842

[24] K. Lee, J. Yi, Y. Lee, S. Choi, and Y.M. Kim. Sept. 2020. GROOT: A Real-time Streaming System of High-Fidelity Volumetric Videos. In *Proc. ACM MobiCom.*

[25] Yu Liu, Puqi Zhou, Zejun Zhang, Anlan Zhang, Bo Han, Zhenhua Li, and Feng Qian. 2024. MuV2: Scaling up Multi-user Mobile Volumetric Video Streaming via Content Hybridization and Sharing. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 327–341.

[26] Mahimahi project. 2019. Mahimahi wireless network traces. https://github.com/ravinet/mahimahi/tree/master/traces.

[27] K. Mammou. 2017. PCC Test Model Category 2 V0. document N17248, ISO/IEC JTC1/SC29/WG11 MPEG.

[28] K. Mammou. 2017. PCC Test Model Category 3 V0. document N17249, ISO/IEC JTC1/SC29/WG11 MPEG.

[29] K. Mammou and P. A. Chou. 2018. PCC Test Model Category 13 V2. document N17519, ISO/IEC JTC1/SC29/WG11 MPEG.

[30] Jiayi Meng, Sibendu Paul, and Y. Charlie Hu. 2020. Coterie: Exploiting Frame Similarity to Enable High-Quality Multiplayer VR on Commodity Mobile Devices. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. 923–937. https://doi.org/10.1145/3373376.3378516

[31] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.

[32] Abhishek kumar Mishra, Sara Ayoubi, Giulio Grassi, and Renata Teixeira. 2021. NemFi: record-and-replay to emulate WiFi. *SIGCOMM Comput. Commun. Rev.* 51, 3 (jul 2021), 2–8. https://doi.org/10.1145/3477482.3477484

[33] Elias Neuman-Donihue, Michael Jarvis, and Yuhao Zhu. 2023. FastPoints: A state-of-the-art point cloud renderer for Unity. *arXiv preprint arXiv:2302.05002* (2023).

[34] Mirko Palmer, Malte Appel, Kevin Spiteri, Balakrishnan Chandrasekaran, Anja Feldmann, and Ramesh K. Sitaraman. 2021. VOXEL: Cross-Layer Optimization for Video Streaming with Imperfect Transmission. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies* (Virtual Event, Germany) *(CoNEXT '21)*. 359–374. https://doi.org/10.1145/3485983.3494864

[35] Mirko Palmer, Thorben Krüger, Balakrishnan Chandrasekaran, and Anja Feldmann. 2018. The QUIC Fix for Optimal Video Streaming *(EPIQ'18)*. 43–49. https://doi.org/10.1145/3284850.3284857

[36] Colin Perkins and Jörg Ott. 2018. Real-Time Audio-Visual Media Transport over QUIC *(EPIQ'18)*. 36–42. https://doi.org/10.1145/3284850.3284856

[37] Point Clouds [n.d.]. Point Clouds: 3D Models. https://sketchfab.com/3d-models/tikal-guatemala-68e7e87ee8664745909518e7b579d3b4.

[38] Point Clouds [n.d.]. Point Clouds: 3D Models. https://sketchfab.com/3d-models/12th-c-ce-water-moon-guanyin-f3316610c8b740c595c22ca1d7df54ca.

[39] Darijo Raca, Dylan Leahy, Cormac J Sreenan, and Jason J Quinlan. 2020. Beyond throughput, the next generation: A 5G dataset with channel and context metrics. In *Proceedings of the 11th ACM multimedia systems conference*. 303–308.

[40] Maximilian Rettinger, Niklas Müller, Christopher Holzmann-Littig, Marjo Wijnen-Meijer, Gerhard Rigoll, and Christoph Schmaderer. 2021. VR-based Equipment Training for Health Professionals. In *CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan). ACM, Article 252, 6 pages. https://doi.org/10.1145/3411763.3451766

[41] Markus Schutz. 2016. *Potree: Rendering Large Point Clouds in Web Browsers.* Master's thesis. Institute of Computer Graphics and Algorithms, Vienna University of Technology. https://www.cg.tuwien.ac.at/research/publications/2016/SCHUETZ-2016-POT/SCHUETZ-2016-POT-thesis.pdf

[42] Markus Schütz, Stefan Ohrhallinger, and Michael Wimmer. 2020. Fast Out-of-Core Octree Generation for Massive Point Clouds. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 155–167.

[43] Markus Schütz, Katharina Krösl, and Michael Wimmer. 2019. Real-Time Continuous Level of Detail Rendering of Point Clouds. In *IEEE VR.* https://doi.org/10.1109/VR.2019.8798284

[44] Randall R. Stewart, Michael Tüxen, and karen Nielsen. 2022. Stream Control Transmission Protocol. RFC 9260. https://doi.org/10.17487/RFC9260

[45] Michael Wand et al. 2007. Interactive Editing of Large Point Clouds.. In *PBG@ Eurographics*. 37–45.

[46] Chendong Wang, Anlan Zhang, Yifan Yang, Lili Qiu, Yuqing Yang, Xinyang Jiang, Feng Qian, and Suman Banerjee. 2025. VoLUT: Efficient Volumetric Streaming Enhanced by LUT-Based Super-Resolution. In *Proceedings of the MLSys 2025.* Santa Clara, CA.

[47] Jianqiang Wang, Dandan Ding, Zhu Li, Xiaoxing Feng, Chuntong Cao, and Zhan Ma. 2022. Sparse tensor-based multiscale representation for point cloud geometry compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 7 (2022), 9055–9071.

[48] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. 2003. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, Vol. 2. Ieee, 1398–1402.

[49] Nan Wu, Kaiyan Liu, Ruizhi Cheng, Bo Han, and Puqi Zhou. 2024. Theia: Gaze-driven and Perception-aware Volumetric Content Delivery for Mixed Reality Headsets. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*. 70–84.

[50] Yaocheng Wu, Huy Vo, Jie Gong, and Zhigang Zhu. 2021. UnityPIC: Unity Point-Cloud Interactive Core. *Parallel graphics and visualisation* (2021).

[51] Dong-Ming Yan, Jian-Wei Guo, Bin Wang, Xiao-Peng Zhang, and Peter Wonka. 2015. A survey of blue-noise sampling and its applications. *Journal of Computer Science and Technology* 30, 3 (2015), 439–452.

[52] Soojeong Yoo, Phillip Gough, and Judy Kay. 2020. Embedding a VR Game Studio in a Sedentary Workplace: Use, Experience and Exercise Benefits. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. ACM, 1–14. https://doi.org/10.1145/3313831.3376371

[53] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. 2022. YuZu: Neural-Enhanced Volumetric Video Streaming. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 137–154. https://www.usenix.org/conference/nsdi22/presentation/zhang-anlan

[54] Anlan Zhang, Chendong Wang, Yuming Hu, Ahmad Hassan, Zejun Zhang, Bo Han, Feng Qian, and Shichang Xu. 2024. Habitus: Boosting Mobile Immersive Content Delivery through Full-body Pose Tracking and Multipath Networking. In *NSDI 24*. 1677–1695. https://www.usenix.org/conference/nsdi24/presentation/zhang-anlan

[55] Ding Zhang, Puqi Zhou, Bo Han, and Parth Pathak. 2023. M5: Facilitating Multi-User Volumetric Content Delivery with Multi-Lobe Multicast over mmWave. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems* (Boston, Massachusetts) *(SenSys '22)*. Association for Computing Machinery, New York, NY, USA, 31–46. https://doi.org/10.1145/3560905.3568540

[56] Jie Zhang, Hang Shi, Yong Cui, Feng Qian, Wei Wang, Kai Zheng, and Jianping Wu. 2022. To Punctuality and Beyond: Meeting Application Deadlines with DTP. In *2022 IEEE 30th International Conference on Network Protocols (ICNP) (ICNP '22)*. 1–11. https://doi.org/10.1109/ICNP55882.2022.9940391