YuZu: Neural-Enhanced Volumetric Video Streaming

Anlan Zhang and Chendong Wang, *University of Minnesota, Twin Cities;*
Bo Han, *George Mason University;* Feng Qian, *University of Minnesota, Twin Cities*

https://www.usenix.org/conference/nsdi22/presentation/zhang-anlan

This paper is included in the Proceedings of the
19th USENIX Symposium on Networked Systems
Design and Implementation.

April 4–6, 2022 • Renton, WA, USA

978-1-939133-27-4

Open access to the Proceedings of the
19th USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by

YuZu: Neural-Enhanced Volumetric Video Streaming

Anlan Zhang[1] *Chendong Wang*[1*] *Bo Han*[2] *Feng Qian*[1]
[1]*University of Minnesota, Twin Cities* [2]*George Mason University*

Abstract

Differing from traditional 2D videos, volumetric videos provide true 3D immersive viewing experiences and allow viewers to exercise six degree-of-freedom (6DoF) motion. However, streaming high-quality volumetric videos over the Internet is extremely bandwidth-consuming. In this paper, we propose to leverage 3D super resolution (SR) to drastically increase the visual quality of volumetric video streaming. To accomplish this goal, we conduct deep intra- and inter-frame optimizations for off-the-shelf 3D SR models, and achieve up to 542× speedup on SR inference without accuracy degradation. We also derive a first Quality of Experience (QoE) model for SR-enhanced volumetric video streaming, and validate it through extensive user studies involving 1,446 subjects, achieving a median QoE estimation error of 12.49%. We then integrate the above components, together with important features such as QoE-driven network/compute resource adaptation, into a holistic system called YuZu that performs line-rate (at 30+ FPS) adaptive SR for volumetric video streaming. Our evaluations show that YuZu can boost the QoE of volumetric video streaming by 37% to 178% compared to no SR, and outperform existing viewport-adaptive solutions by 101% to 175% on QoE.

1 Introduction

Volumetric video is an emerging type of multimedia content. Unlike traditional videos and 360° panoramic videos [28, 53] that are 2D, every frame in a volumetric video consists of a 3D scene represented by a point cloud or a polygon mesh. The 3D nature of volumetric video enables viewers to exercise six degree-of-freedom (6DoF) movement: a viewer can not only "look around" by changing the yaw, pitch, and roll of the viewing direction, but also "walk" in the video by changing the translational position in 3D space. This leads to a truly immersive viewing experience. As the key technology of realizing telepresence [49], volumetric video has registered numerous applications. They can be viewed in multiple ways: through VR/MR (virtual/mixed reality) headsets or directly on PCs (similar to how we play 3D games).

Despite the potentials, streaming volumetric videos over the Internet faces a key challenge of high bandwidth consumption. High-quality volumetric content requires hundreds of Mbps bandwidth [27, 71]. To improve the Quality of Experience (QoE) under limited bandwidth, prior work has mostly focused on viewport-adaptive streaming (*i.e.,* mainly streaming content that will appear in the viewport) [27, 41, 50]. However, they are ineffective when the entire scene falls inside the viewport. They also require 6DoF motion prediction that is unlikely to be accurate for fast motion. Some other proposals explored remote rendering [26, 52] (*e.g.,* having an edge node transcode 3D scenes into regular 2D frames). However, they require not only 6DoF motion prediction, but also edge/cloud-side transcoding that is difficult to scale, as summarized in Table 1.

In this paper, we employ a different and orthogonal approach toward improving the QoE of volumetric video streaming through *3D super resolution* (3D SR). SR was initially designed for improving the visual quality of 2D images [21, 65]. Recently, researchers in the computer vision community developed SR models for point clouds [43, 61, 63, 70]. This inspires us to employ SR for volumetric video streaming, as each frame of a volumetric video is typically either a point cloud or a 3D mesh.[1] Although there have been recent successful attempts on applying SR to 2D video streaming [22, 39, 68], 3D-SR-enhanced volumetric video streaming is unique and challenging due to the following reasons.

- There is a fundamental difference between *pixel-based* 2D frames and volumetric frames consisting of *unstructured 3D points*, making processing volumetric videos (even without SR) vastly different from 2D videos.
- Due to its 3D nature, the computation overhead of 3D SR is very high. We apply off-the-shelf 3D SR models to volumetric videos [1], and find that the runtime performance of 3D SR is unacceptably poor – achieving only ~0.1 frames per second (FPS) on a PC with a powerful GPU. In contrast, 2D SR can achieve line-rate upsampling by simply downscaling the model [68], but we find that only doing model downscaling is far from being adequate for line-rate 3D SR (*i.e.,* at 30+ FPS).
- Given its recent debut, there lacks research on basic infrastructures such as tools and models supporting volumetric video streaming. For example, there is no QoE model for volumetric videos that can guide bitrate adaptation or critical SR parameter selection; the wide range of factors affecting the QoE make constructing such a model quite challenging.
- There are other practical challenges to overcome, such as a lack of color produced by today's 3D SR models.

To address the above challenges, we begin by developing to

* Current affiliation: University of Wisconsin, Madison.

[1] We focus on point-cloud-based volumetric videos in this work, but the key concepts of YuZu also apply to mesh-based volumetric videos.

Schemes	Refs	Advantages (⊕) and Disadvantages (⊖)
Direct Streaming	N/A	⊕ Easy to implement, best QoE (if bandwidth is sufficient). ⊖ Highest network bandwidth (BW) usage.
Direct + VA	[27, 41]	⊕ Lower BW usage. ⊖ BW saving depends on user's motion, QoE depends on motion prediction.
Direct + SR	YuZu	⊕ Good QoE, further lower BW usage, adaptively trades compute resource for BW. ⊖ Requires training.
Remote Rendering	[26, 52]	⊕ Lowest BW usage. ⊖ QoE depends on motion prediction, need edge support (poor scalability).

Table 1: Four categories of volumetric video streaming approaches (VA = Viewport Adaptation; SR = Super Resolution).

our knowledge a first QoE model for assessing SR-enhanced volumetric video streaming. The model takes into account a variety of factors that may affect the QoE, such as video resolution (*i.e.,* point density)[2], viewing distance, upsampling ratio, SR-incurred distortion, and QoE metrics from traditional video streaming. We validate our model by conducting two IRB-approved user studies involving 1,446 voluntary participants from 40 countries, using a major genre of volumetric content, *i.e.,* portraits of single/multiple people. The validation results confirm its accuracy, with a median QoE estimation error of 12.49%. Our user studies offer definitive evidence that 3D SR can significantly boost the QoE of volumetric video streaming.

Next, we design, implement, and evaluate YuZu, which is to our knowledge a first SR-enhanced volumetric video streaming system. At its core, YuZu deeply optimizes the end-to-end upsampling pipeline in three aspects: *intra-frame SR*, *inter-frame SR*, and *network-compute resource management*, whose synergy helps drastically improve the runtime performance of SR while retaining the inference accuracy.

For **intra-frame SR**, our approaches are not limited to generic optimizations for deep learning models such as modifying SR models' structures for fast-paced SR. More importantly, we consider the factors that are unique to 3D SR and its data representation: we design a mechanism that leverages the low-resolution content (*i.e.,* the input to the SR model, which is typically discarded after being fed into the model) to reduce the SR model complexity; we also trim the pre-processing and post-processing stages of 3D SR and tailor them to volumetric video streaming. Note that these optimizations are generic, applicable to all the 3D SR models we have investigated [43, 61, 63, 70].

For **inter-frame SR**, YuZu speeds up SR by caching and reusing 3D SR results across consecutive frames. Realizing that none of the 2D inter-frame encoding techniques can be directly applied to volumetric videos, we design an effective inter-frame content reference scheme for SR-enhanced point cloud streams, followed by robust criteria determining whether SR results can be reused between two frames. We then extend reusing SR results from two to multiple consecutive frames through a dynamic-programming-based optimization. The synergy of the above intra- and inter-frame acceleration schemes fills the huge gap between off-the-shelf 3D SR models' performance and what is required for line-rate upsampling of point cloud streams.

YuZu further performs **network-compute resource management** through making judicious decisions about the quality level of the to-be-fetched content and its upsampling ratio. These two decision dimensions are subject to the dynamic network bandwidth and limited compute resources, respectively, which need to be jointly considered given their complex trade-offs – a unique challenge compared to traditional adaptive bitrate (ABR) video streaming. YuZu takes a QoE-driven approach by maximizing the utility function derived from our QoE model. To solve the underlying optimization problem in real time, we develop a hybrid, two-stage algorithm that employs coarse-grained and fine-grained search at different time to efficiently find a good approximate solution. In addition, YuZu performs fast colorization of SR results through efficient nearest point search.

We implement the above components and integrate them into YuZu in 10,848 lines of code. Our extensive evaluations indicate that YuZu can achieve line-rate, adaptive, high-quality 3D SR. We highlight key evaluation results as follows.

- Our user study suggests that 3D SR can boost the volumetric video QoE by 37% to 178% compared to no SR.
- Our optimizations speed up 3D SR by 140× to 542× and reduce GPU memory usage by 68% to 90% with no accuracy degradation, compared to the vanilla SR models [43, 61].
- Compared to a recently proposed viewport-adaptive volumetric video streaming system [27], YuZu improves the QoE by 100.6% to 174.9%.

To summarize, we make the following contributions.

- We build an empirical QoE model for SR-enhanced volumetric videos, and validate it through large-scale user studies involving 1,446 participants. We build our models using volumetric content of single/multiple human portraits, a major application of volumetric video streaming. Note that the model can be applied to non-SR volumetric videos belonging to the same genre, with an SR ratio of 1.
- We propose and design YuZu, an SR-enhanced, QoE-aware volumetric video streaming system.
- We implement YuZu, and conduct extensive evaluations for its QoE improvement and runtime performance.

2 Background and Motivation

Recently, the computer vision community extended SR to *static* point clouds [43, 61, 63, 70]. When applied to a video v, SR trains offline a deep neural network (DNN) model M that *upsamples* low-resolution frames $L(v)$ to high-resolution ones $H(v)$, using the original (high-resolution) frames $F(v)$ for training. In the online inference, the server sends M and $L(v)$ to the client, which infers $H(v) = M(L(v))$. SR leverages the overfitting property of DNN to ensure that $H(v)$ is highly

[2]The resolution of a point cloud is defined as its point density; the resolution of a volumetric video is the avg. resolution of its point cloud frames.

similar to $F(v)$. It achieves bandwidth reduction (or QoE improvement when bandwidth remains the same) since the combined size of M and $L(v)$ is much smaller than $F(v)$.

We start with a straightforward approach: applying PU-GAN [43], a state-of-the-art 3D SR model, to upsample every point cloud frame of a volumetric video. PU-GAN operates by dividing the entire point cloud of a frame into smaller *patches*, each consisting of a subset of points. Both SR training and inference are performed on a per-patch (as opposed to a per-frame) basis, *i.e.,* each patch is upsampled individually. Its DNN model is based on a generative adversarial network (GAN) and realizes three key stages: feature extraction, feature expansion, and point set generation.

We next describe a case study using PU-GAN to motivate YuZu. Our testing video was captured by three depth cameras. It has 3,622 frames, each consisting of ~100K points depicting a performing actor. We use all its frames to train a PU-GAN model. We set the SR ratio (*i.e.,* upsampling ratio) to 4, making the input and output point clouds consist of roughly 25K and 100K points, respectively. We have both positive and negative findings from this case study. On the positive side, the model can accurately reconstruct each individual frame, *i.e.,* each upsampled point cloud is highly similar to the original one in terms of the geometric structure, as quantified by the Earth Mover's Distance (EMD [54]):

$$\mathcal{L}_{EMD}(I,G) = \min_{\phi:I\to G} \frac{1}{|I|}\sum_{x\in I} ||x-\phi(x)||_2 \qquad (1)$$

where I and G are the upsampled point cloud and the ground truth, respectively; $\phi : I \to G$ is a bijection from the points in I to those in G. The average EMD value across all frames is 1.47cm, which confirms good upsampling accuracy [43]; it is also verified by our IRB-approved user studies (§4.2). Also encouragingly, we find that SR indeed achieves significant bandwidth savings. For this 2-minute video, the compressed sizes of $F(v)$, M, and $L(v)$ are 1.40 GB, 560 KB, and 0.36 GB, respectively, leading to a bandwidth reduction of 74.2%.

Despite the above encouraging results, we notice three major issues from the above case study.

• **A Lack of Quality-of-Experience (QoE) Model.** For traditional 2D video streaming, there exist numerous studies on modeling the viewer's QoE [15, 18, 69]. In contrast, volumetric videos are still in their infancy. There is a lack of generic QoE models that researchers can leverage, not to mention a lack of understanding of how SR impacts QoE.

• **Unacceptably Poor Runtime Performance.** 3D SR models are computationally much more heavyweight than 2D SR models. When applying PU-GAN to the above video, the runtime performance is extremely poor. On a machine with an NVIDIA 2080Ti GPU, the upsampling FPS is only 0.1, far below the desired FPS of at least 30. Besides, the GPU memory usage of PU-GAN is 7GB (out of the 11GB available memory of 2080Ti). This is one reason why all the off-the-shelf 3D SR models operate on a per-patch basis, as this saves memory compared to processing a full frame.

• **No Color Support.** We find that no existing 3D SR model can restore the color information of upsampled point cloud.

Note that the last two limitations are common in that they also apply to all other 3D SR models for point clouds that we have examined, such as MPU [61] and PU-Net [70].

3 YuZu Overview

YuZu is to our knowledge the first SR-enhanced volumetric video streaming system. It streams video-on-demand volumetric content stored on an Internet server to client hosts. On the server side, the volumetric video is divided into *chunks* each consisting of a fixed number of frames (*i.e.,* point clouds encoded by schemes such as Octree [34,46] and k-d tree [35,44]). Each chunk is encoded into multiple versions with different resolutions (*i.e.,* point densities). The SR model training and volumetric content preprocessing (*e.g.,* patch reuse computation, see §5.2) are performed offline on the server side. Similar to a typical DASH server, the YuZu server is stateless (and thus scalable), and all the streaming logic runs on the client side. As shown in Figure 1, the client fetches from the server the video chunks, which can possibly be at a low resolution. Since 3D SR models typically operate on a per-patch basis, the client segments each frame into patches, upsamples them through 3D SR, efficiently colors them (§5.4), and renders them to the viewer.

To achieve line rate SR, YuZu employs novel optimizations tailored to SR-enhanced volumetric video streaming. Regarding *intra-frame optimizations*, off-the-shelf 3D SR models are strategically adapted; low-resolution patches before SR are properly leveraged instead of being discarded; and the patch generation is accelerated (§5.1). For *inter-frame optimizations*, previous SR results are judiciously reused (§5.2).

A crucial decision that YuZu must make is to determine what resolution (quality level) to fetch for each chunk, as well as which SR ratio to apply for upsampling each patch, subject to the resource constraints jointly imposed by the network and computation. YuZu addresses this through a principled, efficient, and QoE-driven discrete optimization framework (§5.3). The framework utilizes a first-of-its-kind QoE model that we derive from ratings of 1,446 real users (§4).

4 QoE Model for Volumetric Videos

For SR-enhanced volumetric video streaming, its QoE is affected by a wide range of factors. The large space formed by these factors and their interplay make constructing QoE models much more challenging than conventional videos.

4.1 An Empirical QoE Model

We first enumerate factors that may affect the QoE for SR-enhanced volumetric video streaming. They are derived based on the domain knowledge of SR and our communication with other volumetric video viewers.

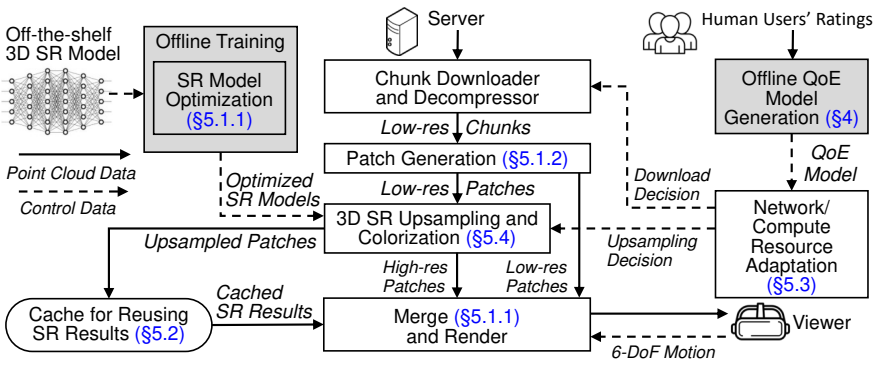

Figure 1: The system architecture of YuZu.

Age	18-25: 21.8%, 26-30: 29.0%, 31-35: 20.4%, 35+: 28.8%
Gender	Male: 60.3%, Female: 39.2%, Other: 0.5%
Country (40 Total)	US: 55.0%, IN: 28.1%, BR: 5.0%, IT: 2.7%, UK: 1.2%, DE: 1.0%, CA: 0.9%, Other: 6.1%
Education	Bachelor: 59.1%, Master: 23.8%, Other: 17.1%

Table 2: Demographics of the 1,446 subjects in our user studies.

• **Point Density**. Similar to 2D image resolution, a 3D object with a higher point density (resolution) contains more details and thus offers a better QoE.

• **Viewing Distance.** As the viewing distance increases, a rendered 3D object becomes smaller in the displayed view, and is thus less sensitive to quality degradation.

• **SR Ratio and Distortion.** A higher SR ratio leads to a higher point density (and thus more QoE gain), but also potentially higher distortions (and thus more QoE loss).

• **Artifacts caused by Patches.** As described in §2, a typical 3D SR model operates by upsampling individual point subsets called patches. If patches within a frame have non-uniform qualities (caused by different SR ratios), the perceived QoE will be affected.

• **Invisibility due to Finite Viewport and Occlusion.** Due to the 3D nature of volumetric videos, a viewer can see only content that is inside the viewport and not occluded. Outside-viewport or occluded content brings no impact on the QoE.

• **QoE Metrics for Regular Video Streaming.** They include factors such as stall and inter-frame quality switches [69].

Next, we develop an empirical QoE model that considers the above factors. Since SR is performed on a per-patch basis, we first model the QoE for each individual patch as:

$$q_{i,j} = g(d_{i,j}, r_{i,j}, \delta_{i,j}) - h(EMD, \delta_{i,j}) \tag{2}$$

where $q_{i,j}$ is the quality of patch j in frame i; $d_{i,j}$ is the patch's original point density before SR; $\delta_{i,j}$ is the viewing distance to the patch; $r_{i,j}$ is the SR ratio of the patch. Eq. 2 has two terms: $g(\cdot)$ considers the patch's perceived density after SR, and $h(\cdot)$ accounts for the QoE penalty incurred by SR distortion, quantified by the viewing distance and the EMD (Eq. 1) between the upsampled patch and the high-quality patch (ground truth). We empirically define $g(\cdot)$ and $h(\cdot)$ as:

$$g(d_{i,j}, r_{i,j}, \delta_{i,j}) = w_1(\delta_{i,j}) \times d_{i,j} \times r_{i,j} \tag{3}$$

$$h(EMD, \delta_{i,j}) = w_2(\delta_{i,j}) \times EMD \tag{4}$$

where $w_1(\delta_{i,j})$ and $w_2(\delta_{i,j})$ are weights parameterized on $\delta_{i,j}$. Intuitively, in Eq. 3, after SR, the perceived point density improves by a factor of $r_{i,j}$; the QoE gain brought by a higher point density after SR (Eq. 3) and the QoE penalty caused by SR distortion (Eq. 4) depend on the viewing distance.

Now given a single frame i, we define its quality Q_i as the average of all its visible patches' quality values:

$$Q_i = \frac{\sum_j v_{i,j} q_{i,j}}{\sum_j v_{i,j}} \tag{5}$$

where $v_{i,j} \in \{0,1\}$ is 1 iff the patch is visible, *i.e.,* it falls inside the viewport and is not occluded by other patches. To account for the artifacts caused by patches, we define *inter-patch quality switch* I_i^{patch} as the quality variation across the visible patches within frame i. To account for inter-frame quality switches, we define *inter-frame quality switch* I_i^{frame} as the quality change from frame $i-1$ to frame i:

$$I_i^{patch} = \textbf{StdDev}\left(\{q_{i,j} | \forall j, v_{i,j} > 0\}\right) \tag{6}$$

$$I_i^{frame} = \|Q_i - Q_{i-1}\| \tag{7}$$

For a volumetric video playback, a possible way to model its overall QoE is a linear combination of Q_i, I_i^{patch}, I_i^{frame}, and I_i^{stall} (the stall of frame i). We choose a linear form that is widely used in 2D Internet videos [69]. Thus, we have

$$QoE = \sum_i Q_i - \sum_i \mu_p(\delta_i) I_i^{patch} - \sum_i \mu_f(\delta_i) I_i^{frame} - \sum_i \mu_s(\delta_i) I_i^{stall} \tag{8}$$

Note that depending on the viewing distance, the weights μ_p, μ_f, and μ_s may differ (*e.g.,* viewers may be more sensitive to stalls when watching a scene at a closer distance), so we parameterize the weights with the viewing distance. In Eq. 8, δ_i summarizes the viewing distances to all the patches in frame i. We empirically choose $\delta_i = (\sum_j v_{i,j}\delta_{i,j})/(\sum_j v_{i,j})$. Also note that the above model is generic and applicable to non-SR-enhanced and non-patch-based volumetric videos as it encompasses special cases without using SR ($r_{i,j}$=1) or patches (I_i^{patch}=0).

4.2 Model Validation through User Studies

We next conduct user studies with two purposes: validating our QoE model and deriving the model parameters. Our QoE model considers many factors as described in §4.1. The high-level approach of the user study is to let participants subjectively rate the QoE for all the combinations of the above factors' different degrees of impairments, and then use the

Scheme	1×1	1×2	1×3	1×4	2×1	2×2	3×1	4×1
Pt. density	25%	25%	25%	25%	50%	50%	75%	100%
SR ratio	-	×2	×3	×4	-	×2	-	-

Table 3: 8 impaired versions (except 4×1) of a video segment. In scheme $m \times n$, m is the point density level and n is SR ratio.

Videos: {*Long Dress*, *Loot* [1]; *Band*, *Haggle* [36]}
Avg. frame quality Q_i: 7 values uniformly selected from Table 3
Avg. distance $dist_{i,j}$: {1m, 2m, 3m, 4m}
Avg. inter-patch switch I_i^{patch}: {0.00,0.45,0.90}
Avg. inter-frame switch I_i^{frame}: {0.00,0.45,0.90}
Avg. stall I_i^{stall}: {0.00,0.01,0.03}

Table 4: The factors and their values selected for model validation.

subjects' ratings to train/validate our QoE model. We obtained IRB approvals for our studies. Instead of performing in-person studies, we conduct both studies online by letting users watch pre-generated videos capturing the rendered viewports (with impairments). We take this approach because: (1) it allows vastly scaling up the study, (2) it helps get diverse users worldwide, and (3) the IRB forbids in-person user studies during COVID-19. We have collected responses from 1,446 subjects, whose demographics are shown in Table 2.

We start by studying the QoE gain brought by SR. We have collected 512 subjects' responses with a total number of 57,344 ratings. The key finding is that SR can effectively boost the QoE. For example, at 1m, compared to 1×1, the (user-rated) QoE increases by 37%, 75%, 150% for 1×2, 1×3, and 1×4, respectively; 2×2 improves the QoE by 178% compared to 2×1. The details can be found in Appendix A.

Next, we validate the overall QoE model (Eq. 8). We choose four videos: *Long Dress* showing a dancing female, *Loot* showing a speaking male, *Band* showing three people playing instruments, and *Haggle* showing three people debating. *Long Dress* and *Loot* are obtained from the 8i dataset [1], each consisting of 800K points per frame for 10 seconds. *Band* and *Haggle* are from the CMU Panoptic dataset [36], each consisting of 300K and 100K points per frame, respectively; we select 10-second segments for our study. For each video, we create 8 versions listed in Table 3. Note that since the participants need to watch a large number of impaired copies, the video length (10 seconds) has to be short. Also note that the videos have different point densities, as we want to make the QoE model generic, applicable to different resolutions. We will experimentally verify this shortly. We use our optimized PU-GAN algorithm (details in §5.1) to perform upsampling and create video clips at 4K resolution for four viewing distances: 1m, 2m, 3m, and 4m, which are determined from a separate IRB-approved user study whose details are described in Appendix B. To maintain a fixed viewing distance d, we display the viewport at d meters in front of and facing the viewer. We design a survey using Qualtrics [11] and publish it on Amazon Mechanical Turk (AMT) [2].

We study the impact of all the factors in Eq. 8 on the QoE. Table 4 lists them and their impairment levels. They lead to a total of 756 combinations for each video segment. Since letting subjects perform $\binom{756}{2}$ pairwise comparisons is infeasible, for each combination, we generate one video clip by putting the impaired version and the high-quality "ground truth" version (4×1, $I_i^{patch} = I_i^{frame} = I_i^{stall} = 0$, same viewing distance) side by side, in a random order. To generate the impaired version, we randomly add perturbations to the patches' quality levels to match the corresponding I_i^{patch} and I_i^{frame} values, and randomly inject stalls to match I_i^{stall}. We then ask each subject to watch 100 randomly selected video clips from the 756 clips of a randomly selected video segment. After watching each clip, the subject is asked to rate which side provides a better QoE through 7 choices ("left looks {much better, better, slightly better, similar to, slightly worse, worse, much worse} than right") If the impaired version is {similar to, slightly worse, worse, much worse} than the ground truth, we give the impaired version a score of {3,2,1,0}, respectively.

We have collected 934 subjects' responses with a total number of 93,400 ratings for the above survey published on AMT. For each viewing distance, we use the subjects' ratings to calculate the average score of each of the 756 impaired clips on a scale from 0 to 3, and use it as the QoE ground truth. We then perform 10-fold cross-validation to validate our QoE model (Eq. 8, trained using multi-variable linear regression) for each viewing distance. Figure 2 plots the CDF of the QoE prediction errors at each viewing distance. The median prediction error for 1m, 2m, 3m, 4m is 11.4%, 12.2%, 12.8%, and 12.9%, respectively. The (Person, Spearman) correlation coefficients between the ground-truth QoE score and the predicted QoE score are also high: (0.89, 0.89) at 1m, (0.87, 0.88) at 2m, (0.87, 0.88) at 3m, and (0.85, 0.85) at 4m.

The above QoE models are trained from all four videos. Table 5 shows the Spearman correlation coefficients between the ground-truth QoE and *cross-video* prediction results. We use the data of three videos to train a QoE model and use it to predict the QoE for the remaining video. The results indicate that the same QoE model and its parameters are applicable to volumetric content of the same genre (portraits of people – a major application of volumetric streaming – in our case). We also confirm that most parameters trained from different video segments are indeed quite similar, in spite of the segments' different point densities. When applied to other genres, the model's parameters may differ, as to be explored in our future work (the same happens to 2D videos [68]). Table 6 lists our final model's parameters trained using the entire dataset. The model will be used by YuZu.

5 System Design of YuZu

We now detail the system design of YuZu (Figure 1) that addresses the challenges we identified in §2.

5.1 Accelerating SR Upsampling

To accelerate 3D upsampling, we take a principled approach by exploring three orthogonal directions:

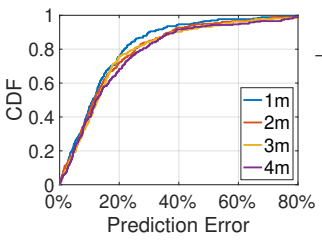

Figure 2: QoE prediction error using our model.

δ	*D*: *Long Dress*; *L*: *Loot*; *B*:*Band*; *H*:*Haggle*			
	DBH ⇒ *L*	*LBH* ⇒ *D*	*DLB* ⇒ *H*	*DLH* ⇒ *B*
1m	0.80	0.74	0.86	0.85
2m	0.76	0.71	0.87	0.87
3m	0.80	0.73	0.83	0.87
4m	0.78	0.71	0.76	0.80

Table 5: Spearman correlation coefficient between QoE ground truth and cross-video prediction. $XYZ \Rightarrow W$ means using the model trained from videos X, Y, and Z to predict video W's QoE.

δ	*Long Dress* + *Loot* + *Band* + *Haggle*				
	w_1	w_2	μ_p	μ_f	μ_s
1m	0.55	27.80	0.52	0.40	170.5
2m	0.42	39.83	1.05	0.91	149.8
3m	0.27	26.63	1.23	1.04	176.7
4m	0.16	17.17	0.47	0.06	304.1

Table 6: Parameters of the final model used in YuZu.

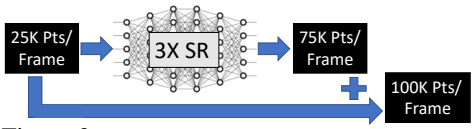

Figure 3: Using a 3× SR model to realize 4× SR.

• **Model Optimization.** How to simplify the upsampling logic while retaining the inference accuracy? (§5.1.1)
• **Data Reduction.** How to strategically feed less data to SR models with negligible impact on QoE? (§5.2)
• **Pre-processing and Post-processing Trimming.** How to simplify the sophisticated pre- and post-processing stages without incurring side effects on inferences? (§5.1.2)

Our optimizations can apply to all 3D SR models we have investigated [43,61,63,70] and they are video-agnostic. In §7, we demonstrate the optimization results for two SR models: PU-GAN [43] and MPU [61].

5.1.1 SR Model Optimization

We take a "top-down" approach by first optimizing the model as a whole and then fine-tuning its detailed structure. For most machine learning models (including 2D SR), after performing an inference, the input is no longer needed and will be discarded. Our investigated 3D SR models [43,61,63,70] make no exception. We instead make a fundamental observation regarding 3D point clouds. Different from a 2D image, a point cloud is a set of unstructured points, which means that point clouds can be *merged via a simple set union operation*. We also note that 3D SR's output points *refine and differ from* the input. Based on this key insight, we propose a simple yet effective optimization: YuZu merges the input low-density point cloud with the SR output in order to improve the visual quality, or to reduce the computation overhead while maintaining the same upsampling ratio. For example, as shown in Figure 3, to achieve 4× upsampling, instead of using a 4× SR model, we can use a (computationally more efficient) 3× SR model and merge the input with the output. Since SR exploits the overfitting nature of DNN, the spatial distributions of upsampled points and the ground truth are expected to be highly similar. By leveraging the input data and downgrading the SR ratio from 4× to 3×, we can achieve an acceleration of up to ~35% without hurting the SR accuracy (Figure 6). Note that in offline training, the loss function is computed *after* merging the input low-density point cloud with the SR output. This makes the trained models aware of and adaptive to the merging process, improving the upsampling accuracy compared to computing the loss function before that.

Next, we explore modifying 3D SR model's DNN structure for inference acceleration. By profiling the inference time of PU-GAN, we find that its three stages, feature extraction, feature expansion, and point set generation, take 78.3%, 19.3%, and 2.4% of execution time, respectively (4× SR). Within the feature extraction stage that dominates the runtime overhead, most operations are *convolutions*. We make the same observation for other 3D SR models that we investigated [61,63,70].

To accelerate convolutions, we replace the original feature extraction, which (*e.g.,* in the case of PU-GAN) enhances the solution in PointNet++ [51] through dynamic graph convolution [56], with a recent proposal called spherical kernel function (SKF) [42]. SKF partitions a 3D space into multiple volumetric bins and specifies a learnable parameter to convolve the points in each bin. In contrast to continuous filter approaches (*e.g.,* multilayer perceptron) used in existing SR models, SKF is a *discrete* metric-based spherical convolutional kernel, and is thus computationally attractive for dense point clouds. Moreover, it is applicable to all the 3D SR models we examined. We find that SKF brings no degradation to the upsampling accuracy (§7.3). One reason may be that the kernel asymmetry of SKF facilitates learning fine geometric details of point clouds [42].

In addition to utilizing SKF, we conduct layer-by-layer profiling [22,66] to fine-tune the SR model's performance-accuracy tradeoff. Take PU-GAN as an example. We remove the last two dense layers of feature extraction and several heavyweight convolution layers in the feature expansion stage, as they make limited contributions to the upsampling accuracy. We also judiciously remove a small number of expanded features to reduce the GPU memory footprint. For other 3D SR models, their model tuning follows a similar approach.

5.1.2 Trimming Pre- and Post-Processing

Recall from §2 that to ensure a manageable model complexity, a 3D SR model divides a point cloud into small patches as basic units for upsampling. We discover that as an important pre-processing step, the patch generation process incurs a high overhead. For example, PU-GAN generates the patches by applying kNN to the seeds created by downsampling. Since the generated patches may overlap, after upsampling, PU-

GAN needs to perform post-processing: it applies the furthest point sampling [48] to remove duplicated points.

To mitigate the above overhead, YuZu adopts a simple patch generation method. It divides the space into cubic cells, and assigns each non-empty cell (*i.e.,* a cell that contains points) to a patch. Compared to the default patch generation approaches used by PU-GAN and other 3D SR frameworks [43, 61], our approach runs very fast; it also brings no overlap among patches, thus eliminating the post-processing step (*i.e.,* overlap removal). In addition, the patches now have a simple geometry shape, so that they can be indexed, searched, and manipulated at runtime. Meanwhile, We find that our patch generation approach does not sacrifice the upsampling accuracy and may even improve the accuracy compared to vanilla PU-GAN and MPU (§7.3). This is likely because cubic cells provide a more consistent structure for the patches, making it easier to perform SR. We also investigate several other patch generation methods based on Voronoi diagram [24] and 3D SIFT [55], but none outperforms our cubic-cell-based approach from either the performance or the accuracy perspective.

5.2 Caching and Reusing SR Results

Videos usually exhibit similarities across frames. We find that volumetric videos make no exceptions. This indicates rich opportunities for caching and reusing SR results.

At a high level, YuZu reuses SR results based on the similarity between patches, which is the basis of inter-frame encoding. Inter-frame similarity has been extensively studied and exploited in 2D videos. However, none of the 2D inter-frame encoding techniques can be directly applied to volumetric videos due to the fundamental difference between pixel-based 2D frames and volumetric frames consisting of unstructured points. There are very few studies on 3D inter-frame encoding [37, 46]; they are incompatible with YuZu's patch-based upsampling, and incur high complexity hindering line-rate decoding. Due to the above reasons, we design our own SR caching/reusing algorithm. Our algorithm is agnostic of and orthogonal to a specific SR model.

YuZu reuses 3D SR results on a per-patch basis to match the patch-based upsampling procedure. Recall from §5.1.2 that YuZu generates patches using 3D cubic cells. Let $p(i,j)$ denote patch j of frame i, and let $N(i,j)$ denote the number of points in $p(i,j)$. YuZu allows reusing the SR result of $p(i,j)$ for subsequent *consecutive patches at the same location*, *i.e.,* $p(i+1,j), p(i+2,j)$, and so on. YuZu restricts reusing patches only at the same location due to two considerations. First, we empirically observe that most patch similarities indeed occur at the same cell location; this makes the benefits (in terms of reduced SR overhead) of reusing a patch belonging to a different cell marginal. Second, allowing reusing a patch at a different cell will drastically increase the overhead of pre-computing the caching/reusing decisions.

We now describe YuZu's SR reuse algorithm. YuZu first determines offline the similarity of two patches. For each patch pair $(p(i,j), p(i+1,j))$, YuZu computes a Weighted Complete Bipartite Graph [17] $B: p(i,j) \rightarrow p(i+1,j)$, which we find to be suitable for dealing with unstructured points. In the bipartite graph, there is a directed edge from every point in $p(i,j)$ to every point in $p(i+1,j)$, and the weight of the edge is their Euclidean distance. We then calculate the minimum-weight matching (MWM) [57] for the graph, *i.e.,* finding $N(i,j)$ edges such that (1) these edges share no common vertices (points), and (2) the sum of their weights is minimized. Intuitively, the MWM identifies a transformation from $p(i,j)$ to $p(i+1,j)$ with a minimum moving distance for the points. The Hungarian algorithm [17] that computes the MWM has a complexity of $O(N^4)$ where $N = \max\{N(i,j), N(i+1,j)\}$. We instead employ a faster $O(N^2)$ approximation algorithm that is found to work well in practice.[3]

We call every edge in the MWM a point motion vector (PMV). A PMV differs from a 2D video's motion vector, which represents a macroblock in a frame based on the position of the same or a similar macroblock in another reference frame. Leveraging the PMVs, we determine that $p(i+1,j)$ and $p(i,j)$ are *similar* if three criteria are satisfied. (1) $N(i,j)$ and $N(i+1,j)$ differ by no more than η_n%; (2) the average length of all the PMVs is smaller than η_a; (3) the top 90-percentile of the shortest PMV is smaller than η_v. These three criteria dictate that $p(i,j)$ and $p(i+1,j)$ have a similar number of points, and the points' collective motions are small. Figure 4 shows how η_a impacts EMD and the patch reuse ratio (% of patches that can reuse a previous SR result). As shown, increasing η_a increases the reuse ratio, but meanwhile decreases the accuracy. According to Figure 4, we set η_a to 0.01m to balance the performance and accuracy. Using similar methods, we empirically set η_n=10 and η_v=0.01m.

Next, we consider how to reuse an SR result across multiple patches belonging to consecutive frames. We define $sim_j(i_1,i_2) \in \{0,1\}$ to be 1 if and only if $p(i_1,j)$ and $p(i_2,j)$ are similar, *i.e.,* satisfying the above three criteria where $i_2 > i_1$. Figure 5 shows an example of 6 consecutive patches at location j where $\forall 1 \le x < y \le 6 : sim_j(x,y) = 0$ except that $sim_j(1,2)$, $sim_j(2,3)$, $sim_j(2,4)$, and $sim_j(2,6)$ are 1. YuZu allows a patch's SR result to be reused across *consecutive* patches if they are all similar to the first patch. For example, Patches 3 and 4 can reuse Patch 2's SR result. However, YuZu does not let Patch 6 reuse Patch 2 because $sim_j(2,5) = 0$. We make this design decision for two reasons. First, we observe that non-consecutive patches are unlikely to be similar in real volumetric videos. Second, supporting non-consecutive reuse requires computing $sim_j(x,y) \forall x < y$, making offline video processing slow.

We develop an algorithm that *minimizes the number of*

[3] The approximation algorithm sorts all the edges by their weights in ascending order. It then adds the edges to the MWM in that order and skips edges that share points with an existing edge in the matching, until every point in $p(i,j)$ or every point in $p(i+1,j)$ is in the MWM.

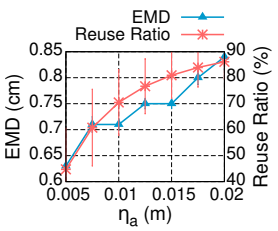

Figure 4: Impact of η_a (using the video in §2, 1×4 SR).

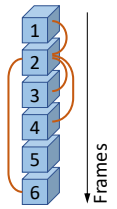

Figure 5: Reusing SR results across consecutive frames.

patches to be upsampled, to boost the online SR performance. For example, in Figure 5, the minimum number of patches to be upsampled is 4: Patches 1, 2, 5, and 6. YuZu efficiently and optimally solves this through dynamic programming (DP). Given n patches $p(1,j),...,p(n,j)$ and their sim_j information, let $u(i,j)$ be the minimum number of patches that need to be upsampled in $\{p(i,j),...,p(n,j)\}$ if we decide to upsample $p(i,j)$. Then $u(i,j)$ can be derived through DP as:

$$u(i,j) = \min\left\{u(i+1,j), \min_{i<k\le n:\forall i<t\le k: sim_j(i,t)=1}\{u(k+1,j)\}\right\}+1 \quad (9)$$

The RHS of Eq. 9 examines each patch following $p(i,j)$ and updates $u(i,j)$ if stopping reusing $p(i,j)$ at $p(k+1,j)$ yields a better $u(i,j)$. The search continues until hitting a patch that is not similar to $p(i,j)$. Eq. 9 can be solved backwards starting from $u(n+1,j)=0$. The solution is $u(1,j)$.

Since YuZu streams VoD volumetric content, all the above logic (calculating MWM, sim_j, and DP) is performed *offline* for each patch location j. Thus, there is no runtime overhead. The SR reuse decisions are sent to the client as meta data, which is only 0.5KB per frame for our testing video in §2.

5.3 Network/Compute Resource Adaptation

YuZu adapts to not only the fluctuating network condition (similar to the job of traditional bitrate adaptation algorithms [45, 64, 69]), but also the available compute resource, due to the high computation overhead of 3D SR. More importantly, these two dimensions incur a tradeoff: given a fixed playback deadline, should YuZu download high-resolution content, or download lower-resolution content and spend time upsampling it? Fortunately, our QoE model (§4.1) dictates how to quantitatively balance this tradeoff.

We first formulate an online network/compute adaptation problem. The video is divided into n chunks each consisting of f frames. To achieve fine-grained adaptation, each chunk is further spatially segmented into b *blocks* (*e.g.,* b=5^3), which are the atomic scheduling units in YuZu's adaptation algorithm. Each block consists of multiple patches (recall from §5.1.2 that each patch occupies a cubic cell). At runtime, YuZu considers all the blocks belonging to a finite horizon of the next w chunks, and searches for their quality and SR ratio assignments that maximize the QoE defined in Eq. 8. This formulation extends the model predictive control (MPC) scheme [69] that proves to be effective for traditional 2D video streaming. The solution space is $O(8^{wb})$ (the 8 possible assignments are listed in Table 3).

We consider how to efficiently solve the above discrete optimization problem. An exhaustive search is clearly infeasible. Due to the large solution space, even the memorization approach (FastMPC [69]) is not practical. Another possibility is a learning-based approach such as Pensieve [45]. However, it requires offline training and may incur a non-trivial inference overhead. Moreover, a recent work [64] indicates that reinforcement learning based bitrate adaptation solutions do not necessarily outperform simple buffer-based approaches [33].

To overcome the above challenges, we develop a lightweight approximation algorithm. It executes in two stages: first determine the quality and SR ratios of to-be-downloaded chunks, and then fine-tune the SR ratios before upsampling. Specifically, in the first stage, *before downloading each chunk*, YuZu performs a *coarse-grained search* by assuming that all the blocks in each chunk have the same quality/SR-ratio assignment. The rationale is that, at this moment, the playback deadline is still far away (compared to Stage 2), and thus the network/computation-load uncertainty diminishes the benefits brought by a block-level, fine-grained search. Meanwhile, this reduces the solution space from $O(8^{wb})$ to $O(8^w)$. Specifically, we (1) start with a quasi-optimal solution obtained from an even coarser-grained search at the granularity of every two consecutive chunks, and (2) perform pruning by bounding [19]. After the above two optimizations, for a practical w (*e.g.,* w=10), the search time (for maximizing the QoE in Eq. 8) becomes negligible compared to the downloading and upsampling time. To estimate I_i^{stall} in Eq. 8, at runtime, YuZu continuously estimates (1) the network bandwidth using the method in [29] and (2) the local processing time of a frame using EWMA-based estimation.

The second stage takes place *before upsampling each frame*. At this stage, the playback deadline gets closer and thus a *block-level, fine-grained search* would be beneficial. To reduce the search complexity, YuZu employs Simulated Annealing (SA) [40] – a probabilistic, greedy approach that approximates the global optimum. Our algorithm begins with setting all the blocks' SR ratios to the lowest (no SR). For each block, the algorithm tries to increase its SR ratio by one level. If the resulting QoE of the finite horizon increases, this change is always accepted; otherwise, we may still accept this change with a probability of $exp(-\frac{\Delta}{t})$, where Δ is the decrease of the QoE and t is the current number of iterations, to avoid a potential local maximum. To speed up the SA algorithm, we reduce the finite horizon to two frames: the previous frame and the current (to be upsampled) frame – we empirically find that conducting frequent adaptations with a short horizon at a per-frame basis outperforms infrequent adaptations with a long horizon at a per-chunk basis in terms of the QoE.

5.4 Coloring SR Results

As described in §2, none of the 3D SR models we investigated performs colorization. There are two high-level approaches for colorization. One is augmenting the SR models by adding the color component. This may yield good colorization results, but at the cost of significantly increasing the SR workload. Given this concern, YuZu takes a much more lightweight approach: approximating each upsampled point's color using the color of the nearest point in the low-density point cloud (*i.e.,* the input to the SR model). In Appendix C, we present the details of our method and experimentally confirm that it can indeed produce good visual quality (with a PSNR >38).

6 Implementation

We integrate all the components in §5 into YuZu, a holistic system as shown in Figure 1. Our implementation consists of 10,848 lines of code (LoC), with 8,326 LoC for the client.

For offline SR model training, we modify the source code of PU-GAN [10] and MPU [8] using TensorFlow 1.14 [13] and custom TensorFlow operators from SPH3D-GCN [12]. Our pre-trained models are saved in the ProtoBuf format [9] that is language- and platform-neutral, facilitating future reuse. For online streaming, we implement the client player on Linux in C++. We use the Draco Library [4] for encoding and decoding the point cloud data. We employ Bazel [3] to compile the TensorFlow 1.14 C/C++ library and use the compiled library to load and execute the SR models. The client *pipelines* content fetching (network-bound), point cloud decoding & patch generation (CPU-bound), 3D SR (GPU-bound), and colorization (CPU-bound) of different frames for better performance. The server is also built in C++, with a custom DASH-like protocol over TCP for client-server communication.

7 Evaluation

7.1 Experimental Setup

Volumetric Videos. We use four point-cloud-based volumetric videos throughout our evaluations. (1) Our own video. We capture a volumetric video by ourselves using 3 synchronized depth cameras. It has 3,622 frames (2 min) each consisting of ~100K points. We refer to this video as *Lab*. We have used it to motivate YuZu in §2. (2) The Long Dress (*Dress*) and *Loot* videos (§4.2). They have 300 frames (10 sec) each consisting of ~100K points. Since they are short, we loop them (with cold caches) 10 times in our evaluations. (3) The *Haggle* video (§4.2). It has 7,800 frames (4'20") each consisting of ~100K points. For all four videos, the eight possible resolution/SR-ratio assignments are listed in Table 3. For each video, we train their SR models separately. All the videos are at 30 FPS, encoded by Draco [4]. Unless otherwise mentioned, the results reported in the remainder of this section are generated using all four videos. The average encoded bitrate of *Lab*, *Dress*, *Loot*, and *Haggle* (4×1) are 96, 108, 118, and 118 Mbps, respectively.

M_1	The vanilla 3D SR model (PU-GAN and MPU)
M_2	M_1 and optimizing patch generation
M_3	M_2 and layer profiling & pruning
M_4	M_3 and applying the spherical kernal function (SKF)
M_5	M_4 and merging SR input with SR output
M_6	M_5 and caching/reusing SR results

Table 7: SR acceleration methods (cumulative).

3D SR Models. We apply our developed model acceleration techniques to two recently proposed 3D SR models: PU-GAN [43] and MPU [61]. The two models usually yield qualitatively similar results, so we show the results of PU-GAN by default. For certain SR-specific experiments (*e.g.,* SR acceleration), we show both models' results. The models are trained on a per-video basis. For each video, the total size of all its models (×2, ×3, and ×4) is around 1.25 MB.

Metrics and Roadmap. We thoroughly evaluate YuZu in terms of performance, QoE, and resource utilization. §7.2 evaluates the QoE improvement brought by our 3D SR optimizations using both subjective (*i.e.,* real-user ratings) and objective (*e.g.,* PSNR [30]) metrics. §7.3 focuses on the performance gain of our 3D SR optimizations, from the perspectives of resource usage, inference time, and upsampling accuracy. §7.4 and §7.5 evaluate the end-to-end performance (*e.g.,* QoE and data usage) of YuZu. §7.6 provides additional micro benchmarks.

Network Conditions. We consider the following network conditions that are readily available in today's wired and wireless networks. (1) *Wired network with stable bandwidth* (*e.g.,* 50, 75, and 100 Mbps) and ~10ms RTT. (2) *Fluctuating bandwidth* captured from real LTE networks. We collect 12 bandwidth traces from a major LTE carrier in multiple U.S. states at diverse locations (campus, malls, streets, *etc.*). Across the traces, their average bandwidth varies from 33.7 to 176.5 Mbps, and the standard deviation ranges from 13.5 to 26.8 Mbps. We use `tc` [6] to replay these traces (with a 50ms base RTT typically observed in LTE [38]). (3) We also conduct *live LTE experiments* at 9 diverse locations in a U.S. city where the average bandwidth varies from 41.1 to 52.4 Mbps and the standard deviation is between 16.6 and 20.7 Mbps.

Devices. We use a commodity machine with an Intel Core i7-9800X CPU @ 3.80GHz and 32GB memory as the YuZu server. We use three client hosts: (1) a desktop with an Intel Core i9-10900X CPU @ 3.70GHz, an NVIDIA GeForce RTX 2080Ti GPU, and 32GB memory (the default client used in our evaluations); (2) a desktop with the same CPU, an NVIDIA GeForce GTX 1660Ti GPU, and 32GB memory; (3) an NVIDIA Jetson TX2 embedded system board with a Pascal-architecture GPU of 256 CUDA Cores, 8GB memory, and a quad-core CPU. They represent a typical high-end PC, a medium-class PC, and a mobile device, respectively.

User Motion Traces. We collect 32 users' 6DoF motion traces when watching the four videos, and replay them in some experiments. The details about how we collect the motion traces can be found in Appendix B.

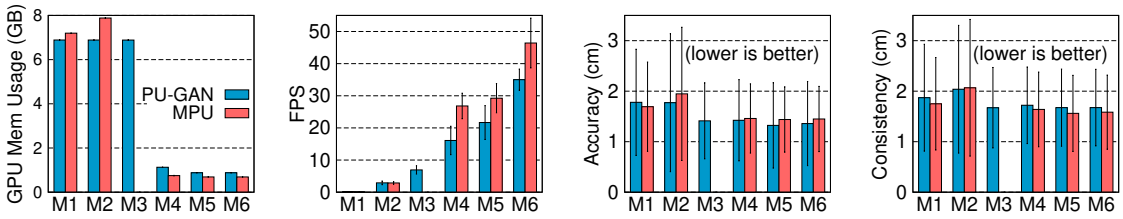

Figure 6: Memory usage, upsampling FPS, upsampling accuracy, and visual consistency of **M1** to **M6** (2080Ti desktop).

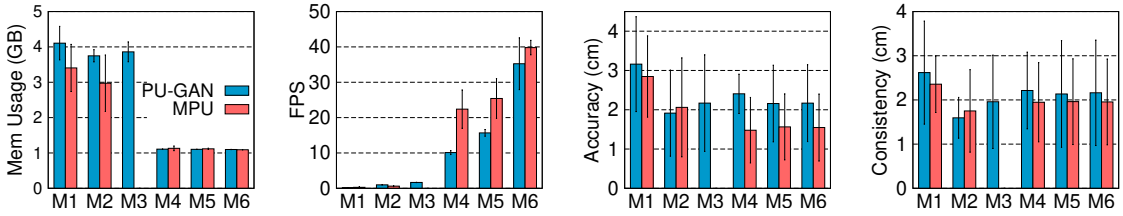

Figure 7: Memory usage, upsampling FPS, upsampling accuracy, and visual consistency of **M1** to **M6** (Jetson TX2 board).

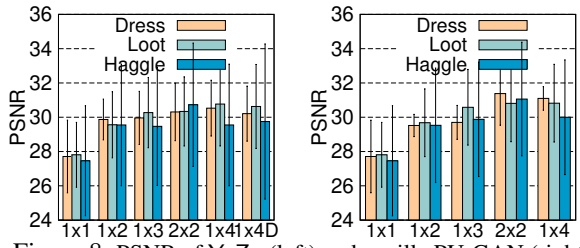

Figure 8: PSNR of YuZu (left) and vanilla PU-GAN (right).

7.2 SR Quality

Subjective Ratings. Recall that in our user studies, we ask our participants to rate the SR results generated by our optimized SR scheme (§5.1). Figure 15 shows that SR brings a significant boost to the user-perceived QoE. For example, at 1m, compared to 1×1, the user-rated QoE increases by 37%, 75%, and 150% for 1×2, 1×3, and 1×4, respectively; 2×2 improves the QoE by 178% compared to 2×1 (§4.2).

Objective Metric. We also examine how SR improves PSNR [30], an objective metric of image quality. The methodology is as follows. We replay the 32 users' 6DoF motion traces of watching the videos under different SR settings, and save the rendered viewports as images $\{I_{SR}\}$. We then repeat the above process using the original videos (4×1), and capture the viewport images $\{I_{4\times1}\}$. We compute the PSNR values by comparing each image in $\{I_{SR}\}$ with its corresponding image in $\{I_{4\times1}\}$. Figure 8 (left) shows the PSNR values for 1×1, 1×2, 1×3, 2×2, 1×4, and 1×4 with reusing SR results (denoted as "1×4D") across all the captured viewports. We notice a significant increase of PSNR from 1×1 to 1×2. The PSNR also increases marginally from 1×2 to 1×4. Meanwhile, the PSNR change between 1×4 and 1×4D is negligible, indicating that caching and reusing SR results brings little impact on the perceived video quality (but drastic performance gain as shown in §7.3). The results of *Lab* are similar. Note that a PSNR value over 30 typically indicates good visual quality [22, 58]. Figure 8 (right) shows the PSNR values for the unmodified PU-GAN model. The qualitatively similar results between the left and right plots of Figure 8 indicate that our SR acceleration modifications sacrifice little visual quality. Note the above results include the colorization step, which is described and separately evaluated in Appendix C.

Comparing Figure 15 and Figure 8, we notice disparities between users' QoE ratings and PSNR values. This indicates that image qualities of rendered 2D content do not directly reflect the perceived QoE of volumetric content. This is a key reason for developing the QoE model for volumetric videos.

7.3 SR Performance Breakdown

We now take a closer look at the effectiveness of each of our proposed methods for accelerating SR. As listed in Table 7, M_1 denotes the vanilla 3D SR model as the comparison baseline; M_2 to M_6 are our proposed SR acceleration methods in §5.1 and §5.2. They are presented in a *cumulative* fashion, *i.e.,* M_i includes every feature of M_{i-1} plus some new feature. The experiments are conducted using two 3D SR models (PU-GAN [43] and MPU [61]), 100Mbps wired network, 4× SR, with network/compute resource adaptation (§5.3) disabled.

Figures 6 and 7 show the results of PU-GAN and MPU on the PC (2080Ti) and Jetson TX2 board, respectively. On the Jetson board, due to its low compute power (and mobile devices' small screen size), we reduce the original video's resolution from 100K to 20K points per frame (*i.e.,* the SR is from 5K to 20K points per frame). We consider four metrics: (1) maximum GPU memory usage (on Jetson TX2 we measure the system memory shared by GPU and CPU), (2) average upsampling speed (in FPS), (3) inference accuracy measured in EMD between each upsampled frame and the ground truth (4×1), and (4) visual consistency measured in EMD between each consecutive pair of upsampled frames.

As shown, on 2080Ti, for PU-GAN (MPU), compared to M_1, M_6 reduces the GPU memory usage by 87% (90%), accelerates the upsampling by 307× (542×), improves the average upsampling accuracy by 24% (14%), and slightly improves the consistency. Also, each optimization (M_2 to M_6) indi-

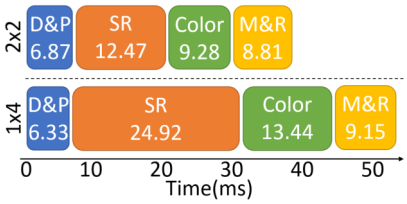

Figure 9: Frame processing time breakdown. D&P: decoding and patch generation; SR: upsampling; Color: colorization, M&R: merging and rendering.

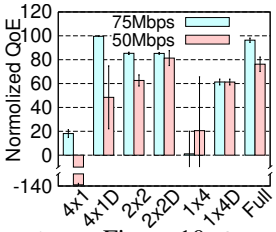

Figure 10: QoE over stable bandwidth ("D"=caching & reusing SR results).

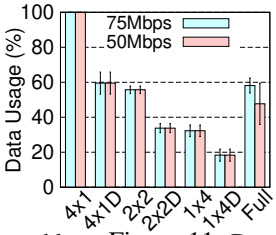

Figure 11: Data usage over stable bandwidth.

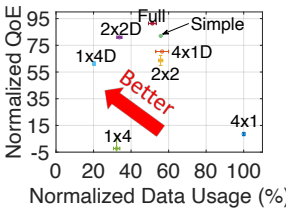

Figure 12: QoE vs. Data usage over fluctuating bandwidth (LTE traces).

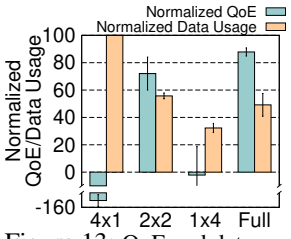

Figure 13: QoE and data usage over live LTE networks.

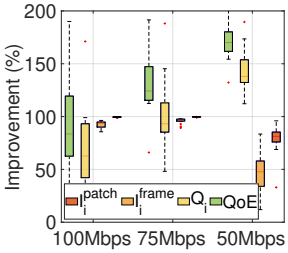

Figure 14: YuZu over ViVo.

vidually improves the upsampling speed and possibly other metric(s). The Jetson setup shows a similar trend. The two models (PU-GAN and MPU) we studied exhibit similar performance gains as we progressively apply our optimizations, except that MPU is less sensitive to M_5. This is because of the network structure difference between PU-GAN and MPU. Note that we do not apply M3 to MPU because our layer-by-layer profiling (§5.1.1) reveals there is no layer that only makes a marginal contribution to the overall upsampling accuracy in the MPU model.

Latency Breakdown. Figure 9 shows the latency breakdown of processing an average frame using PU-GAN (*Lab* video, wired 100Mbps, 2080Ti desktop) under two settings: 2×2 and 1×4. As shown, SR remains the most time-consuming component. The breakdown for MPU is similar. The above results indicate the importance of SR acceleration.

7.4 Diverse Network Conditions

We evaluate the QoE of YuZu under different network conditions, using the four videos and the associated motion traces.

Stable Bandwidth. We first consider two stable bandwidth: 50Mbps and 75Mbps. Under each bandwidth profile, we run the full-fledged YuZu ("Full") and six statically configured YuZu instances: 4×1, 2×2, and 1×4 with and without SR result reusing. The QoE results are shown in Figure 10. We make several observations. First, when the bandwidth is low (50Mbps), 4×1 (without SR) gives the lowest (and even negative) QoE. This is because the limited bandwidth leads to high *network-incurred* stall when fetching high-resolution content; SR can effectively improve the QoE by using computation to compensate for the low bandwidth. Second, when the bandwidth increases to 75Mbps, 1×4 gives the lowest QoE due to the distortion and *computation-incurred* stall due to the high SR ratio. Instead, when the bandwidth is sufficient, the player should fetch the content with a higher quality (*e.g.,* 4×1D). Third, caching and reusing (C&R) the SR results improves the QoE when either the bandwidth is low (*e.g.,* 4×1 at 50Mbps), or the SR ratio is high (*e.g.,* 1×4). Under these two scenarios, C&R reduces the network and compute resource usage, respectively. The saved resources can be used to improve the content quality for other frames with more heterogeneity.

Figure 11 compares the (normalized) data usage, which is defined as the total downloaded bytes including the SR models and meta data. Compared to 4×1, applying C&R reduces the data usage by 40.5%. Also, increasing the SR ratio reduces the data usage, *e.g.,* 1×4D consumes only 18.3% of the data compared to 4×1. The full-fledged YuZu with adaptation gives the overall best QoE (Figure 10) and low data usage (Figure 11) by balancing the compute and network resource consumption. Compared to 4×1, full YuZu reduces the data usage by 52.3% (50Mbps) and 41.9% (75Mbps) while boosting the QoE by 214% (50Mbps) and 78.3% (75Mbps).

Fluctuating Bandwidth. We repeat the above experiment over fluctuating bandwidth emulated using our collected LTE traces (§7.1). The results are shown in Figure 12, which considers both the data usage (x-axis) and the QoE (y-axis). 4×1 yields the highest data usage; further applying C&R (4×1D) not only reduces the data usage by 40.5%, but also increases the QoE by 61.8% due to reduced stall. The full YuZu further improves the QoE by 21.0% and reduces the average data usage by 8.2%. This is achieved through strategically fetching lower-quality blocks and using higher SR ratios. In addition, the full YuZu improves the QoE by 10.4% to 93.7%, compared to 1×4 and 2×2 with and without C&R.

Live LTE. We conduct live LTE experiments at 9 locations in a major U.S. city. As shown in Figure 13, the results are largely aligned with those in Figure 12, except for the lower QoE of 4×1. This is because of the lower bandwidth of live LTE throughout the test locations compared to the LTE traces used in Figure 12. Compared to 4×1, the full YuZu improves the QoE by 210.3% and reduces the data usage by 50.8%.

7.5 YuZu vs. Existing Approaches

YuZu vs. Viewport-Adaptive Streaming. We compare YuZu with ViVo [27], a recently proposed viewport-adaptive approach. Leveraging 6DoF motion prediction, ViVo determines what content to fetch and which quality to fetch based on

predicted viewport and viewing distance. Similar viewport-adaptive approaches are used in the other systems [41, 50].

We develop a custom replication of ViVo on Linux in 7,101 LoC with the same set of configuration parameters. Figure 14 shows the improvement brought by YuZu compared to ViVo in terms of the overall QoE and its three components (Q_i, I_i^{patch}, and I_i^{frame}, see Eq. 8), using all four videos and the users' motion traces.[4] Note that both systems exhibit negligible stall so I_i^{stall} is not plotted. As shown, YuZu brings significant improvement on the average QoE (by 100.6% to 174.9%) and on each QoE component. YuZu outperforms ViVo due to three reasons. First, ViVo does not support SR, which YuZu leverages to boost the QoE. Second, ViVo's viewport adaptation approach becomes less effective when the whole scene appears inside the viewport (which oftentimes appears in our motion traces). SR does not suffer from this limitation. Third, to realize viewport adaptive streaming, ViVo has to perform 6DoF motion prediction, which is error-prone. In contrast, YuZu does not require motion prediction, and therefore exhibits more stable performance in particular when the motion is fast. Note that viewport-adaptation and SR are orthogonal approaches and can be *jointly* applied.

YuZu vs. Simple SR Adaptation. To demonstrate the efficacy of our network/compute resource adaptation design (§5.3), we compare it with a simple adaptation approach that differs in two aspects. First, unlike YuZu's *two-stage* adaptation, it only performs *single-stage* adaptation before downloading each chunk. Second, it employs a *deterministic* greedy algorithm that increases the SR ratio of each block within the finite horizon (in chronological order) until the QoE does not further improve. In contrast, YuZu employs a *probabilistic* greedy approach that is less vulnerable to a local maximum. We evaluate the simple adaptation algorithm using our LTE traces (§7.4) and plot its result as "Simple" in Figure 12. Compared to it, the full YuZu increases the average QoE by 11.4% and reduces the average data usage by 7.9%.

7.6 Micro Benchmarks and Resource Usage

We conduct experiments to show the following. (1) YuZu can work adaptively with different hardware (we compare the results on 2080Ti and 1660Ti; we also ported YuZu to an embedded system, see Figure 7). (2) The main memory (~5GB) and GPU memory (~2GB) usage of YuZu is acceptable. (3) The (one-time) offline training time is non-trivial but acceptable, and the sizes of SR models are negligible (<0.2% of the video size). The details can be found in Appendix D.

8 Related Work

Volumetric Video Streaming. There exist only a few studies on point-cloud-based volumetric video streaming [25–27, 31, 41, 50, 52, 59]. For example, DASH-PC [31] extends DASH to volumetric videos. PCC-DASH [59] is another DASH-based streaming scheme of compressed point clouds with bitrate adaptation support. ViVo [27] introduces visibility-aware optimizations for volumetric video streaming. GROOT [41] optimizes point cloud compression for volumetric videos. To the best of our knowledge, there is no existing work on applying 3D SR to volumetric video streaming.

Point Cloud SR. We can classify existing work on point cloud SR into two categories: optimization-based [16, 32] and learning-based [43, 61, 63, 70]. Most learning-based approaches follow the workflow established in PU-Net [70], which divides a point cloud into patches, learns multi-level point features of each patch, expands the features, and reconstructs the points from the features. All the above methods are designed for a *single* point cloud; they suffer from numerous limitations when applied to volumetric videos (§2).

Visual Quality Assessment of Point Clouds. The state-of-the-art visual quality assessment focuses on *static, non-SR* point clouds [23, 47, 60]. For example, using a data-driven approach, Meynet *et al.* [47] present a full-reference visual quality metric for colored point clouds. Different from the above studies, we model the QoE of SR-enhanced volumetric video streaming. We address new challenges on modeling the impact of various factors such as the viewing distance, upsampling ratio, and SR incurred distortion (§4).

SR for Regular 2D Videos. NAS [67, 68] is one of the first proposals that apply 2D SR to Internet video streaming. Other recent efforts on 2D SR include PARSEC [22] for 360° panoramic video streaming, LiveNAS [39] for live video streaming, and NEMO [66] for mobile video streaming. In contrast, YuZu addresses numerous unique challenges (§1) on applying 3D SR to volumetric video streaming.

9 Concluding Remarks

In this paper, we conduct an in-depth investigation on applying 3D SR to streaming volumetric content. Our proposed QoE model and the YuZu system take a first and important step toward making SR-enhanced volumetric video streaming principled, practical, and affordable. YuZu demonstrates how a series of novel optimizations, which fill a 500× performance gap, as well as judicious network/compute resource adaptation can help significantly improve the QoE for volumetric video streaming.

Acknowledgments

We thank the anonymous reviewers and our shepherd Anirudh Badam for their insightful comments. The research of Feng Qian was supported in part by a Cisco research award. The research of Bo Han was funded in part by 4-VA, a collaborative partnership for advancing the Commonwealth of Virginia.

[4] ViVo does not have the notion of patch; instead its basic adaptation unit is a cubic cell. To ensure fair comparisons, we further divide ViVo's cells into virtual "patches" with the same size as YuZu and assign to them its parent cell's corresponding quality level when calculating I_i^{patch}.

References

[1] 8i Voxelized Full Bodies (8iVFB v2) - Dynamic Voxelized Point Cloud Dataset. http://plenodb.jpeg.org/pc/8ilabs.

[2] Amazon Mechanical Turk. https://www.mturk.com/.

[3] Bazel. https://bazel.build/.

[4] Draco 3D Data Compression. https://google.github.io/draco/.

[5] ITU-P.913: Methods for the subjective assessment of video quality, audio quality and audiovisual quality of Internet video and distribution quality television in any environment. https://www.itu.int/rec/T-REC-P.913.

[6] Linux TC Man Page. https://linux.die.net/man/8/tc.

[7] Magic Leap One. https://www.magicleap.com/en-us/magic-leap-1.

[8] MPU. https://github.com/yifita/3PU.

[9] Protocol Buffers. https://developers.google.com/protocol-buffers.

[10] PU-GAN. https://github.com/liruihui/PU-GAN.

[11] Qualtrics experience management platform. https://www.qualtrics.com/.

[12] SPH3D-GCN. https://github.com/hlei-ziyan/SPH3D-GCN.

[13] TensorFlow 1.14. https://github.com/tensorflow/tensorflow/tree/r1.14.

[14] The Octree Data Structure. https://en.wikipedia.org/wiki/Octree.

[15] Video Multimethod Assessment Fusion. https://en.wikipedia.org/wiki/Video_Multimethod_Assessment_Fusion, 2016.

[16] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and Rendering Point Set Surfaces. *IEEE Trans. on Visualization and Computer Graphics*, 9(1):3–15, 2003.

[17] Armen S Asratian, Tristan MJ Denley, and Roland Häggkvist. *Bipartite graphs and their applications*, volume 131. Cambridge university press, 1998.

[18] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a Predictive Model of Quality of Experience for Internet Video. In *Proceedings of ACM SIGCOMM*, 2013.

[19] Egon Balas and Paolo Toth. Branch and bound methods for the traveling salesman problem. 1983.

[20] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[21] Hong Chang, Dit-Yan Yeung, and Yimin Xiong. Super-Resolution through Neighbor Embedding. In *Proceedings of CVPR*, 2004.

[22] Mallesham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjul Sahu, Aruna Balasubramanian, and Samir Das. Streaming 360 degree Videos using Super-resolution. In *Proceedings of IEEE INFOCOM*, 2020.

[23] Rafael Diniz, Pedro Garcia Freitas, and Mylène C.Q. Farias. Towards a Point Cloud Quality Assessment Model Using Local Binary Patterns. In *Proceedings of the 12th International Conference on Quality of Multimedia Experience (QoMEX)*, 2020.

[24] Steven Fortune. Voronoi diagrams and delaunay triangulations. In *Comp. in Euclidean Geometry*, pages 225–265. World Sci., 1995.

[25] Serhan Gül, Dimitri Podborski, Thomas Buchholz, Thomas Schierl, and Cornelius Hellge. Low-latency cloud-based volumetric video streaming using head motion prediction. In *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 27–33, 2020.

[26] Serhan Gül, Dimitri Podborski, Jangwoo Son, Gurdeep Singh Bhullar, Thomas Buchholz, Thomas Schierl, and Cornelius Hellge. Cloud Rendering-based Volumetric Video Streaming System for Mixed Reality Services. In *Proceedings of ACM MMSys*, 2020.

[27] Bo Han, Yu Liu, and Feng Qian. ViVo: Visibility-Aware Mobile Volumetric Video Streaming. In *Proceedings of ACM MobiCom*, 2020.

[28] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. Rubiks: Practical 360° Streaming for Smartphones. In *Proceedings of ACM MobiSys*, 2018.

[29] Qi He, Constantine Dovrolis, and Mostafa Ammar. On the predictability of large transfer tcp throughput. *ACM SIGCOMM Computer Communication Review*, 35(4):145–156, 2005.

[30] Alain Hore and Djemel Ziou. Image quality metrics: PSNR vs. SSIM. In *Proceedings of the 20th International Conference on Pattern Recognition*, pages 2366–2369. IEEE, 2010.

[31] Mohammad Hosseini and Christian Timmerer. Dynamic Adaptive Point Cloud Streaming. In *Proceedings of ACM Packet Video*, 2018.

[32] Hui Huang, Shihao Wu, Minglun Gong, Daniel Cohen-Or, Uri Ascher, and Hao Zhang. Edge-Aware Point Set Resampling. *ACM Transactions on Graphics*, 32(1):9:1–9:12, 2013.

[33] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of ACM SIGCOMM*, 2014.

[34] Yan Huang, Jingliang Peng, C.-C. Jay Kuo, and M. Gopi. A Generic Scheme for Progressive Point Cloud Coding. *IEEE Trans. on Vis. and Computer Graphics*, 14(2):440–453, 2008.

[35] Erik Hubo, Tom Mertens, Tom Haber, and Philippe Bekaert. The Quantized kd-Tree: Efficient Ray Tracing of Compressed Point Clouds. In *Proceedings of IEEE Symposium on Interactive Ray Tracing*, 2006.

[36] Hanbyul Joo, Tomas Simon, Xulong Li, Hao Liu, Lei Tan, Lin Gui, Sean Banerjee, Timothy Scott Godisart, Bart Nabbe, Iain Matthews, Takeo Kanade, Shohei Nobuhara, and Yaser Sheikh. Panoptic studio: A massively multiview system for social interaction capture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[37] Julius Kammerl, Nico Blodow, Radu Bogdan Rusu, Suat Gedikli, Michael Beetz, and Eckehard Steinbach. Real-time Compression of Point Cloud Streams. In *Proceedings of International Conference on Robotics and Automation*, 2012.

[38] Ali Safari Khatouni, Marco Mellia, Marco Ajmone Marsan, Stefan Alfredsson, Jonas Karlsson, Anna Brunstrom, Ozgu Alay, Andra Lutu, Cise Midoglu, and Vincenzo Mancuso. Speedtest-like measurements in 3g/4g networks: The monroe experience. In *Proceedings of the 29th International Teletraffic Congress (ITC 29)*, pages 169–177, 2017.

[39] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. Neural-Enhanced Live Streaming: Improving Live Video Ingest via Online Learning. In *Proceedings of ACM SIGCOMM*, 2020.

[40] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[41] Kyungjin Lee, Juheon Yi, Youngki Lee, Sunghyun Choi, and Young Min Kim. GROOT: A Real-Time Streaming System of High-Fidelity Volumetric Videos. In *Proceedings of ACM MobiCom*, 2020.

[42] Huan Lei, Naveed Akhtar, and Ajmal Mian. Spherical Kernel for Efficient Graph Convolution on 3D Point Clouds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[43] Ruihui Li, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. PU-GAN: A Point Cloud Upsampling Adversarial Network. In *Proceedings of ICCV*, 2019.

[44] Jyh-Ming Lien, Gregorij Kurillo, and Ruzena Bajcsy. Multi-camera tele-immersion system with real-time model driven data compression. *The Visual Computer*, 26(3):3–15, 2010.

[45] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of ACM SIGCOMM*, 2017.

[46] Rufael Mekuria, Kees Blom, and Pablo Cesar. Design, Implementation and Evaluation of a Point Cloud Codec for Tele-Immersive Video. *IEEE Trans. on Circuits and Systems for Video Technology*, 27(4):828–842, 2017.

[47] Gabriel Meynet, Yana Nehmé, Julie Digne, and Guillaume Lavoué. PCQM: A Full-Reference Quality Metric for Colored 3D Point Clouds. In *Proceedings of the 12th International Conference on Quality of Multimedia Experience (QoMEX)*, 2020.

[48] Carsten Moenning and Neil A Dodgson. Fast marching farthest point sampling. Technical report, University of Cambridge, 2003.

[49] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip Davidson, Sameh Khamis, Mingsong Dou, Vladimir Tankovich, Charles Loop, Qin Cai, Philip Chou, Sarah Mennicken, Julien Valentin, Vivek Pradeep, Shenlong Wang, Sing Bing Kang, Pushmeet Kohli, Yuliya Lutchyn, Cem Keskin, and Shahram Izadi. Holoportation: Virtual 3D Teleportation in Real-time. In *Proceedings of ACM UIST*, 2016.

[50] Jounsup Park, Philip A Chou, and Jenq-Neng Hwang. Volumetric media streaming for augmented reality. In *2018 IEEE Global communications conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[51] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Proceedings of Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[52] Feng Qian, Bo Han, Jarrell Pair, and Vijay Gopalakrishnan. Toward Practical Volumetric Video Streaming On Commodity Smartphones. In *Proceedings of ACM HotMobile*, 2019.

[53] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *Proceedings of ACM MobiCom*, 2018.

[54] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[55] Paul Scovanner, Saad Ali, and Mubarak Shah. A 3-dimensional SIFT descriptor and its application to action recognition. In *Proceedings of the 15th ACM International Conference on Multimedia*, pages 357–360, 2007.

[56] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining Point Cloud Local Structures by Kernel Correlation and Graph Pooling. In *Proceedings of CVPR*, 2018.

[57] Steven L Tanimoto, Alon Itai, and Michael Rodeh. Some matching problems for bipartite graphs. *Journal of the ACM (JACM)*, 25(4):517–525, 1978.

[58] Nikolaos Thomos, Nikolaos V Boulgouris, and Michael G Strintzis. Optimized transmission of jpeg2000 streams over wireless channels. *IEEE Transactions on image processing*, 15(1):54–67, 2005.

[59] Jeroen van der Hooft, Tim Wauters, Filip De Turck, Christian Timmerer, and Hermann Hellwagner. Towards 6DoF HTTP Adaptive Streaming Through Point Cloud Compression. In *Proceedings of ACM Multimedia*, 2019.

[60] Irene Viola, Shishir Subramanyam, and Pablo Cesar. A color-based objective quality metric for point cloud contents. In *Proceedings of the 12th International Conference on Quality of Multimedia Experience (QoMEX)*, 2020.

[61] Yifan Wang, Shihao Wu, Hui Huang, Daniel Cohen-Or, and Olga Sorkine-Hornung. Patch-based Progressive 3D Point Set Upsampling. In *Proceedings of CVPR*, 2019.

[62] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[63] Huikai Wu, Junge Zhang, and Kaiqi Huang. Point cloud super resolution with adversarial residual graph networks. In *arXiv preprint arXiv:1908.02111*, 2019.

[64] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *Proceedings of USENIX NSDI*, 2020.

[65] Jianchao Yang, John Wright, Thomas S. Huang, and Yi Ma. Image Super-Resolution Via Sparse Representation. *IEEE Transactions on Image Processing*, 19(11):2861–2873, 2010.

[66] Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. NEMO: Enabling Neural-enhanced Video Streaming on Commodity Mobile Devices. In *Proceedings of ACM MobiCom*, 2020.

[67] Hyunho Yeo, Sunghyun Do, and Dongsu Han. How will Deep Learning Change Internet Video Delivery? In *Proceedings of ACM HotNets*, 2017.

[68] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. Neural Adaptive Content-aware Internet Video Delivery. In *Proceedings of USENIX OSDI*, 2018.

[69] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of ACM SIGCOMM*, 2015.

[70] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. PU-Net: Point Cloud Upsampling Network. In *Proceedings of CVPR*, 2018.

[71] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. Efficient volumetric video streaming through super resolution. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, pages 106–111, 2021.

Appendices

A Evaluation of QoE Gain Brought by SR

We study the QoE model for $q_{i,j}$ (Eq. 2) while keeping I_i^{patch}, I_i^{frame}, and I_i^{stall} as zero. This allows us to measure the impact of SR without interference from other factors.

We use the four videos introduced in §4.2 for the experiment. We apply our optimized PU-GAN algorithm (details

in §5.1) to perform upsampling, and create $\binom{8}{2} = 28$ video clips where each clip contains 2 out of 8 versions in Table 3 side by side (in a random order). This approach is known as the double stimulus comparison scale (DSCS) method [5] as recommended by ITU (International Telecommunication Union). We repeat the above process for four viewing distances: 1m, 2m, 3m, and 4m, which are determined from a separate IRB-approved user study whose details are described in Appendix B. To maintain a fixed viewing distance d, we display the viewport at d meters in front of and facing the viewer. We generate 112 video clips at 4K resolution for each video segment.

Next, we design a survey using Qualtrics [11] and publish it on Amazon Mechanical Turk (AMT) [2]. In the survey, we invite each paid AMT subject to view the 112 clips of a random video segment (out of the 4 videos) in a random order. After watching each clip, the subject is asked to rate which side provides a better QoE through 7 choices ("left looks {much better, better, slightly better, similar to, slightly worse, worse, much worse} than right"). We have collected 512 subjects' responses with a total number of 57,344 ratings. We show the demographics of the participants in Table 2.

Figure 15 shows the average ratings of the 8 versions across all the users. The four subplots correspond to the four viewing distances. We make four observations. First, when the viewing distance is short, SR can effectively boost the QoE. For example, at 1m, compared to 1×1, the (user-rated) QoE increases by 37%, 75%, 150% for 1×2, 1×3, and 1×4, respectively; 2×2 improves the QoE by 178% compared to 2×1. Second, under the same point density, the upsampled version's QoE is usually lower than the original content's QoE, in particular when the SR ratio is large. This is caused by SR's distortion. However, the gap tends to reduce as the SR ratio decreases. Third, SR's gain diminishes as the distance increases, because the rendered object becomes smaller in the view. Note that the scores for different distances are not directly comparable. Fourth, the four video segments exhibit similar trends (figure not shown).

Converting User Ratings to Numerical Scores. For a given tuple of (user, viewing distance, video segment), we construct a weighted directed graph for the user based on his/her ratings, where the nodes are the 8 schemes. Assume a video clip contains schemes A (on the left) and B (on the right). If the user thinks that the left (right) is much better, better, or slightly better than the right (left), we add an edge from B to A (A to B) with a weight of 3, 2, and 1, respectively. If the user thinks that the left is similar to the right, we add two edges between A and B, one from A to B and the other from B to A, with both edges' weights set to 0. We then normalize the weights of all the edges to $[0,1]$ and apply the PageRank algorithm [20] to each graph to compute the weight of every node. We then use the weights (multiplied by 10 for easy interpretation) as the numerical scores of the 8 schemes for the corresponding (user, viewing distance, video segment) tuple. Finally, for each of the 8 schemes under a given viewing distance, we average the numerical scores across all the tuples (of that viewing distance) to obtain the results shown in Figure 15. Note that for each viewing distance, the weights of all the schemes (in each of the graphs) add up to 1. As a result, the numerical scores of the same scheme for different viewing distances are not directly comparable.

B User Study for Collecting 6DoF Motion Traces

We conducted a separate IRB-approved user study for collecting 6DoF motion traces of volumetric videos. Specifically, it captured the viewport trajectories of 32 users who watched the four video segments (*Lab*, *Dress*, *Loot*, *Haggle*) introduced in §2 and §4.2 through either a mixed reality headset (Magic Leap One [7]) or an Android smartphone. We developed custom volumetric video players for both device types. The 6DoF motion data (yaw, pitch, roll, X, Y, Z) was captured at the granularity of 30 Hz. The participants are diverse in terms of their education level (from freshman to Ph.D.), gender (16 females), and age (from 22 to 57). We determine the viewing distances used in §4.2 by analyzing the above traces. As shown in Figure 16, about 70% of the viewing distances are less than 4m. Therefore, we set the maximum viewing distance to be 4m for our user studies, and select the other three distances by evenly dividing this maximum distance into four ranges (*i.e.,* at 1, 2, and 3m).

C Colorization Algorithm of YuZu and its Evaluation

Recall from §5.4 that YuZu takes a lightweight approach to color the SR results: it approximates each upsampled point's color using the color of the nearest point in the low-density point cloud (*i.e.,* the input to the SR model).

YuZu employs two mechanisms to speed up the nearest point search. First, the search is performed on an octree [14], which recursively divides a point cloud (as the root node) into eight octants, each associated with a child node. The levels of detail of the point cloud are controlled by the height of the tree. Performing nearest point search on an octree has a low complexity of $O(logN)$ where N is the number of nodes in the tree.

Second, YuZu caches and reuses the results of previously searched points. The cache is indexed by a point's discretized coordinates, and the cached value is the color looked up from the octree. When coloring an upsampled point, YuZu first performs cache lookup in $O(1)$; upon a hit, the cached color will be directly used as the color of the point; otherwise, YuZu performs a full octree search and adds the search result to the cache. The discretization granularity incurs a tradeoff between colorization performance and quality. We empirically observe that a discretization granularity of $1cm^3$ can yield good visual

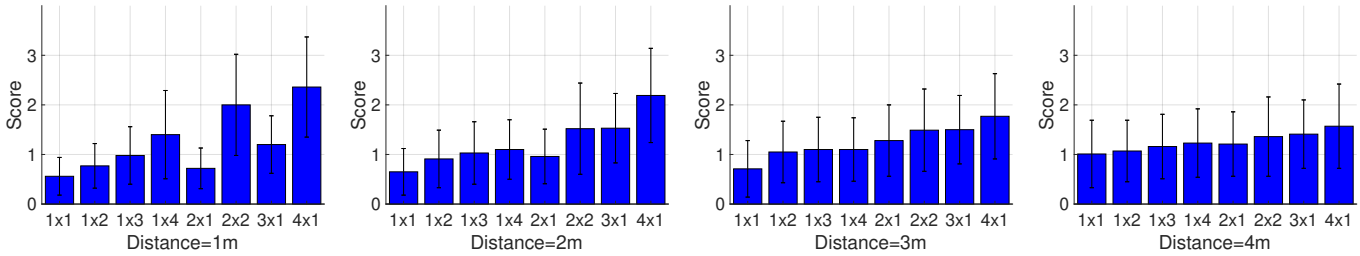

Figure 15: The average ratings of the 8 versions across all the users watching all the four video segments (*Long Dress*, *Loot*, *Band*, and *Haggle*).

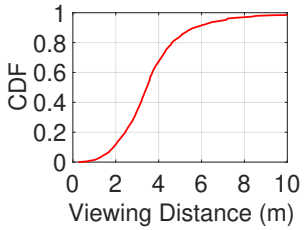

Figure 16: Distribution of viewing distance in our motion traces.

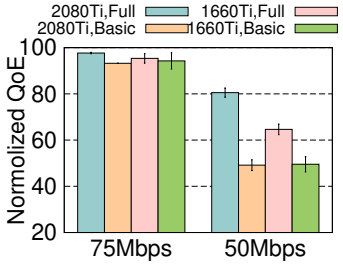

Figure 17: Impact of hardware and computation-aware adaptation.

quality under typical viewing distances (≥ 1m).

We also notice opportunities for further improving the colorization quality. For example, the nearest point approach can be generalized into interpolating the nearest k points' colors; it can also be used in conjunction with DNN-based colorization, which may be more suitable for patches with complex, heterogeneous colors. Nevertheless, these enhancements inevitably increase the runtime overhead. We will explore them in future work.

Evaluation of Quality of Colorization. To evaluate the quality of the colorization step alone, we employ the approach in §7.2 where we use PSNR to objectively assess the image quality of rendered viewports. Specifically, we calculate the PSNR values by comparing $\{I_{4\times1}^{NP-Color}\}$ (defined below) with $\{I_{4\times1}\}$ (defined in §7.2), using the *Dress* and *Loot* videos and the real users' motion traces (Appendix B). The viewport images of $\{I_{4\times1}^{NP-Color}\}$ are obtained as follows: (1) remove the color from the original (4×1) video; (2) apply the above nearest-point (NP) colorization method to the video generated in Step (1), using the 1×1 video as the low-resolution point cloud stream from which the colors are picked; (3) replay the same motion traces to render the viewport images for the video colored in Step (2). The PSNR values of $\{I_{4\times1}^{NP-Color}\}$ are 38.09±2.44 and 44.15±2.59 for *Dress* and *Loot*, respectively, indicating the high fidelity of colors produced by our method. The above numbers are much higher than the PSNR values in Figure 8 (which also includes the colorization step) due to the following reason. PSNR and many other 2D image metrics such as SSIM [62] perform a pixel-wise comparison between two images. In the case of Figure 8, a tiny position shift of a 3D point may result in an also tiny position shift of its projected 2D pixel, leading to a pixel mismatch and thus a decreased PSNR score. This problem does not appear in the colorization step.

D Additional Micro Benchmarks

The following micro benchmark results are generated using the PU-GAN model. The results for the MPU model are qualitatively similar.

Impact of Computation-aware Adaptation. 3D SR demands considerable compute resources. Figure 17 demonstrates the impact of hardware and computation-aware adaptation, using the *Lab* video. Figure 17 considers two GPUs: a more powerful 2080Ti GPU and a less powerful 1060Ti GPU. It also considers two adaptation schemes: the full network/compute adaptation scheme described in §5.3 ("Full") and a computation-agnostic scheme that only adapts according to the network bandwidth ("Basic"). The Basic scheme works as follows. (1) It assumes that SR takes no time to complete; (2) it disables 2×2 and 1×4 (otherwise the QoE will degrade too much due to excessive stalls). Under the above setup, each bandwidth setting in Figure 17 has four schemes: {2080Ti, 1660Ti} × {Full, Basic}. As shown, when there is sufficient bandwidth, the QoE differences among the four schemes are small, because the player is more likely to fetch 3×1 and 4×1 blocks that do not require SR. However, when the bandwidth becomes low, the difference between 2080Ti and 1060Ti becomes noticeable, and the gap between Full and Basic is even larger. The Basic scheme yields much lower QoE scores because it ignores SR's computation overhead, leading to excessive stalls.

Memory Usage. We measure the client-side memory usage when streaming the *Lab* video over 50Mbps bandwidth (which leads to extensive invocations of SR). On the 2080Ti

(1660Ti) desktop, the peak main memory usage is 5.03GB (5.33 GB); the peak GPU memory usage is 1.97 GB (1.83 GB). YuZu's GPU memory usage on 2080Ti is higher than the numbers reported in Figure 6 because YuZu loads multiple SR models at runtime. When the available bandwidth is higher, the CPU/GPU memory will reduce because of fewer SR operations.

Offline Training Time and Model Size. YuZu incurs non-trivial model training time. For example, on the 2080Ti desktop, it takes about 88 minutes to train the 1×2, 1×3, and 1×4 models altogether for the *Lab* video consisting of 3,622 frames. However, note that (1) this is a one-time overhead; (2) we did not conduct any performance optimization for training; for a large-scale deployment, the training overhead could potentially be reduced by training one generic model and fine-tuning it for each specific video [68] (left as future work). The SR model size is negligible ($< 0.2\%$) compared to the video size.