Toward Practical Volumetric Video Streaming On Commodity Smartphones

Feng Qian[1] Bo Han[2] Jarrell Pair[3] Vijay Gopalakrishnan[2]
[1]University of Minnesota – Twin Cities [2]AT&T Labs – Research [3]AT&T

ABSTRACT

Volumetric videos offer six degree-of-freedom (DoF) as well as 3D rendering, making them highly immersive, interactive, and expressive. In this paper, we design Nebula, a practical and resource-efficient volumetric video streaming system for commodity mobile devices. Our design leverages edge computing to reduce the computation burden on mobile clients. We also introduce various optimizations to lower the perceived "motion-to-photon" delay, to dynamically adapt to the fluctuating network bandwidth, and to reduce the system's resource consumption while maintaining a high QoE.

ACM Reference Format:
Feng Qian, Bo Han, Jarrell Pair, and Vijay Gopalakrishnan. 2019. Toward Practical Volumetric Video Streaming On Commodity Smartphones. In *The 20th International Workshop on Mobile Computing Systems and Applications (HotMobile '19), February 27–28, 2019, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3301293.3302358

1 INTRODUCTION

This paper examines *volumetric videos*[1], an emerging immersive media, and explores how to efficiently stream them to mobile devices. Compared to regular videos and 360° panoramic videos that recently became popular, volumetric videos are unique in two key aspects. First, regular and 360° video content is on either a plane or a sphere. Volumetric videos are instead truly *three-dimensional*: they consist of not 2D pixels, but elements such as voxels (volume pixels) or 3D meshes (polygons). Second, volumetric videos provide six degrees of freedom (6DoF), allowing a viewer to freely change both the position (X, Y, Z) and the orientation (yaw, pitch, roll) of her viewport. By contrast, regular videos provide no viewport freedom, and 360° videos allow only 3DoF as the viewer's translational position is always fixed.

Both features above make volumetric videos highly immersive, interactive, and expressive. They can support numerous innovative applications from entertainment to medical and education. Volumetric videos can be represented in different ways. In this paper, we exemplify our approaches using the *Point Cloud* (PtCl) representation where each video frame consists of multiple voxels or points [13]. Nevertheless our high-level approaches are applicable to other sophisticated representations such as 3D meshes [17]. PtCl is a popular way to represent 3D objects due to its simplistic data structure and good rendering performance. Static PtCls have been well studied in the computer graphics and multimedia communities (§5). However, existing work has not studied the delivery of volumetric videos that consist of a *stream* of PtCls to resource-constrained mobile devices and over bandwidth-limited wireless links, which is the focus of this paper.

Streaming volumetric videos is challenging due to several reasons. First, they are extremely bandwidth-consuming (§3), and thus their wireless delivery may require the support from future 5G networks [1]. Second, unlike regular pixel videos that can be decoded using dedicated hardware, decoding volumetric videos can only be done by software today. This results in high computational overhead. Third, adaptive-bitrate (ABR) video streaming systems typically have several key components such as rate adaptation, QoE inference, and buffer control. Little research has been done on any of these aspects for volumetric video streaming on mobile devices.

In this paper, we begin with developing a proof-of-concept PtCl player on Android platform (§3), and use it to conduct measurements to demonstrate several key challenges described above. Based on our observations, we present a holistic design of a practical PtCl video streaming system for commodity smartphones. We call our system Nebula, whose key design aspects consist of the following.

• **Layered Content Organization** (§4.1). We devise a DASH-style scheme to organize the content on the server side. By leveraging the unique characteristics of PtCl data, we delta-encode each video chunk into *layers* to allow its quality to be incrementally upgraded.

• **Edge Assistance** (§4.2). Since directly decoding a compressed PtCl video on a phone is expensive, we introduce an edge (proxy) server in Nebula. The proxy *transcodes* the PtCl stream into a regular pixel-based video stream, which captures the viewer's viewport and can be efficiently decoded on smartphones. We propose various optimizations to reduce the motion-to-photon delay [2] between the client and the proxy.

• **Rate Adaptation** (§4.3). Nebula integrates two separate rate adaptation mechanisms, with one running between the phone and proxy, and the other between the proxy and the server, to adapt the video quality to the varying network condition. In particular, we propose directions toward developing a robust rate adaptation algorithm for volumetric videos.

• **Viewport Adaptation** (§4.4). For a spatially large PtCl or a scene with multiple scattered PtCls, fetching or decoding the entire PtCl may be infeasible. In this case, Nebula applies *viewport adaptation* that allows the proxy to fetch only the portions that the viewer is seeing or about to see, based on predicting the 6DoF movement of the viewport. We further propose optimizations that reduce the

[1]A demonstration of high-quality volumetric video streaming can be found at https://www.youtube.com/watch?v=feGGKasvamg.

HotMobile '19, February 27–28, 2019, Santa Cruz, CA, USA

ACM ISBN 978-1-4503-6273-3/19/02…$15.00
https://doi.org/10.1145/3301293.3302358

resource utilization while maintaining a high QoE for viewport adaptation.

2 VOLUMETRIC VIDEOS: A PRIMER

Capturing. Volumetric videos are captured using multiple RGB-D cameras with *depth* sensors, *e.g.,* offered by Microsoft Kinect, Intel RealSense, and various LIDAR scanners [23], which acquire 3D data from different viewpoints. The acquired data is then merged to obtain the entire scene. During the merging process, frames captured by different cameras should be properly synchronized, their coordinates be unified, and noises be removed. The whole processing pipeline can be realized by open-source software such as LiveScan3D [14], a 3D data acquisition system supporting multiple Kinect devices.

Representation. Point cloud (PtCl) and 3D mesh are two popular ways to represent volumetric videos. 3D mesh has been extensively investigated by the computer graphics community for a long time [17]. It employs a collection of meshes, which can be triangles, quadrangles, or general polygons, to represent the geometry of 3D models. Although 3D mesh can accurately model objects, its algorithms are complex as they need to maintain the topological consistency in the processing pipeline [13]. In contrast, PtCl is a much more flexible and simpler representation as introduced in §1.

Compression (*i.e.,* encoding) of 3D objects has been well studied in the literature, with the state-of-the-art compression scheme being *octree*-based approaches [7, 11, 18]. An octree is a tree data structure that partitions a 3D region by recursively dividing it into eight subregions [24], with the levels of detail being controlled by the height of the tree. When applied to a PtCl, an octree efficiently stores the points in such a way that its nodes correspond to regions that contain at least one point, and the best estimation of the points' locations is given by the leaf nodes. In addition to compressing a single PtCl (or a single PtCl video frame), the octree can be extended to perform delta encoding between two PtCls (or inter-frame compression for a PtCl video) [12]. Note that as of today, decoding volumetric videos is typically performed by software. But as volumetric video becomes more popular, dedicated hardware support for volumetric video decoding may appear in the future.

3 PTCL STREAMING ON SMARTPHONES

We examine the performance of volumetric video streaming on mobile devices, to motivate the design of Nebula.

PtCl Video Player. Despite existing efforts on volumetric video streaming [10, 21], fewer studies, if any, have been conducted on understanding its performance on commodity smartphones. We thus first develop a PtCl video player for Android. Our player was written in Java and C++ in 2,800 LoC. It is capable of playing PtCl videos hosted locally or remotely (fetched over TCP). The videos can be in either raw or compressed format. A raw video contains a series of *frames* to be played at a fixed rate (*e.g.,* 24 or 30 FPS). Each frame consists of a PtCl *i.e.,* a list of 3D points. Each point occupies 9 bytes: its position (X, Y, Z, 2 bytes each) and its color (R, G, B, 1 byte each). A compressed video has a similar format except that each frame (PtCl) is encoded using an octree (we have not yet implemented inter-frame compression). The player maintains a playback buffer of up to 50 frames, and plays each frame by rendering its PtCl using OpenGL/GPU.

During a playback, the viewer can freely change the viewport's position (X, Y, Z) and orientation (yaw, pitch, roll). For intuitive interactions, the viewer can select one of three modes and swipe the screen to change the viewport: *Orbit* (rotating the viewport around the Y axis), *Zoom* (moving the viewport along the Z axis, closer or further to the origin), and *Pan* (moving the viewport along the X or Y axis).

Dataset. We use a volumetric video captured at AT&T Shape [1]. Depicting a female singer, it allows viewers to immersively watch her performance. The video consists of 3,188 frames, and each frame contains on average 50,360 points. Using this video, we further construct four other videos with sparser or denser points: 12.6K, 25.2K, 75.5K, and 100.7K on average in a frame. We make the points sparser by performing random sampling, and make them denser by adding noises (randomly generated points) to each frame.

Measurements. We next use our player to measure the streaming performance. All experiments were conducted on Samsung Galaxy S8, a state-of-the-art smartphone as in 2018. Unless otherwise stated, all reported measurement results are averaged over 5 playbacks of our test video described above.

Observation 1. Rendering an uncompressed (decoded) PtCl video is fast. We begin with playing uncompressed PtCl videos locally. As shown in the "Avg Render FPS" column in Table 1, depending on the frame density, the average FPS ranges from 173 to 1110, considerably higher than the required frame rate of 24 FPS. This is attributed to the simplicity of the PtCl structure that allows fast rendering, compared to other complex representations such as 3D mesh [11, 13].

Observation 2. Transferring uncompressed PtCl video is challenging over today's wireless networks. We next stream the uncompressed videos from a server to a client over commercial LTE networks. Despite the high bandwidth (up to 40Mbps) offered by LTE, the playback experienced unacceptably long stalls for almost all videos. The reason is simply the large frame size as shown in Table 1. For a PtCl video with 50K points per frame, the required bandwidth for streaming uncompressed frames at 24 FPS is $9 \times 50\text{K} \times 24 \times 8 = 86.4$Mbps.

Observation 3. Decoding performance on today's mobile devices is poor. Given the above observation, a natural idea is to encode (compress) a PtCl video before its transmission. We thus cross-compile the Point Cloud Library (PCL [3]), a production-quality library for PtCl processing, on Android. We integrate PCL into our player by using PCL's built-in functions for efficient octree-based PtCl encoding and decoding. Surprisingly, as shown in Table 1, the decoding performance on SGS8 is very poor, with the FPS ranging from 1.5 to 13.9, due to the costly operations of walking through the octree, inspecting each node, and reconstructing the decoded data to be consumed by the OpenGL shader.

Octree-based encoding is typically lossy. Therefore PCL supports different resolution profiles that control the video quality by adjusting the height of the octree. The results in Table 1 are for the low-resolution profile (favoring faster decoding, higher compression ratio). Table 2 shows the phone-side decoding performance for medium- and high-resolution profiles, whose FPS and compression ratio further degrade.

Avg # Points	Frame Size	Avg Render FPS (Phone)	Dec. FPS (Phone)	Dec. FPS (Server)
12.6K	0.11MB	1110.8	13.9	41.7
25.2K	0.23MB	776.0	7.2	20.2
50.4K	0.45MB	351.6	3.5	10.1
75.5K	0.68MB	233.3	2.1	6.1
100.7K	0.91MB	173.3	1.5	4.4

Table 1: PtCl stream rendering performance on SGS8 and decoding performance on SGS8 and an edge server (1 thread, low-resolution profile).

Res. Profile	Dec. FPS (Phone)	Comp. Ratio
Low	3.5	36.3%
Med	3.2	25.4%
High	3.0	15.7%

Table 2: Decoding performance on SGS8 for different resolution profiles (1 thread, 50K points per frame).

# Threads	Dec. FPS (Phone)	Dec. FPS (Server)
1	3.5	10.1
2	7.0	19.0
4	8.6	34.3
6	8.2	41.3
8	6.7	47.0

Table 3: Multi-core decoding performance on SGS8 and an edge server (low-res profile, 50K points per frame).

Similar to regular videos, there are two opportunities one can leverage: *intra-frame* compression, which compresses a single PtCl, and *inter-frame* compression, which delta-encodes a PtCl based on another PtCl (*e.g.,* of a previous frame). The above results as well as most existing studies only concern with intra-frame compression (as a result the compression ratios in Table 2 are low). Applying inter-frame compression can further reduce the video size, but at the cost of even slower decoding speed on smartphones.

Observation 4. Multi-core decoding provides limited performance improvement on smartphones. A straightforward idea to boost the decoding performance is to leverage the multiple CPU cores to concurrently decode multiple frames. We implement multi-threaded decoding in our player. Our SGS8 phone is equipped with an octa-core CPU. However, as shown in Table 3, when using 4 threads, the decoding FPS only increases by 1.5× compared to single-threaded decoding. Also, launching more than 4 threads causes the performance to degrade. This may be possibly attributed to multiple factors such as shared I/O and limited CPU cache. Note that PtCl decoding may also be accelerated by GPU. Nevertheless, GPUs on mobile devices are significantly weaker than those on PCs/servers due to the fundamental constraints of energy consumption and heat dissipation.

4 THE DESIGN OF THE NEBULA SYSTEM

We now detail the design of the Nebula system whose architecture is shown in Figure 1.

4.1 Video Content Organization

To store the PtCl content, Nebula employs a scheme that is compliant with DASH (Dynamic Adaptive Streaming over HTTP). Video frames are properly compressed using both intra-frame and inter-frame encoding. The PtCl stream is then segmented into *chunks* each having a fixed duration. A manifest file will be provided to the client to inform the URL of each chunk.

Layered Representation. In traditional DASH schemes, a chunk has multiple *independently* encoded versions corresponding to different quality levels. Nebula instead leverages the unique characteristics of PtCl chunks by organizing the chunk data (the points) into *layers*. Assume a PtCl chunk has n versions/layers. Among all versions, only the lowest version (layer) L_0, which contains the smallest number of points, is self-contained; the next lowest layer L_1 only contains the *delta* from L_0. In general, the client needs to fetch all layers from L_0 to L_i, and to combine their points to form the actual chunk data belonging to the version of the i-th quality level. Such a layered representation greatly improves the flexibility of the rate adaptation algorithm by allowing *incrementally* upgrading a chunk's quality (§4.3). For intra-frame encoding using an octree, different layers' encoded data can be generated by vertically partitioning the tree. The partition method for inter-frame encoding can also be developed, with the constraint that encoded data of a given layer should only depend on the same or lower layers of other frames within the same chunk.

We note that the underlying concept of layered encoding is not new. It has been proposed in the context of regular pixel-based videos, known as Scalable Video Coding (SVC). However, SVC has never registered commercial deployment due to its high complexity and high encoding overhead [15]. In contrast, our layered encoding scheme for PtCl is simple due to the very nature of PtCl, whose points belonging to different layers can easily be merged.

4.2 Edge Offloading

Our findings in §3 suggest that directly decoding a PtCl video stream on today's COTS smartphones might be challenging. Nebula therefore offloads the PtCl decoding to an edge server (proxy). Specifically, the client keeps reporting its viewport's position and orientation to the edge proxy. Meanwhile, the proxy fetches the PtCl stream from the remote server, decodes it, and transcodes it (based on the viewport's position and orientation) into a regular pixel-based video stream. The transcoded video is then sent to the client, which can efficiently decode the video through, for example, its hardware H.264/H.265 decoders.

We demonstrate that a commodity edge server is capable of decoding PtCl streams at the line rate. We repeat the decoding experiments in §3 on a commodity server with Intel Xeon E3-1240 processor at 3.5GHz (launched in 2015), 16GB memory, and 1TB SATA HDD. As shown in Table 1, for single-core decoding, the server outperforms our SGS8 smartphone by about 200% in terms of FPS. Table 3 compares the multi-core decoding performance between the phone and server. The server's performance scales well with the number of threads, and outperforms the phone by up to 450% under multi-core. For 12.6K, 25.2K, 50.4K, 75.5K, and 100.7K points per frame, our server achieves decoding FPS of up to 190, 95, 47.0, 29.2, and 21.4, respectively.

The Overall Processing Pipeline on the proxy side consists of the following five tasks: (1) fetching the PtCl data from the server (I/O bound), (2) decoding the PtCl stream (CPU bound), (3) rendering the PtCl stream based on the client's viewport position and orientation (GPU bound), (4) encoding the rendering result into a pixel video stream (DSP), and (5) sending the pixel video to the client (I/O). Note that the above tasks consume different types of system resources and thus can be executed in a pipelined manner.

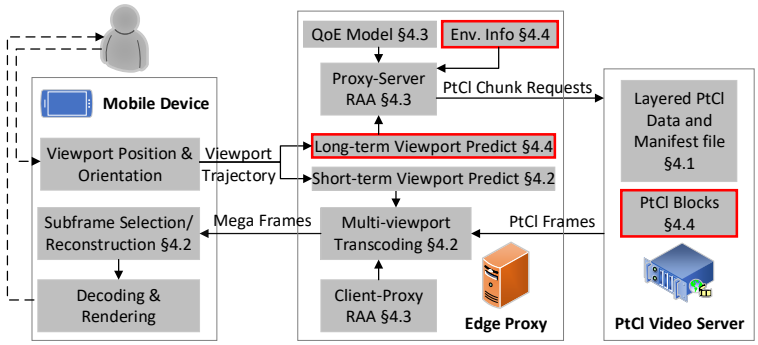

Figure 1: The Nebula **architecture. Components in red boxes relate to viewport adaptation (§4.4) that may be optionally enabled.**

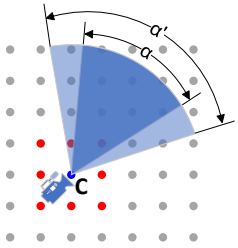

Figure 2: Transcoding a PtCl frame to multiple subframes, each with a different viewport, to reduce the motion-to-photon latency.

Now let us consider the actual transcoding process. A simple approach is to take a single "snapshot" for the current viewport position and orientation reported by the client. Let t_1 be the time when the client reports its viewport v, and t_2 be the time when the frame transcoded for v is displayed. Under the above transcoding approach, the delay $t_2 - t_1$ is the so-called "motion-to-photon" latency [2] (*i.e.,* the time taken for a viewport movement to be reflected on the screen).

We next examine this critical QoE metric. In the five tasks described above, Tasks (1) and (2) do not depend on the client's real-time viewport information so the decoded PtCl data can be buffered beforehand. Only (3), (4), and (5) contribute to the motion-to-photon latency. To quantify it, we conduct experiments on a DELL laptop (as an edge server) with an NVIDIA GTX 1080 GPU. On this laptop, rendering a PtCl frame with 50K points takes less than 1ms, and the per-frame encoding time for 4K, 2K, and 1080p H.264 videos is 10ms, 5ms, and 3ms, respectively. We then measure the per-frame decoding time on SGS8, which is 11ms, 5ms, and 3ms for 4K, 2K, and 1080p frames, respectively. Thus, the motion-to-photon delay for 4K, 2K, and 1080p resolutions are 21ms, 10ms, and 6ms, respectively, plus their corresponding network transmission time (including the uplink latency for reporting the viewport).

Multi-viewport Transcoding. We consider further reducing the motion-to-photon delay. Recall that when a motion occurs, the user's intended viewport at t_2 may be different from the displayed viewport *i.e.,* the one at t_1 (t_1 and t_2 defined previously). Such a discrepancy degrades the QoE. Our basic idea is thus to let the proxy transcode one PtCl frame into multiple pixel-based frames (called *subframes*), which include not only the reported viewport (at t_1), but also its nearby viewports for t_2 based on predicting the (short-term) viewport movement. Ideally, if the prediction is accurate, then at t_2, one of these subframes' viewports will match the user's intended viewport. Displaying it effectively eliminates the motion-to-photon delay. In reality, we anticipate that some of these viewports will be at least close to the user's intended viewport, thus reducing the impact on QoE.

Consider the two-dimensional example shown in Figure 2. The current viewport position is C and covers the dark blue area with a FoV (field-of-view) of α. When actually generating the subframes, the proxy will take two approaches. First, it generates subframes at nearby locations (in red dots) to accommodate the potential translational movement of the viewport. Second, the proxy may enlarge some subframes' viewports (*e.g.,* to $\alpha' > \alpha$) to tolerate the possible rotational movement. A larger-than-FoV subframe will be generated using a panoramic representation such as Equirectangular or CubeMap so any FoV covered by it can be restored.

To facilitate deciding which viewports to transcode, Nebula employs effective algorithms to predict the 6DoF viewport movement. This is inspired by the fact that in 3DoF panoramic video streaming, there were studies showing that the yaw, pitch, and roll can be accurately predicted in the near future (less than 0.5 seconds or so) [22]. Then, leveraging the *short-term* prediction algorithms, Nebula can dynamically decide how many and which viewports (subframes) to include, as well as each viewport's position and size (α'). Several factors affect the decision: the speed, direction, and mode (orbit, zoom, pan) of the viewer's viewport movement as well as its prediction accuracy.

Client-side Subframe Decoding, Selection and Reconstruction. The proxy then encodes the multiple subframes (transcoded from a PtCl frame) into a "mega frame" and sends it to the client. Due to their proximity, typically the subframes are visually similar, allowing efficient cross-subframe compression. During a playback, the client picks one subframe from each mega frame, decodes it, and displays it. To reduce the decoding overhead, Nebula can leverage new codec features such as the Tiling Mode [19] in H.265 to let the client decode only its selected subframe in a mega frame. Regarding the actual selection policy, the selected subframe's viewport should be the closest to the user's intended viewport. We will also investigate the feasibility of using fast 2D scene reconstruction [8] to achieve better viewport approximation by "interpolating" the received subframes when user's viewport C is not aligned exactly with the red dots in Figure 2.

4.3 Rate Adaptation

A rate adaptation algorithm (RAA) dynamically adjusts the video quality based on the available network bandwidth. In Nebula, due to the transcoding proxy, we need to consider two types of RAAs: one runs between the client and the proxy for the pixel video stream, and the other operates between the proxy and the server for the PtCl stream.

The Client-proxy RAA is overall similar to a traditional RAA. However, one key difference is the multi-viewport (subframe) nature in Nebula. Therefore a challenge here is how to assign quality levels to different subframes. A simple approach would be to treat

a mega frame as a single unit by assigning the same quality level to all its subframes. An advanced solution would be to assign different quality levels to them based on their probabilities of being selected.

The Proxy-server RAA differs vastly from existing RAA due to two reasons. First, it operates on PtCl streams whose many characteristics such as bitrate dynamics and QoE metrics are very different from those of traditional pixel-based videos. Second, recall from §4.1 that PtCl chunks are progressively encoded into layers; therefore, to be more bandwidth-adaptive, the RAA should have the capability of incrementally upgrading an existing chunk by fetching additional layer(s). We next describe two essential steps toward developing a robust RAA for PtCl videos: deriving QoE metrics and designing the actual RAA.

Deriving QoE Metrics. For traditional videos, a commonly agreed QoE metric is a weighted sum of several components: bitrate, stall duration, video quality switch, *etc.* [25]. QoE metrics for volumetric videos, however, still remain an open problem (see §5).

We exemplify several interesting research questions we would like to answer. (1) Compared to users watching regular videos (0DoF) and 360° videos (3DoF), are PtCl video viewers more sensitive to stalls as they exercise their 6DoF when navigating in the 3D space? (2) How do quality changes in a PtCl stream impact the viewer's QoE? (3) A new dimension in 6DoF videos is the distance from the viewer to the PtCl. As the viewing distance becomes larger, how should we reduce the PtCl quality while still maintaining a good QoE?

To get the QoE ground truth, we plan to conduct IRB-approved user studies. We will play strategically crafted PtCl videos to the subjects and ask them to give subjective scores. The ground truth will be leveraged to build a comprehensive QoE model of PtCl video streaming on mobile devices.

Designing RAA for Layered PtCl Stream. Among many candidate solutions, Nebula employs an RAA under a discrete optimization framework [25], which provides a principled way for rate adaptation. Specifically, it periodically examines a finite window of the next N chunks, and searches for a quality assignment that maximizes a utility function derived from the QoE model (the above task). Incremental chunk upgrade can be easily integrated into the optimization framework: even if a chunk has already been downloaded, it will still be considered by our RAA as follows. Assume a fetched chunk has a current quality level of l. The RAA will set a search space from l to the highest quality level. Then if the RAA computes a level higher than l, the additional layer(s) will be fetched to upgrade the chunk.

A challenge we need to overcome is the potentially high runtime overhead of our RAA, whose single invocation incurs an overhead of $O(Q^N)$ if an exhaustive search is performed (N is the search window size and Q is the total number of quality levels). Such an overhead may be effectively reduced by, for example, reusing previous computation results and applying various heuristics.

4.4 Viewport Adaptation

Use Case: a PtCl Zoo. So far we assume a PtCl video contains only one point cloud, whose entire stream is fetched by the proxy. We now consider a more general scenario where the entire scene, oftentimes consisting of multiple PtCls, is too large to be fetched as a whole. Consider a virtual zoo application. It is comprised of a static 3D background and many animals each being rendered as a PtCl video. At a given time, a user "walking" in the zoo can see only a small subset of all animals. For large animals such as a dinosaur, the user may see only part of it when standing close to it.

Streaming this virtual zoo is very different from and consumes much higher bandwidth than today's VR apps, whose dynamic objects typically consist of very short animation sequences [5]. To efficiently stream the PtCl zoo to a mobile device, our previous building blocks such as edge transcoding, rate adaptation, and layered representation remain useful. In addition, we need the mechanism of *viewport-adaptation* where the proxy prefetches only the content that the viewer is about to see. This is the key mechanism for reducing the server-to-proxy traffic and the proxy-side workload.

3D Point Cloud Blocks. A prerequisite of viewport-adaptive streaming is to segment (when necessary) a PtCl chunk into *blocks* [20]. A block has the same duration as a chunk but only occupies a smaller, bounded 3D region. With their URLs listed in the manifest file, blocks can be independently fetched and decoded, thus providing a mechanism of partially fetching a PtCl. In addition, as a block may move, the client needs to be aware of the block's position during its playback (we will shortly see why). To realize this, for each block, we include the moving trajectories of the eight corners of its *bounding box* in the manifest file.

Viewport Adaptation. Let us first assume an ideal scenario where the viewer's 6DoF movement trajectory is perfectly known. Then at any given time, by knowing the position and orientation of the viewport, as well as the positions of the eight corners of each block's bounding box, the proxy can quickly compute the set of blocks (more accurately, their bounding boxes) that the viewer sees and therefore should be fetched.

In reality, however, due to the viewer's movement randomness, we need to *predict* her 6DoF viewport movement. This prediction differs from the short-term prediction described in §4.2 (for multi-viewport transcoding) in the time scale: here the prediction needs to be more ahead of time (*e.g.,* 1 to 2 seconds) due to the extra delay the proxy takes to fetch, decode, and render the PtCl blocks. Unfortunately, even for 3DoF viewport movement, its predictability decreases considerably as the prediction window becomes longer [22]. We expect that *long-term* 6DoF prediction may also be challenging. We plan to conduct a user study that collects real users' 6DoF movement traces, and study their predictability.

Despite the potential challenges described above, we do believe it is feasible to design a practical viewport-adaptive streaming scheme by leveraging an edge proxy. We highlight several high-level design principles below.

- The proxy should account for both the bandwidth and its decoding capability when deciding which blocks to fetch.
- The proxy can use the environment (walls, obstacles, *etc.*) to quickly filter out blocks that the viewer cannot see.
- To minimize the risk of stalls, even if a block has a low probability of being perceived, the proxy may still need to conservatively fetch it at a low quality, and incrementally upgrade it later when necessary (§4.3).
- Distant blocks are shown in small sizes to the viewer, so they can also be fetched at a low quality per the QoE model. For such distant blocks, when the resource is limited, the proxy may even just fetch

a single frame that is statically displayed to satisfy a minimum user experience requirement.

5 RELATED WORK

Compression of PtCl and 3D mesh has been well investigated in the literature. In particular, several variations of octree-based (§2) compression have been proposed (*e.g.,* [7, 11, 24]). Kammerl *et al.* extended the octree to perform inter-frame compression for real-time 3D data acquisition [12]. Another inter-frame compression scheme based on iterative closest points (ICP) algorithm was proposed by Mekuria *et al.* [18]. We refer readers to [17] for 3D mesh compression.

Streaming volumetric videos is a new topic. Very recently (2018), Park *et al.* [21] sketched a greedy volumetric video streaming algorithm that considers video bitrate, visibility, and the distance from the viewer. DASH-PC [10] extends DASH to PtCl streaming. It proposes sub-sampling dense PtCls to create different quality representations, as well as designs a DASH-style manifest file format. Compared to both proposals, Nebula is a holistic PtCl video streaming system designed specifically for mobile devices, with unique features such as edge assistance, perceived delay reduction, principled rate adaptation, and incremental quality upgrade.

QoE Metrics have been well studied for regular videos, but remain an open problem for volumetric video streaming. Most of the existing work focuses on assessing the quality of a static 3D model, with the reference model known, using simple metrics such as point-to-point distance or angular similarity [4]. For volumetric videos, researchers have done limited subjective tests or simply used the above per-frame distortion metrics [6]. However, it is well known that (for regular videos) traditional image quality metrics such as PSNR and SSIM do not correlate well with subjective measures (QoE). The same likely holds for volumetric videos. We thus plan to thoroughly investigate their QoE metrics by considering the impact of, for example, stalls, quality changes, viewing distance, and the motion-to-photon delay.

VR and 360° Video Streaming Systems. Finally, there exist a plethora of systems on mobile VR and 360° video streaming. Representative research prototypes include FlashBack [5] (boosting mobile VR quality through caching rendered scenes), Furion [16] (cloud-assisted VR through separating foreground and background content), Rubiks [9] (tile-based 360° video streaming), and Flare [22] (another viewport-adaptive 360° video streaming system for smartphones with further optimizations). Compared to VR and 360° video streaming, PtCl streaming faces numerous challenges such as poor decoding performance on smartphones, a lack of rate adaptation algorithms, and the difficulty for predicting the 6DoF viewport movement, as well as unique opportunities such as the specific data structure of PtCl data. All these challenges and opportunities are considered in Nebula's design.

6 ON-GOING WORK AND CONCLUSION

Motivated by the poor PtCl streaming performance on smartphones, we present Nebula, a holistic system for high-quality mobile volumetric video streaming. Our central idea is to use an edge server to judiciously transcode a PtCl stream into a regular pixel-based video that can be efficiently transmitted to and decoded by mobile devices. We further describe various optimizations such as incremental quality upgrade, motion-to-photon delay reduction, principled QoE-aware rate adaptation, and viewport adaptation.

We are now prototyping Nebula according to our design detailed in §4, as well as conducting the IRB-approved user studies as described earlier. We will thoroughly evaluate Nebula using PtCl video content on real mobile devices and under diverse network conditions. We also plan to extend Nebula to support 3D mesh based volumetric videos.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their comments. We also thank Dr. Aakanksha Chowdhery for shepherding our paper.

REFERENCES

[1] AT&T Future of 5G Technology. http://bit.ly/2NRvNQF.
[2] Google VR – Fundamental Concepts. https://developers.google.com/vr/discover/fundamentals.
[3] Point Cloud Library (PCL). http://pointclouds.org/.
[4] E. Alexiou and T. Ebrahimi. Point Cloud Quality Assessment Metric Based on Angular Similarity. In *IEEE ICME*, 2018.
[5] K. Boos, D. Chu, and E. Cuervo. FlashBack: Immersive Virtual Reality on Mobile Devices via Rendering Memoization. In *MobiSys*, 2016.
[6] E. Dumic, C. R. Duarte, and L. A. da Silva Cruz. Subjective Evaluation and Objective Measures for Point Clouds – State of the Art. In *Intl. Colloquium on Smart Grid Metrology*, 2018.
[7] T. Golla and R. Klein. Real-time Point Cloud Compression. In *Intl. Conference on Intelligent Robots and Systems*, 2015.
[8] A. Hamza and M. Hefeeda. Adaptive Streaming of Interactive Free Viewpoint Videos to Heterogeneous Clients. In *MMSys*, 2016.
[9] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han. Rubiks: Practical 360-Degree Streaming for Smartphones. In *ACM MobiSys*, 2018.
[10] M. Hosseini and C. Timmerer. Dynamic Adaptive Point Cloud Streaming. In *ACM Packet Video*, 2018.
[11] Y. Huang, J. Peng, C.-C. J. Kuo, and M. Gopi. A Generic Scheme for Progressive Point Cloud Coding. *IEEE Trans. on Vis. and Computer Graphics*, 14(2):440–453, 2008.
[12] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach. Real-time Compression of Point Cloud Streams. In *Intl. Conference on Robotics and Automation*, 2012.
[13] L. Kobbelt and M. Botsch. A Survey of Point-Based Techniques in Computer Graphics. *Computers & Graphics*, 28(6):801–814, 2004.
[14] M. Kowalski, J. Naruniec, and M. Daniluk. LiveScan3D: A Fast and Inexpensive 3D Data Acquisition System for Multiple Kinect v2 Sensors. In *Intl. Conf. on 3D Vision*, 2015.
[15] C. Kreuzberger, D. Posch, and H. Hellwagner. A Scalable Video Coding Dataset and Toolchain for Dynamic Adaptive Streaming over HTTP. In *MMSys*, 2015.
[16] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *Proceedings of MobiCom 2017*, pages 409–421. ACM, 2017.
[17] A. Maglo, G. Lavoué, F. Dupont, and C. Hudelot. 3D Mesh Compression: Survey, Comparisons, and Emerging Trends. *ACM Computing Surveys*, 47(3), 2015.
[18] R. Mekuria, K. Blom, and P. 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.
[19] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou. An overview of tiles in HEVC. *IEEE Journal of Selected Topics in Signal Processing*, 7(6):969–977, 2013.
[20] A. Nguyen and B. Le. 3D Point Cloud Segmentation: A Survey. In *International Conference on Robotics, Automation and Mechatronics*, 2013.
[21] J. Park, P. A. Chou, and J.-N. Hwang. Volumetric Media Streaming for Augmented Reality. In *GLOBECOM*, 2018.
[22] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *ACM MobiCom*, 2018.
[23] H. Qiu, F. Ahmad, F. Bai, M. Gruteser, and R. Govindan. Augmented Vehicular Reality. In *Proceedings of MobiSys*, 2018.
[24] R. Schnabel and R. Klein. Octree-based Point-Cloud Compression. In *Euro. Symp. on Point-Based Graphics*, 2006.
[25] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *SIGCOMM*, 2015.