GROOT: A Real-time Streaming System of High-Fidelity Volumetric Videos

Kyungjin Lee
Seoul National University
jin11542@snu.ac.kr

Juheon Yi
Seoul National University
johnyi0606@snu.ac.kr

Youngki Lee
Seoul National University
youngkilee@snu.ac.kr

Sunghyun Choi
Samsung Research
sungh.choi@samsung.com

Young Min Kim
Seoul National University
youngmin.kim@snu.ac.kr

Abstract

We present GROOT, a mobile volumetric video streaming system that delivers three-dimensional data to mobile devices for a fully immersive virtual and augmented reality experience. The system design for streaming volumetric videos should be fundamentally different from conventional 2D video streaming systems. First, the amount of data required to deliver the 3D volume is considerably larger than conventional videos with frames of 2D images, even compared to high-resolution 2D or 360° videos. Second, the 3D data representation, which encodes the surface of objects within the volume, is a sparse and unorganized data structure with varying scales, whereas a conventional video is composed of a sequence of images with the fixed-size 2D grid structure. GROOT is a streaming framework with a novel data structure that enables not only real-time transmission and decoding on mobile devices but also continuous on-demand user view adaptation. Specifically, we modify the conventional octree to introduce the independence of leaf nodes with minimal memory overhead, which enables parallel decoding of highly irregular 3D data. We also developed a suite of techniques to compress color information and filter out 3D points outside of a user's view, which efficiently minimizes the data size and decoding cost. Our extensive evaluation shows that GROOT achieves more stable but faster frame rates compared to any previous method to stream and visualize volumetric videos on mobile devices.

CCS Concepts

• **Networks → Mobile networks**; • **Computer systems organization → Real-time system architecture**; • **Computing methodologies → Image compression**; **Mixed / augmented reality**.

Keywords

Volumetric Video, Video Streaming, Mobile Augmented Reality, Virtual Reality, Point Cloud

ACM Reference Format:
Kyungjin Lee, Juheon Yi, Youngki Lee, Sunghyun Choi, and Young Min Kim. 2020. GROOT: A Real-time Streaming System of High-Fidelity Volumetric Videos . In *The 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20), September 21–25, 2020, London, United Kingdom.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372224.3419214

1 Introduction

Volumetric video is an emerging media that provides a highly immersive and interactive user experience. Different from 2D videos and 360° videos, the volumetric video consists of 3D data, enabling users to watch the video with six-degrees-of-freedom (6DoF). Alongside with the recent advances in virtual reality (VR) and augmented reality (AR), volumetric videos conjure numerous compelling applications. For instance, music performances or sports games that are captured as volumetric videos will allow users to view their favorite performer or player from different angles and distances. It can also be utilized in various fields such as education for remote virtual lectures and even for medical, architecture, and arts. While volumetric video contents may consist of anything captured in 3D, from objects, animals, to humans, human contents are the most popular and yet challenging subject to handle. The market growth of volumetric videos is projected to reach $2.8 billion by 2023 [9], and it is considered as the key application of 5G [10].

Despite its potential, it involves several critical technical challenges to enable real-time delivery of volumetric videos to mobile devices.

• **Real-time Data Transmission.** Volumetric videos should be delivered to mobile devices over the wireless network to allow the users to move freely in 6DoF for an immersive experience. Volumetric video is most commonly represented as the *point cloud* format composed of a list of 3D coordinates and RGB colors. If the individual points are simply concatenated without compression, it enables rendering with fully parallel processing at the expense of large data size. Single frame size of a volumetric video can typically range from 4 MB to more than 15 MB (See Table 1), and naive, uncompressed delivery will require 1Gbps to 3.6Gbps transmission rate, which is beyond the capability of current WiFi or 5G networks [12, 39].

• **Real-time Decoding and Rendering on Resource-constrained Mobile Devices.** For the users to watch volumetric videos in 6DoF, the content should be decoded and updated in 30 fps while the viewpoint of the user is updated in 60 fps (<20ms of

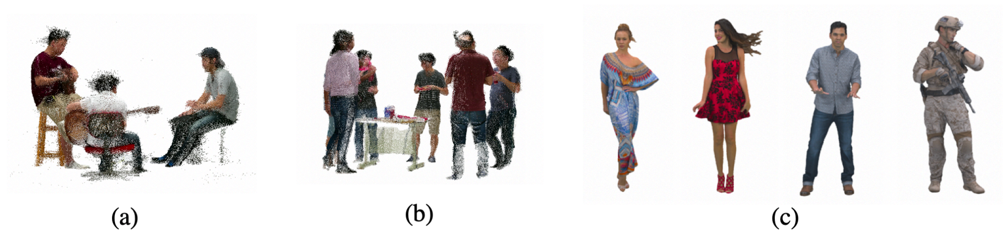


Figure 1: State-of-the-art volumetric video samples from CMU Panoptic Dataset (a, b) [29] and 8i Voxelized Full Bodies (c) [21].

motion-to-photon latency). There has been extensive research in 3D volumetric video compression [19, 22, 30, 50, 56], but system design for real-time streaming has not been explored sufficiently. For instance, Google Draco [2] and Point Cloud Library (PCL) [8], widely used open-source libraries to compress volumetric videos, decrease the data size by 4×, but the decoding speed is far from reaching the 30 fps frame rate even on state-of-the-art mobile devices such as iPhone XS (See details in Section3.3). Recently, ViVo [14] took the first step to enable the streaming of volumetric videos to mobile devices. The system applies visibility-aware optimizations to reduce the network bandwidth usage on average of 40% while improving the decoding speed using multi-threading. However, the underlying decoder is also based on Google Draco, which is hard to scale when decoding complexity increases with a larger number of points; it supports real-time streaming with up to 300k points but is hard to scale when the number of points and resolution increase.

To address the challenges, we propose GROOT, an end-to-end streaming pipeline for volumetric videos. We observe that for seamless and real-time streaming of volumetric videos, it is critical to compress the large amount of 3D data and, at the same time, enable fast and light-weighted decoding on the mobile device fully exploiting the parallel nature of point clouds. We suggest a novel compression and decoding scheme, which not only improves the compression rate but also supports real-time decoding on mobile devices. In addition, it can be modified in real-time for runtime optimizations such as user view adaptation to further reduce the data size while maintaining a sufficient perceptual quality. GROOT can stream state-of-the-art videos at a 30 fps frame rate while maintaining the rendering frame rate at 60 fps, whereas the prior systems support only 2 to 20 fps depending on the volumetric video content.

In particular, GROOT features the following three novel techniques.

• **Real-time Parallel Decoding.** We develop new data compression and decoding techniques specialized for volumetric video streaming with a newly designed tree structure, *PD-Tree*, Parallel Decodable Tree. We take the *octree*-based approach [30] as the baseline, which compresses the point clouds in a tree format recursively dividing the 3D space into 8 sub-spaces. The problem with the original octree structure is that the inter-dependency from the root to the leaf node prohibits the parallel decoding and significantly delays the decoding process. The suggested PD-Tree removes dependencies between the different branches of the tree, and we demonstrate the power of the data structure with a high-speed GPU-enabled decoder integrated into the rendering pipeline. The new compression and decoding techniques enable 30 fps streaming speed, which was not possible with the original tree-based compression techniques such as Google Draco [2] or PCL [8]. Note that we excluded 2D projection-based compression methods such as MPEG V-PCC [7]. The approach enables fast decoding based on long-studied 2D video compression techniques, but it is difficult to encode generic point cloud videos with multiple overlapping people and to apply user-adaptive optimization techniques.

• **Color Map Encoding.** We develop a color map encoding technique to further reduce the size of the compressed video. The prior methods focus more on compressing coordinates of 3D points that the size of the color map is 2× of the coordinate information after the compression. To overcome the low compression rate, we pack the color values as 2D pixels of images and utilize the conventional JPEG compression. Here, we take advantage of the PD-Tree, where the leaf nodes can be processed in parallel, and therefore the order of the points can be flexibly re-arranged. We position similar colors into 8×8 blocks within the image to maintain the visual quality while achieving a higher compression ratio.

• **Continuous and Responsive User-view Adaptation.** Finally, we show that it is possible to adopt runtime optimization techniques such as user view frustum culling and resolution adaptation directly to the streaming pipeline without modifying the decoder. Although prior systems like ViVo [14] has applied the similar idea, our paralleled structure is unique in that: 1) it provides much faster updates of the content when a user's view needs to change, and 2) it enables continuous changes of the contents to effectively reduce the data size while maintaining the perceived quality of the videos.

The summary of our contributions is as follows:

- To our knowledge, GROOT is one of the first mobile volumetric video streaming systems. In particular, our system features in supporting real-time streaming of state-of-the-art volumetric videos (with a large number of sparse points or high resolutions).
- We develop a suite of techniques, real-time parallel decoding, color map encoding, and continuous and responsive user-view adaptation. These techniques enable real-time streaming of volumetric videos by reducing the data size and improving the decoding speed on resource-constrained mobile devices.
- We implement a prototype of our proposed system and conduct thorough experiments with the existing state-of-the-art datasets. GROOT achieves a 30 fps frame rate and 60 fps motion-to-photon latency, which is 9.4× and 3× better than optimized Google Draco and PCL, respectively. GROOT also improves the compression rate by 2×.

Table 1: Datasets.

Name	Dataset	Description	Avg # of Points/Frame	Total # of Frames	Required Bandwidth (Gbps)	Avg Rendering (FPS)
band	Panoptic	Three people playing instruments	288k	3000	1.03	390
pizza	Panoptic	Five people standing in a circle	515k	3000	2.31	259
longdress (long)	8i	One person moving	860k	300	3.09	176
twopeople (two)	8i	Two people moving	988k	300	3.55	126

2 Background

2.1 What is Volumentric Video?

Creation. Volumetric video is composed of a sequence of 3D data allowing the users to move in 6DoF and interact with the 3D contents from any direction and location. The real-world information can be turned into a volumetric video by capturing the 3D information in real-time using multiple RGB-D cameras. It is becoming accessible for general developers with open-source capture libraries [3, 6] using commodity depth sensors (e.g., Intel RealSense, Microsoft Kinect). Specifically, volumetric videos can be generated by combining multiple calibrated and synchronized RGB-D cameras observing the scene from various viewing angles. 3D acquisition with RGB-D cameras is becoming ubiquitous as the quality and resolution of the RGB-D measurement are improving. The sensors can be deployed in an everyday environment with minimal cost, including the recent generation of smart-phones. Consequently, the volumetric video is expected to be a wide-spread medium to provide a fully immersive experience allowing the users to walk around and interact with the 3D contents.

Representations. Unlike 2D images that are composed of 2D *pixels* (picture elements), the 3D data can be represented in various formats, such as splines, implicit surfaces, a grid of voxels (volume elements), polygonal meshes, or a point cloud. In particular, the raw measurement is represented as point clouds, which is a list of 3D points where each point is represented by a 3D coordinate ((x, y, z), 4 bytes each) and its corresponding attributes such as color values ((R, G, B), 1 byte each) or normals. The representation results in a linear increase in data size and does not encode any dependency or correlation between neighboring measurements. While we can post-process the point cloud data to transform into other data formats for rendering, the point cloud format is popular due to its simplicity and its flexibility to represent non-manifold structures, which can better represent the captured real world [50, 52].

Dataset. Human subjects are the most popular content of volumetric videos for compelling applications such as telepresence or virtual events. It is also the most challenging due to the complex and diverse shapes and continuous movements. Existing state-of-the-art volumetric videos containing human subjects can be divided into two categories. One is large-scale captures with multiple people and objects but sparse point density such as the Panoptic dataset [29]. Another type includes higher quality captures of individual people with a much denser number of points, for example, the 8i dataset [21]. Table 1 describes the characteristics of each dataset, and we design our system to support both types.

2.2 Streaming Volumetric Video

Streaming volumetric video includes sending, decoding frames of 3D data at a minimum of 30 fps data rate, and rendering in 60 fps. The data transmission is bounded by the wireless network, and the decoding and rendering complexity is bounded by the capability of the mobile device. Compared to conventional video streaming, where extensive progress has been made with a vast amount of technologies [26, 32, 46, 48], only a limited number of research is available for 3D videos. Most of the techniques for conventional videos cannot be directly applied to the volumetric video because of the fundamental difference of the 3D frames, namely, the lack of regularity of data format and the large data size. The recently proposed techniques for 3D data compression suffer from the trade-off between the compression ratio, perceptual quality, and the decoding latency to meet both the requirement on the network bandwidth and the mobile processors. We further analyze the existing representative techniques for 3D video streaming in the next section.

3 Motivational Studies

The volumetric video needs to be compressed to meet the bandwidth requirement. There are two mainstream approaches for compression: 2D projection-based (Sec. 3.2) and 3D tree-based methods (Sec. 3.3). The former extracts surface patches from 3D models and packs them into a 2D frame, whereas the latter directly handles 3D data with the help of spatial data structures. Both of the approaches face challenges for volumetric video streaming, mainly due to the complexity of the 3D data.

3.1 Raw Data Streaming

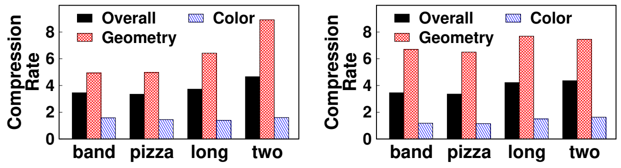
When the individual frames are in the format of the point cloud, the data size increases linearly by the number of points. This implies that with large-scale volumetric captures that may consist of 300k points to 1M points, the required bandwidth for streaming can vary from 1.08 Gbps to 3.6 Gbps. Even with the current state-of-the-art Wi-Fi or 5G network, it is not possible to send the raw data in 30 fps. Therefore, volumetric video compression is necessary.

3.2 2D Projection-based Compression

Representation. There is an ongoing standardization activity in volumetric video compression by the Moving Picture Experts Group (MPEG) under the name MPEG V-PCC [24, 50]. The key idea is to decompose the point cloud into multiple patches based on their surface normal information and project them onto a 2D image by dense packing. Then, existing 2D video codecs such as HEVC can be applied for a high compression rate and real-time decoding. The

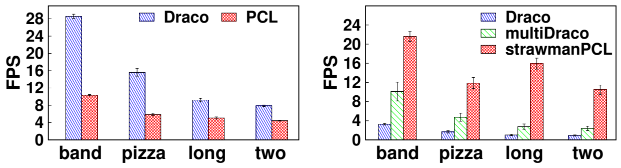
Table 2: Average data size and encoding/decoding latency with MPEG V-PCC.

Scheme	Data Size(MB)	Enc(min)	Dec(fps)
lossless	1.46M	42	3.2
lossy	0.19M	11	7.28



(a) Draco. (b) PCL.

Figure 2: Compression ratio of baseline systems.



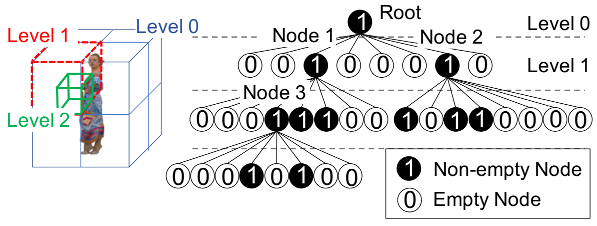
(a) Desktop PC. (b) Mobile Device.

Figure 3: Decoding performance of baseline systems.

3D geometry can be reconstructed at the client-side with three individual streams of images including: 1) a grey-scale image with each pixel representing the depth value of the points in each patch, 2) the 2D projected color textures, and 3) the occupancy image where each pixel has 1-bit information to indicate if the corresponding pixel is a valid point for reconstruction.

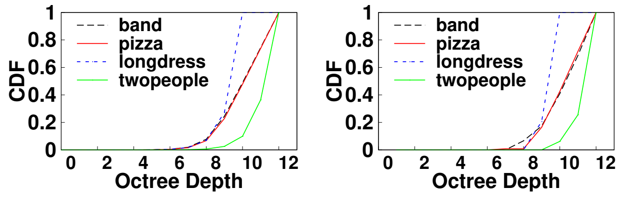
Performance. Since the standardization is in progress, only test software is available online. We test the performance of the codec with the datasets in Table 1 on a Desktop computer equipped with Intel Core i7-8700 3.2 GHz CPU. Table 2 shows the summary of the performance tested on 30 frames from the *longdress* dataset. For lossless compression, an average of 1.46 MB is required per frame while lossy compression can reduce the size to 190 kB per frame. The lossy compression significantly reduces the data size by compressing the projected images with intra- and inter-frame coding using 2D video codecs. While the encoding speed is very slow, requiring 11 (lossy) to 42 (lossless) minutes to encode a one-second video, decoding is relatively fast since it uses conventional 2D video codecs with hardware acceleration. Recently, results in [49] show that it can support 30 fps playback of a volumetric video containing a single person on mobile devices, but the resolution is limited.

Limitations. The 2D projection-based method suffers from several limitations, which makes it not applicable for large-scale and generic point cloud videos. First, sparse and noisy captures in Figure 1 (a) and (b) cannot be properly encoded with the codec. This is because it is difficult to extract surface normals from the noisy data. Also, even when normal patches are generated, the 2D projection is a collection of sparse points rather than a smooth image of texture. Next, for complex scenes with multiple objects and people (i.e., 4 people in Figure 1(c) standing close to each other), multi-layer projections need to be generated which increases the decoding



Index	0	1	2	3	...
Node	Root	Node 1	Node 2	Node 3	...
Occupancy Byte	00100010	00100010	10110000	00010100	

Figure 4: Octree data structure.



(a) Number of octree nodes. (b) Decoding latency.

Figure 5: Octree data structure breakdown.

complexity, and the loss of information is inevitable where 2D projections overlap.

3.3 3D Tree-based Compression

Representation. In order to represent the large volume of 3D data, *k*d-tree [16] or octree [35] data structures are used to exploit the sparsity of the representation. We describe the octree structure in more detail. (*k*d-tree is similar, but implemented as a balanced tree with varying size of nodes, where an octree node creates its children by uniform partitions.)

Starting from a tight bounding cube of the 3D space as the root node, octree divides the non-empty nodes recursively into 8 equally-sized child nodes by dividing each dimension into its half ($(1/2)^3 = 1/8$). As in Figure 4, the occupancy of 8 child nodes is represented as 8 bits of a single byte. Only the nodes that contain points should be further divided, and empty nodes can be ignored. With the center value and side length of the parent node, the location and size of the 8 child nodes can be automatically calculated.

The leaf nodes are where the actual points are saved and the center values of the leaf nodes approximate the original x, y, z coordinates. Therefore, the resolution of the point cloud is determined by the maximum depth of the tree. By traversing towards the leaf nodes, the octree can represent the point cloud without saving the individual coordinates. When using this octree data structure to deliver point clouds, the data structure is represented as a serial byte stream concatenating the occupancy byte of each octree node that can simulate the traversal.

Performance. Two of the most commonly used tree-based point cloud compression libraries are Google Draco [2] and Point Cloud Library (PCL) [8]. Draco is a state-of-the-art library specialized for mesh and point cloud compression, while PCL is a generic library for point cloud processing, which also has compression features. We first test the performance of Draco and PCL on a desktop server equipped with Intel Core i7-8700 3.2 GHz CPU. Figure 2 shows that both Draco and PCL could only reduce the overall data size by a

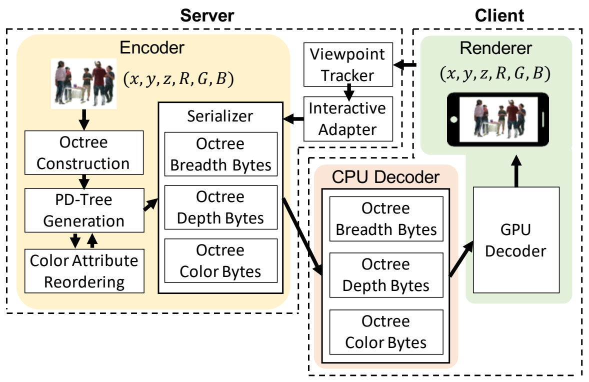


Figure 6: Groot system architecture.

factor of 3.35× to 4.22×. In particular, the color compression rate is limited to 1.5×. Furthermore, Figure 3(a) shows that the decoding speed is below 30 fps for all datasets. Next, we measure the decoding speed on a state-of-the-art smartphone, iPhone XS equipped with a Hexa-core CPU (2×2.5 GHz Vortex + 4×1.6 GHz Tempest). To further improve the decoding speed, we apply several existing optimization techniques such as multithreading and leverage SIMD CPU architectures supported in most latest smartphones [1, 4]. See Section 8 for the details of our implementation. Despite the applied optimizations, the decoding speed is limited to 2 fps to 20 fps, as shown in Figure 3(b). It is worth noting that multiDraco, which is Draco with multithreading, cannot ensure stable performance since the CPU needs to run multiple background jobs for rendering (i.e., generating vertex and color buffers for GPU rendering, tracking device positions, etc.) and wireless networking.

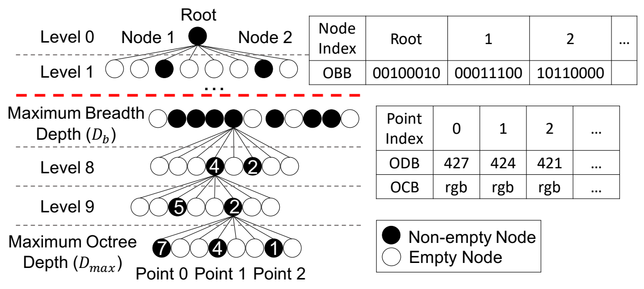
Challenges. While the tree structure efficiently handles the sparsity of 3D data and subdivides only the volumes where the point exists, the irregularity requires traversing the serialized byte stream and recursively calculating the child node geometry. The exact positions that represent intermediate nodes depend on the occupancy of their ancestors, which cannot be parallelized.

Figure 4 shows an example of an octree and the corresponding occupancy byte stream. The root node occupancy byte (index 0) shows that it has two child nodes, whose occupancy bytes are the following two bytes (index 1 and 2). Similarly, the occupancy byte of child nodes of Node 2 is located after the occupancy bytes of Node 1 child nodes in the bytes stream. This implies that the number of nodes at each octree depth is determined by the number of occupied bits among the nodes at the parent-depth, which requires sequential decoding. Especially with the increasing number of points and octree depth, the decoding time becomes a bottleneck in the streaming pipeline. When tested on the dataset, it is observed that the last few octree depths contain the majority of the octree nodes and therefore is the dominant cause for the decoding latency (Figure 5). In addition, the tree-based algorithm only considers the geometry compression of individual frames without color information.

3.4 Summary

Our observations can be summarized as follows.

- MPEG V-PCC has a high compression ratio with a fast decoding rate but fails to encode generic volumetric videos.



Node Index	Root	1	2	...
OBB	00100010	00011100	10110000	

Point Index	0	1	2	...
ODB	427	424	421	...
OCB	rgb	rgb	rgb	...

Figure 7: Architecture of PD-Tree.

- Octree data structure is efficient in handling and compressing geometry data. However, the data size and decoding complexity increase with the depth of the tree and the number of points.
- Existing 3D tree-based open source point cloud codecs suffer from a low color compression ratio.

4 GROOT System Overview

Based on the observations, we present GROOT, an interactive and continuous user-adaptive volumetric video streaming system that enables fast decoding for both geometry and color with our novel data structure. The proposed system considers the following criteria:

3D Data Delivery. Our primary goal is to deliver volumetric videos on mobile devices in the form of 3D data without a 2D projection. This is to support a fully immersive and interactive user experience for general large-scale 3D scenes and to meet the motion-to-photon latency (<20ms) requirement of rendering [18].

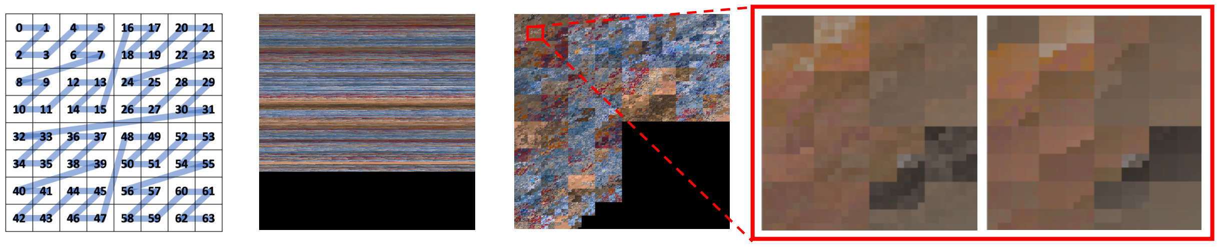
Real-Time Streaming Over Wireless Network. The users should be able to freely explore the 3D space in an untethered environment (i.e., without a wired link). Hence, we aim at delivering volumetric videos over the wireless network to mobile devices in real-time (e.g., 30 fps).

Interactive Runtime Optimization. We leverage a continuous and interactive viewpoint adaptation to reduce the data size and the amount of computation while maintaining the user perception quality under diverse and dynamic user viewpoint in 6DoF.

4.1 System Architecture

The overall system of GROOT is shown in Figure 6. We first redesign the codec to enable parallel decoding on mobile devices. The encoding process includes generating the *Parallel Decodable Tree* (PD-Tree) from the original octree structure (Section 5.1). The proposed PD-Tree represents nodes individually with minimal overhead, which allows the re-ordering of high-resolution child nodes and decreases the decoding complexity. This enables fully-parallel decoding on the mobile device, leveraging the existing rendering pipeline for point clouds on the GPU (Section 5.3). In addition, the color information can be significantly compressed by packing the pixels into an image to maximize the locality of similar color values (Section 5.2).

In order to further reduce the data size and decoding complexity, we can apply the two interactive run-time optimizations given the PD-Tree with minimal overhead. As the server predicts the viewport from the received viewpoint information, real-time frustum



(a) Morton ordering (b) Raster order-packed *longdress* frame (c) Morton order-packed *longdress* frame (d) Unsorted (left) and sorted (right) colors with Morton order packing

Figure 8: Comparison of color packing methods.

culling removes the points not seen by the user (Section 6.1). Also, we apply resolution adaption depending on the distance between the 3D content and the user in continuous levels to avoid perceiving a discrete quality degrade while moving (Section 6.2). The aforementioned two optimizations can be applied to our encoder in real-time since the points are individually represented with PD-Tree.

5 Real-time Streaming Pipeline

5.1 PD-Tree-Enabled Encoding

We design a new tree structure named *Parallel Decodable Tree*, or PD-Tree, to overcome the limitations of existing octree-based compression and decoding. As shown in Section 3.3, the compression rate and decoding speed of octree-encoded 3D geometry are throttled by the number of octree nodes residing at the last few depths of the tree. The PD-Tree overcomes this issue by modifying the octree data structure to enable fast parallel decoding while it inherits the benefit of the compression power of the octree structure.

To compose PD-Tree, we first split the octree structure into two parts at the *Maximum Breadth Depth* (D_b), which is predefined depending on the dataset. We empirically analyzed that in most datasets, the decoding speed drastically slows down in the last three depth layers. Therefore, we set $D_b = D_{max} - 3$ where D_{max} is the maximum octree depth. The occupancy bytes of the tree is serialized and compressed in two different ways as follows.

Octree Breadth Bytes (OBB). Starting from the root of the tree up to the nodes at D_b, we encode the occupancy bytes as *Octree Breadth Bytes* (OBB), which is the breadth-first serialization used in conventional octree-based compression, also shown in Figure 4. The serialized data, which includes the occupancy bytes of non-empty octree nodes in the breadth-first order, can be traversed and decoded into a list of (x, y, z) coordinates. These coordinates are the center positions of octree nodes at depth D_b, which serve as the root nodes for the sub-trees representing the deeper depths. Therefore, the nodes at depth D_b has the quantized (x, y, z) coordinates of the 3D content with the resolution of D_b bits.

When occupancy bytes are encoded in the breadth-first order, the decoding process cannot be fully parallelized as the exact byte locations of intermediate nodes depend on the number of non-empty nodes in previous depth layers. However, as analyzed in Figure 5, the decoding latency for depths smaller than D_b is under 20% for all datasets, and the serial decoding latency is negligible.

Octree Depth Bytes (ODB). Instead of sharing the traversal from their ancestors, the leaf nodes are individually encoded with their paths from the root of the sub-tree at D_b, breaking the dependency. *Octree Depth Bytes* (ODB) encode the path to each leaf node with the concatenation of the occupancy bytes with a single bit set. Figure 7 shows an example of ODB to represent Point 0, 1, and 2, which are leaf nodes that represent the final location of the points. Point 0, 1, and 2 are represented by the 7th, 4th, and 1st bit of the occupancy byte of the common parent node in Level 9. In the same way, the parent node in Level 9 is represented by the 2nd bit of its parent node in Level 8, which is represented by the 4th node in its parent node at D_b.

Traversing back down from the node at D_b, Point 0, 1, and 2 can be represented by the three sequences of numbers 427, 424, and 421 reusing the representations of the shared parent node. Since each number in this representation is an integer value in the range of 0 to 7, the three-number sequence can be encoded into 9 bits (i.e., 3 bits each). When unpacking a 9-bits sequence back to a 3-byte representation, it incurred an additional decoding complexity on the client-side. Thus, we employ 4-bits encoding rather than 3-bits for faster decoding.

The independent representation allows parallel decoding for individual points, greatly reducing the decoding latency. Furthermore, it enables sampling and removing of points in real-time for user-interactive encoding, because the order and density of the leaf nodes can arbitrarily change within the subtree rooted at a node at D_b.

5.2 Octree Color Bytes (OCB) Compression

We additionally develop the color map encoding technique to further reduce the size of the compressed video. The prior techniques using 3D trees focuses only on compressing geometry, making the color map size twice bigger than that of geometry information after the compression. This makes the color map as the new bottleneck to send volumetric videos. Since PD-Tree enables fast decoding of the geometry data with a high compression rate, more sophisticated methods can be applied to color compression while meeting the real-time requirement of 30 fps. The key idea of our color map encoding is to compress the color values as 2D pixels of images and utilize the conventional JPEG compression. Note that the unique structure of PD-Tree, where the leaf nodes can be processed in parallel, enables fast and flexible re-ordering of points for the color map encoding.

The core challenge in the image-based color compression is to put neighboring points in 3D video contents (having similar colors) in

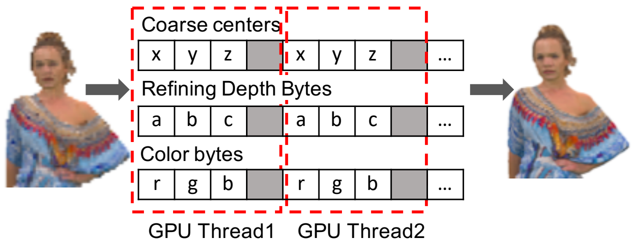


Figure 9: GPU-assisted decoding system of GROOT**.**

the 2D color image. One naive way of packing an image is following the raster scan order [38], which is a row-by-row scanning of an image from left to right, for the leaf node colors of the tree. This yields poor compression because of the abrupt color change in the vertical direction. [36] uses a similar approach but applies serial packing within smaller blocks. However, it does not fundamentally solve the problem of the abrupt color change in the vertical direction. Instead, we employ the *Morton Order* [55] to maximize the locality of adjacent point colors within 8×8 pixel blocks, which is the processing unit for JPEG compression. Figure 8 shows the schematic of Morton order packing (a) and the packing results with raster scan (b) and Morton order (c), respectively.

We can further cluster similar colors for a higher compression rate due to the unique structure of PD-Tree. Leveraging the independence of the leaf node representation, the points residing in the same octree node at D_b can be reordered arbitrarily. When the colors of the points are reordered, corresponding ODB are sorted accordingly. Figure 8(d) shows the closed-up result before and after reordering the colors. Reordering generates a smoother pattern of colors, thus enhancing the compression performance. The image size for packing can be determined by the final number of points. In our current system, the image size is chosen between 512×512, 512×1024, or 1024×1024 pixels.

5.3 Low-Latency Parallel Decoder

The decoding complexity of point cloud videos increases with a larger number of points and octree depth. The current decoding of the volumetric video is usually processed on the CPU, and the state-of-the-art CPUs cannot support a decoding rate of 30 fps for more than 300k points. Therefore, in GROOT, we leverage the parallel processing ability of the GPU to overcome the limitation. The encoded bitstream of the PD-Tree includes OBB and ODB, and the corresponding color information represented with OCB. The hybrid decoding pipeline utilizes both CPU and GPU on a mobile platform to reconstruct the 3D point cloud from the compressed data. First, the CPU parses the header information and handles the serialized bitstream of OBB. The OCB is executed by the JPEG decoder with dedicated hardware in commodity mobile devices. Then, the GPU decodes the ODB to refine the point cloud location and render them directly in a completely parallel manner.

Frame Structure. Each frame consists of a header and a payload that has the necessary information for the decoding process. The header includes the meta information of the frame, such as the number of final points, length of OBB, ODB, and OCB, and the coordinate and size of the root node. The payload contains four separate streams of data, including OBB, ODB, OCB, and lastly, a list of numbers of leaf nodes that reside in each octree node at D_b. With this information, the leaf nodes of the same sub-tree for ODB can be pointed to the correct root node coordinates at D_b.

CPU Decoder. The CPU first decodes the OBB stream, which recursively computes the child node location until it reaches the depth, D_b. Next, the 4-bit encoding of ODB is decoded to regenerate the byte stream that represents each point independently (i.e., 3 bytes per point, as in Figure 7). The OCB is decoded by the JPEG decoder [5], and the decoded 2D image is re-ordered by the Morton ordering, which is saved as a lookup table (LUT), into the original color stream aligned with the ODB.

GPU Decoder. We include the refinement of ODB within the point cloud rendering pipeline, which is executed on the GPU. Specifically, we implement the GPU decoder with the vertex shader, which is designed to run in parallel on the GPU during the conventional rendering pipeline. The original rendering pipeline of point clouds receives an aligned list of vertex and color data and render them individually on the screen by multiplying the view and projection matrix within the vertex shader. We implement the decoding of ODB for individual points within the same vertex shader for synchronized coordinate refinement and rendering.

To utilize the GPU rendering pipeline, the input list has to be composed of independent representations of points, which are aligned by a multiple of four bytes, as shown in Figure 9. We rearrange the data such that the individual points are composed of the decoded node location of OBB at depth D_b, three bytes of ODB, and an additional three bytes for unpacked RGB color of OCB. To meet the 4 bytes memory alignment requirement, we add one-byte dummy values each. Since the intermediate node locations from OBB are shared by the leaf nodes of the sub-tree, the locations are duplicated and aligned with the corresponding ODB information to create an independent representation of points. The time overhead incurred by the additional processing is negligible, averaging below 1 ms. Furthermore, the integrated pipeline guarantees that the vertex and color attributes of the same frame are synchronized, while having separate pipelines for decoding and rendering will require explicit synchronization of the two.

6 Interactive User-View Adaptation

Apart from optimizing the underlying codec, additional optimization techniques are required to handle high-fidelity volumetric videos with a large number of points and high resolutions. In this light, we develop a new interactive user view-adaptation technique, further reducing the size of the video and improving the decoding speed.

A similar idea has been adopted for 360° video streaming systems [26, 46] and early approaches of volumetric video streaming such as ViVo [14]. We are inspired by such prior techniques, but our adaptation technique is unique in that it enables *responsive* and *continuous* adaptation. This allows the smooth transition of the scenes when a user takes a different view by moving her head, which was not possible with prior techniques.

We develop two specific techniques to reduce the number of points without affecting the perceived quality: i.e., *frustum culling* that removes the points outside of the user's view and *depth-based*

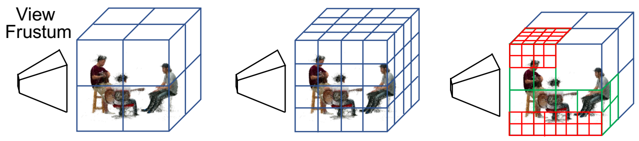


Figure 10: Comparing frustum culling with fixed-size grids (left, center, $1m \times 1m \times 1m$, $0.5m \times 0.5m \times 0.5m$) and hierarchical culling using the octree structure (right).

sampling that reduces the resolution of far away points. These techniques are designed based on the advantages of PD-Tree, which enables fast determination whether a point is in the user's current view or not and also direct removal of points from the encoded stream.

6.1 Frustum Culling

Removing the unseen points from the current user's viewpoint, known as frustum culling, is a straightforward optimization technique to reduce the data size and decoding complexity. However, the difficulty lies in applying this view adaptation in real-time at the server since the user viewpoint can change dynamically. The hierarchical structure and the independent representation in PD-Tree enable this with minimal processing overhead at the server.

Frustum culling is a widely used technique in graphics rendering, where only the geometry that falls within the 3D viewing frustum is displayed [13, 17]. Whether or not a point falls in the view frustum can be determined by the relative position of the point to the view frustum which is defined as the intersection of half-spaces that the six planes of the frustum creates. Exhaustive testing for all the points in the video and removing them from the encoded stream requires a non-negligible amount of computation. A common way to accelerate the calculation is using a grid of the sub-volumes and test the inclusion, similar to 2D video tiles as in Figure 10. User-adaptive approaches in [27, 42, 43] and ViVo [14] also used fixed-size 3D blocks. However, since 3D data is sparse, the large block size will not be able to sufficiently remove unnecessary points, while small block size will incur a large computation overhead.

Therefore, we utilize the hierarchical structure of PD-Tree for fast calculation even under a significant motion within the scene of varying density of points. Starting from the root node of PD-Tree, we recursively check if the child octree node intersects with the view frustum. When all of the eight corners of the parent node are included (or excluded) in the view frustum, the child nodes do not need further validation. Only when a subset of eight corners is included, we proceed with the test to its child nodes.

To handle user viewpoint prediction errors, we set a non-zero threshold for the half-spaces of the frustum to generate a bigger frustum that includes the actual user view frustum, automatically creating a margin around the edges of the view frustum. Also, GROOT sets the maximum depth to 4 or 5 to stop the frustum culling where the smaller node size has no effect in further reducing the data size and only generates an additional processing overhead at the server (See Section 8.5 for the analysis). After frustum culling, points that are included in the culled nodes should be removed from the ODB and OCB streams. Since the points are represented independently, it is possible to remove the points directly by indexing the corresponding leaf nodes for each culled nodes. Therefore, no modification is required on the client-side to decode the modified frame.

6.2 Depth-based Sampling

Given the list of culled octree nodes at D_b, we apply continuous depth-based sampling to stream only a subset of points from the list of ODB rooted at D_b with minimal impact on the perceptual quality. The basic intuition is that, with the fixed screen resolution, the density of observable points at a particular depth is inversely proportional to the depth of the point. A common way to apply depth-based sampling is to stop the tree traversal at a particular octree depth. However, as shown in Figure 5, the number of octree nodes increases abruptly between a single depth, especially in the last few depths, which can result in a sudden drop of perceptual quality. Also, most existing work such as [27] and [14] uses fixed-size blocks with predetermined density as in conventional 2D videos [26]. Since the user's viewpoint is much more diverse and changes dynamically, using fixed-size blocks can either not effectively remove unseen points or drop the perceptual quality significantly.

With PD-Tree, we enable a finer level of density change since each point is represented independently, and removing a subset of points does not affect the encoding and decoding pipeline. During runtime, the sampling density is determined for each node at the maximum frustum culling depth regarding the depth value of the center position of the node. However, since each device has different screen resolution and field-of-view, the rendering quality is not consistent across devices even when the points have the same depth value. Therefore, we determine the sampling density based on the depth value in the device-independent Normalized Device Coordinate (NDC) space, which maps the perspective frustum to a fixed-size cube. The location of the point in NDC space can be calculated with the model, view, and projection matrix which are continuously measured on the client device and updated to the server periodically. For fast and simple sampling, we determine the resolution by rendering or not rendering 1 out of $N(= 1, 2, 3, 4, 6, 8, 10)$ points generating a gradual resolution change of 12 levels: from 10%, 13%, 17%, 25%, 33%, 50%, 67%, 75%, 83%, 87%, 90%, 100%. Finally, we divide the cube into 12 sections and allocate the sampling density, which can maintain sufficient perceptual quality (i.e., SSIM value of 0.98).

7 Implementation

Client. We implement the GROOT client on iOS devices. We use continuous camera position tracking results provided by Apple's ARKit (camera.viewMatrix() and camera.projectionMatrix()) for the interactive user-adaptive methods in our system. The client system consists of three parts: data receiving, decoding, and rendering, where decoding is further divided into the CPU and GPU decoder. The data receiver and CPU decoder are implemented in C/C++ for faster performance. For color bytes decoding, we cross-compile libjpeg-turbo [5] for iOS and use the C/C++ API rather than using the JPEG decoding API on iOS to integrate the CPU into a single

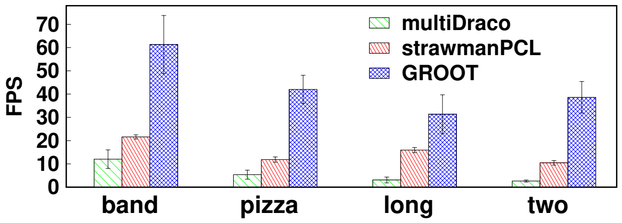


Figure 11: Overall frame update rate.

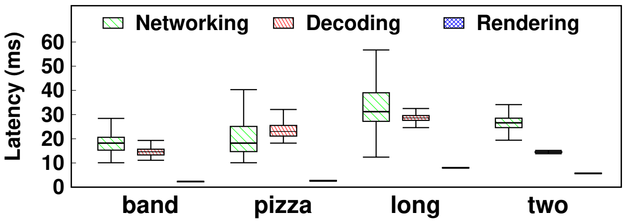


Figure 12: End-to-end latency breakdown of GROOT.

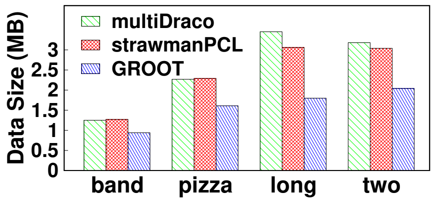


Figure 13: Overall compression performance of PD-Tree.

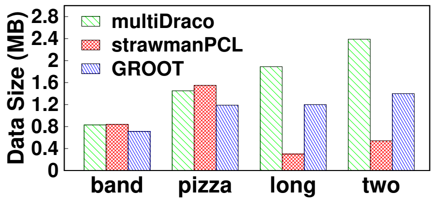


Figure 14: Average data size of compressed 3D geometry.

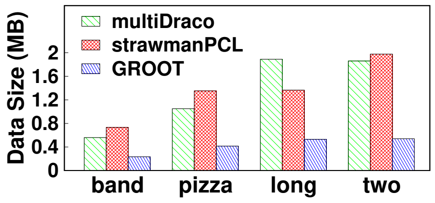


Figure 15: Average data size of compressed color attribute.

module and for faster performance. We implement the GPU decoder and renderer using Apple's Metal Framework as a single GPU kernel function.

Server. We implement the GROOT server on a Linux desktop PC (Ubuntu 18.04) with all functions written in C/C++. The encoder can be divided into three modules: PD-Tree generator, OBB and ODB compressor, and OCB compressor. For the implementation of the PD-Tree generator, we modify the octree generator from Point Cloud Library (PCL). OBB and ODB compression and OCB packing is implemented with C/C++, and we use libjpeg-turbo [5] for JPEG compression. The runtime server implementation reads the encoded files without JPEG compression to apply interactive user-view adaptation directly without decoding the entire file. Our continuous and interactive user-adaptive methods, frustum culling, and depth-based sampling are also implemented with C/C++ functions.

8 Evaluation

8.1 Experimental Setup

Dataset. We conducted extensive evaluation using the datasets in Table 1. Commonly in AR applications, the virtual objects are anchored on a plane through a plane detection algorithms [41]. For the consistency of the experiment, we place the volumetric video contents by anchoring the center of each data on a fixed location.

System. We test the GROOT client on iPhone XS running on iOS 13.1.3, equipped with the Apple A12 Bionic chipset, Hexa-core CPU (2×2.5 GHz Vortex + 4×1.6 GHz Tempest), and a 4-core GPU with 4GB RAM. The GROOT server is tested on a desktop PC running on Ubuntu 18.04 equipped with Intel Core i7-8700 3.2 GHz CPU. The client communicates with the server over TCP through a commodity 802.11ac Wi-Fi AP at 5 GHz. The AP is connected to the server by a 1 Gbps Ethernet cable. The average downlink throughput is 540 Mbps, which is comparable to the practical performance of current indoor wireless technologies, as reported in [32].

Baselines. We compare our system to following baseline systems:
1) *multiDraco*: an optimized version of Draco [2] by applying multithreading as in ViVo [14]. The maximum number of threads is fixed as 4 since it showed the best performance.
2) *strawmanPCL*: an optimized version of PCL [8]. The core functions (i.e., octree-based geometry encoding/decoding and entropy encoding/decoding for colors) are implemented using SIMD instructions for fast vector calculations. For entropy encoding, we use the Zstandard [11] library, a state-of-the-art data compression technique, since it can run in real-time on mobile devices while maintaining the compression rate.

8.2 Overall Performance

We first assess the overall performance of GROOT. Figure 11 shows the overall frame update rate. The frame update rate is measured when the next frame is received, decoded, and enqueued into the frame buffer to be rendered. Due to the sparsity of point clouds, the system performance can vary depending on the user's movements. For example, in the scene of Figure 1(a), if the user focuses on a single person, the data size can be reduced by a maximum of 3×. Therefore, we test our system on multiple user view traces, which cover diverse direction and locations and show the averaged results. Since user-view prediction is not in the scope of this work, we follow the results in [14] and assume that it is possible to maintain an accurate prediction within the window of 200ms (6 to 7 consecutive frames). GROOT enables a 30 fps or higher frame update rate for all datasets showing significant improvement from conventional systems. Other methods could not achieve the real-time frame update rate due to the complexity in decoding. Furthermore, the latency breakdown results of GROOT in Figure 12 show that the decoding latency is well-below 30ms making decoding on mobile devices no longer the bottleneck for volumetric video streaming. Also, the rendering latency remains stable as below 10ms to meet the motion-to-photon latency requirement.

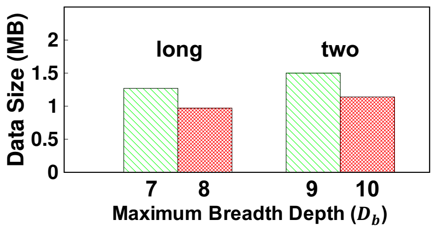


Figure 16: Data size of geometry for different Maximum Breadth Depth (D_b) for 8i datasets.

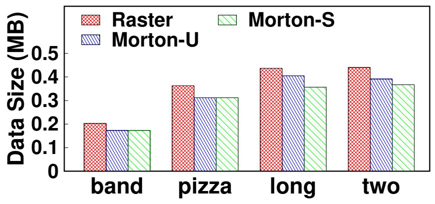


Figure 17: Data size comparison of color packing methods.

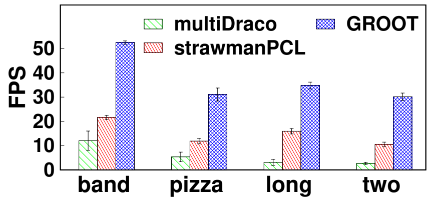


Figure 18: Average decoding frame rate of GROOT without user view adaptation.

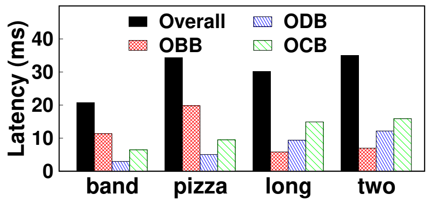


Figure 19: Decoding latency breakdown.

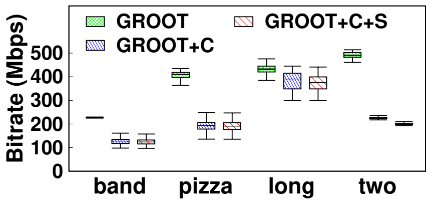


Figure 20: Average bitrate with interactive user adaptation.

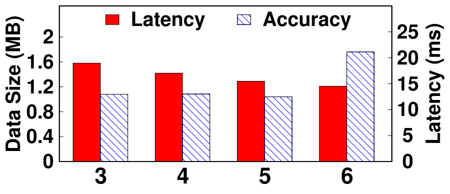


Figure 21: Performance trade-off for frustum culling maximum depth.

8.3 Breakdown of PD-Tree-based Compression

In this section, we evaluate each component of our PD-Tree-based compression scheme. The overall compression performance of PD-Tree is shown in Figure 13. Even without user view adaptation, the data size is reduced by 1.7×.

Geometry Compression. Figure 14 shows the average frame size for the lossless encoding of the geometry. For Panoptic datasets (*band* and *pizza*), the compressed data size of GROOT is smaller than both multiDraco and strawmanPCL. However, the compressed geometry of the 8i dataset (*longdress* and *twopeople*) exhibits a larger size than strawmanPCL. This is due to the different characteristics of the dataset. Panoptic datasets are larger in scale, which means deeper octree depth, but has less number of points than 8i datasets. Therefore, the individual representation of points incurs a smaller overhead. On the other hand, as shown in Figure 5, the 8i dataset has a sharper curve than the Panoptic datasets. This means that less number of higher depths in the 8i dataset is the bottleneck compared to the Panoptic dataset. Therefore, it would be more effective to increase the maximum breadth depth (D_b) for the 8i dataset and reduce the overhead of the ODB stream. Figure 16 shows the reduced data size when the maximum breadth depth is increased by 1 for the 8i datasets. Specifically, D_b for the longdress dataset is 7 to encode the last 3 depths as ODB and 8 to encode the last 2 depths as ODB. In the case of the twopeople dataset, D_b is 9 and 10 to make the last 3 and 2 bytes as ODB, respectively.

Despite the variability in the dataset, GROOT still achieves the best compression ratio when combined with the color compression, and more importantly, GROOT enables faster decoding and runtime modification.

Color Compression. We now evaluate the benefits of our color compression method. Overall, Figure 15 shows that our proposed method can improve the compression rate by 4.3× to 6.5× compared to existing methods while maintaining the perceptual quality (See Section 8.6 for the quality evaluation). Results show that the color compression ratio is higher in 8i datasets. This is because Panoptic datasets are lower in resolution compared to 8i datasets and show less variation in color. Next, we show the effect of the individual components of our method. In Figure 17, *Morton-S* is the scheme used in GROOT, which packs the colors into a 2D image following Morton ordering and sorts the color to maximize the similarity between adjacent points. *Morton-U* is the same as *Morton-S* except no sorting is applied, and *Raster* uses serial packing of the colors without sorting. It shows that Morton ordering can reduce the data size by an average of 50 kB and another 10 kB to 50 kB when sorting is applied. In particular, sorting the colors to maximize the locality has more effect on the 8i datasets, since they have more color variations than the Panoptic datasets.

8.4 Performance of Parallel Decoder

Breakdown. Figure 19 shows the decoding latency of each component in the decoding pipeline. In general, latency for decoding ODB and OCB increases linearly with the number of points. However, the overall decoding latency is not linearly correlated with the number of points. For sparse and large-scale datasets such as *band* and *pizza*, the decoding latency resides mainly in decoding the OBB. Since the points are sparsely located, each of the last few depths of the octree has a number of octree nodes similar to the number of points. For high-resolution datasets such as *longdress* and *twopeople*, only the last octree depth incurs a non-negligible decoding latency. In both cases, latency at the last octree depth can be effectively eliminated to meet the 30 fps frame rate with the PD-Tree-based decoding.

Resource Utilization. We analyze the resource utilization when the decoding, rendering, and camera capturing are simultaneously running as described in Section 7. On a fully charged phone, we play the *longdress* video repeatedly, which required the most computation, for 54,000 frames (30 minutes video when the frame update rate is 30 fps). There was an average memory usage of 340 MB, 25%

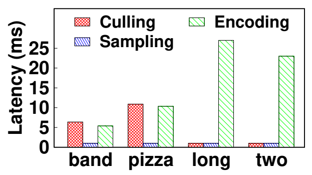


Figure 22: Server processing overhead for interactive user adaptation.

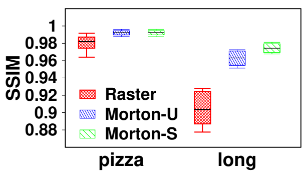


Figure 23: SSIM value comparison of color packing methods.

CPU utilization, and the battery level dropped to 86%. The energy consumption was partly because of the rendering and the background jobs (i.e., camera position tracking, camera frame capturing) rather than just our parallel decoder. When we only ran the two components without our decoder in the same setting, the battery level still dropped by 7%.

8.5 Performance of Interactive User Adaptation

Comparison with fixed-size blocks for frustum culling. We compare our adaptive frustum culling technique to fixed grid size methods used in most existing methods such as ViVo [14]. Figure 21 shows the results of the *pizza* dataset, which is the largest in scale and consists of the most number of people and objects. In general, using a smaller grid size can more effectively remove unnecessary points resulting in smaller data size. However, when the frustum depth increases from 5 to 6, the average frame size reduces by only 80 kB, while the latency increases by 7ms. The results show that the design choice of GROOT detailed in Section 6.1 can effectively reduce the data size with small grid sizes while maintaining a low processing overhead at the server.

Server-side processing overhead. We measure the overall processing overhead at the server in terms of latency to apply interactive user adaptation, which includes frustum culling, sampling, and encoding of each frame, in runtime. Figure 22 shows the latency for each component. The results show that even without pipelining the steps, runtime adaptation is possible within 30 fps. For large scale datasets such as *band* and *pizza*, the frustum culling latency is dominant since it is larger in scale with multiple people and objects. In *longdress* and *twopeople* datasets, the encoding latency is dominant since it has more number of points resulting in a bigger color image to compress. We note that the encoding latency can be further improved by using dedicated hardware for JPEG compression. In conclusion, since the user-adaptive methods can be applied in runtime with minimal overhead, generating and storing pre-encoded files for fixed-size grids as in [14, 43] or in conventional video streaming systems is not necessary.

8.6 Perceptual Quality of Color Compression

Color Comparison to baselines. We measure the PSNR per YUV component when color attributes are encoded with the same bit rate to compare the performance of OCB compression to the baseline methods. We calculate the metric based on the definition in the released MPEG standard [28] and the implementation of [47]. Results showed that GROOT achieves a higher PSNR value of 36.4dB, 45.6dB, and 43.5dB for each YUV component while Google Draco and PCL achieved 29.7dB, 39.4dB, and 38.1dB on average.

Table 3: SSIM values of GROOT with interactive user adaptation.

Dataset	band	pizza	longdress	twopeople
SSIM	0.9852 ±0.0043	0.9880 ± 0.0064	0.9898 ±0.011	0.9767 ±0.0037

Effectiveness of color reordering and Morton coding. Next, we use the SSIM metric [59] to measure the perceptual quality of the 2D projected and rendered view of the 3D content on a mobile device to evaluate the effect of color reordering and Morton coding. In Figure 23, each *Morton-S*, *Morton-U*, and *Raster* are the same as indicated in Section 8.3 for Figure 17. The results show that the SSIM value can be significantly improved with Morton ordering and the color sorting also improves the perceptual quality. Note that results for *band* and *twopeople* were omitted since they had similar results to *pizza* and *longdress*, respectively.

The reason behind the improvement of the perceptual quality with Morton ordering is because 2D image compression applies frequency-domain color quantization on pixel blocks (i.e., 8×8 pixels). This results in a blur effect on the final image. When packing the color values pixel by pixel, rather than as a texture, the blurring effect mixes the color between adjacent pixels in a block, which may have very different colors. Following the Morton ordering, the distortion can be minimized by maximizing the probability of adjacent points in a block to have similar colors. For the same reason, sorting the colors to further maximize the locality of similar colors helps improve the quality.

8.7 Perceptual Quality of Interactive User Adaptation

We evaluate the impact of our interactive user adaptation methods, frustum culling and depth-based sampling. We play the videos on a smartphone and capture the rendered screen to measure the SSIM value. Results in Table 3 show that there is a minimal impact on the perceptual quality maintaining a high SSIM value of over 0.98 for all datasets. The frustum culling method does not affect the perceptual quality at all since it only removes the points that are not rendered on the screen. As shown in Figure 20, depth-based sampling has a less impact on the bitrate compared to frustum culling. This is because sampling is not applied as aggressively to maintain the SSIM value. Depending on the requirement of the user, it is possible to apply further sampling for acceptable quality.

9 Discussion

Interframe Compression. The current version of GROOT does not include inter-frame compression. Inter-frame compression is challenging in 3D volumetric videos since there is no adjacency information between the points. Thus, they should be processed individually. Some prior work modified the octree structure to reuse the serialized byte stream of previous frames [30]. Another approach in [37] extracts a rigid transformation between two subsets of points of consecutive frames using the Iterative Closest Point (ICP) algorithm [58]. Even though it can reduce the data size, the complexity of the decoder increases. Utilizing the independent representation of GROOT, we plan to develop inter-frame compression by applying existing solutions and adopting recent studies in deep learning, which can generate 3D motion vectors for individual points [33].

Quality Metric. We have used the most common quality metrics of 3D point clouds for the quality assessments. However, most of the quality metrics are simply an adaptation of the existing 2D video quality metrics, which does not explain the unique characteristics of 3D data. For example, there are various ways of rendering point clouds. In this work, we used a fixed-size square to represent individual points. However, using surface splatting [60] or adaptive point sizes can further reduce the number of necessary points while maintaining the user perceptual quality. Furthermore, the distance of the 3D data from the user and the scale can also affect the perceptual quality. Therefore, it would be necessary to develop a new subjective and objective quality metric specifically for point clouds taking account of the various rendering options and the dynamic user movements.

Video Content Adaptation. On top of our depth-based sampling scheme, more sophisticated techniques can be applied. A popular approach in processing volumetric videos in the graphics field is to focus on the face and the hands since they have a higher impact on the visual perception quality [44]. Adopting this concept to volumetric video streaming systems would be able to further reduce the bandwidth consumption without sacrificing the perceptive visual quality. We can also consider applying different sampling rate depending on the movement of the content or the luminance changes as in [25]. It will be possible to directly apply any optimization schemes to GROOT, since the PD-Tree structure allows the individual points to be removed, reordered, or manipulated without having to modify the decoder.

Head-mounted Displays (HMDs). In this work, we used smartphones with 2D displays as primary client devices. In order to provide the users with a more immersive experience, using head-mounted displays such as Oculus VR, Microsoft Hololens or Magic Leap can be more plausible. However, current commercially available devices still suffer from limited field-of-view and resolution, which makes it difficult to achieve the same perceptual quality as smartphone displays. Since there are active research and product development on next-generation devices, we plan to further explore volumetric video streaming for HMDs. Furthermore, reducing resource utilization is much more important since HMDs confront tighter power and thermal constraints.

10 Related Work

Mobile 2D and 360° Video Streaming. There have been prior works that aim at streaming 2D or 360° videos on mobile devices. To stream high-resolution videos under limited network bandwidth, existing approaches dynamically adapt the video bitrate [31, 51], or divide the video into spatial blocks (tiles) and selectively stream the ones included in the user's field-of-view [15, 26, 34, 40, 46]. However, the existing techniques cannot be directly applied to volumetric videos since 3D data is inherently different from 2D videos. Unlike 2D videos, where the consecutive frames are the same size and thus can be divided into equal-sized tiles, it is difficult to divide the 3D data into equal-sized blocks since the number of points and the location of the occupied grid differs every frame.

Volumetric Video Compression. A large number of approaches have been made to compress the data size of volumetric videos [19, 20, 23, 53, 57]. Most work focuses on the compression rate rather than enabling real-time decoding, especially on resource-constrained mobile devices. There are also active standardization movements by MPEG [28, 50]. Since the finalization of the standards is expected in the near future, it would also be an interesting research direction to apply our user-adaptive methods. Currently, as far as our knowledge, there are a limited number of studies in the system design for real-time volumetric video streaming of generic point cloud videos for mobile devices. Since the latest commercial wireless technologies such as 802.11ad and 5G exhibit compelling performances, it is becoming more important to balance the trade-off between the network latency and the resource utilization on mobile devices for decoding.

Volumetric Video Streaming. There are some recent attempts in volumetric video streaming. Nebula [45] proposed the concept of an edge-assisted volumetric video system where the edge server decodes the 3D data and generates a 2D video for the mobile device. More recently, ViVo [14] uses Google Draco as the underlying codec and applies visibility-aware optimizations to enable real-time streaming. It well supports small-sized videos but is not easy to scale due to the inherent limitations of Draco as described in Section 3.3. [43] also applies user-adaptive optimizations similar to ViVo and [54] proposes a DASH-compliant volumetric video streaming system. GROOT is a novel system that enables real-time volumetric video streaming on mobile devices, especially for the large-size high-fidelity videos with continuous and interactive user-view adaptation tightly coupled with the compression scheme.

11 Conclusion

We presented the design and implementation of GROOT, a mobile volumetric video streaming system. With the novel *PD-Tree* data structure, GROOT enabled real-time streaming at a 30 fps frame rate with minimal memory usage and computation for decoding, which was the bottleneck of the streaming pipeline. Specifically, we modified the conventional octree to introduce the independence of leaf nodes, which enabled the parallel decoding of highly irregular 3D data. Leveraging the unique advantages of our PD-Tree, we also developed a suite of techniques to compress color information and filter out and sample 3D points outside of a user's view, which could further minimize the data size and the decoding cost. With the proposed techniques, GROOT achieved faster frame rates in streaming high-fidelity volumetric videos to mobile devices compared to any previous methods.

Acknowledgments

We sincerely thank our anonymous shepherd and reviewers for their valuable comments. This work was supported by the New Faculty Startup Fund from Seoul National University and the National Research Foundation of Korea (NRF) grant (No. 2019R1C1C1006088). Youngki Lee and Youngmin Kim are the corresponding authors of this work.

References

[1] ARM Neon. https://www.arm.com/why-arm/technologies/neon.
[2] Google Draco. https://google.github.io/draco/.
[3] Intel LibRealSense. https://github.com/IntelRealSense/librealsense.
[4] iOS SIMD. https://developer.apple.com/documentation/accelerate/simd.
[5] libjpeg-turbo. https://libjpeg-turbo.org.
[6] LiveScan3D. https://github.com/MarekKowalski/LiveScan3D.
[7] Point cloud compression. https://mpeg.chiariglione.org/standards/mpeg-i/point-cloud-compression.
[8] Point Cloud Library (PCL). https://github.com/PointCloudLibrary/pcl.
[9] Volumetric video market by volumetric capture (hardware (camera processing unit), software, and services), application (sports, events and entertainment, medical, signage, education training), content delivery and region - global forecast to 2025. https://www.marketsandmarkets.com/Market-Reports/volumetric-video-market-259585041.html.
[10] What a 5g world could look like: 3d holograms, faster ai and new security concerns. https://www.cbsnews.com/news/what-a-5g-world-could-look-like-3d-\holograms-ai-new-security-concerns/.
[11] Zstandard. http://www.zstd.net/.
[12] S. Aggarwal, A. Thirumurugan, and D. Koutsonikolas. A first look at 802.11ad performance on a smartphone. In *Proceedings of the 3rd ACM Workshop on Millimeter-Wave Networks and Sensing Systems*, mmNets'19, pages 13–18, New York, NY, USA, 2019.
[13] U. Assarsson and T. Moller. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools*, 5(1):9–22, 2000.
[14] Y. L. B. Han and F. Qian. Vivo: Visibility-aware mobile volumetric video streaming. In *In The 26th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2020.
[15] G. Baig, J. He, M. A. Qureshi, L. Qiu, G. Chen, P. Chen, and Y. Hu. Jigsaw: Robust live 4K video streaming. In *Proceedings of the 25th Annual International Conference on Mobile Computing and Networking*, 2019.
[16] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.
[17] J. Bittner, V. Havran, and P. Slavik. Hierarchical visibility culling with occlusion trees. In *Proceedings. Computer Graphics International (Cat. No.98EX149)*, pages 207–219, 1998.
[18] E. Cuervo, K. Chintalapudi, and M. Kotaru. Creating the perfect illusion: What will it take to create life-like virtual reality headsets? In *Proceedings of the 19th International Workshop on Mobile Computing Systems Applications*, HotMobile'18, pages 7–12, New York, NY, USA, 2018. Association for Computing Machinery.
[19] R. L. de Queiroz and P. A. Chou. Compression of 3d point clouds using a region-adaptive hierarchical transform. *IEEE Transactions on Image Processing*, 25(8):3947–3956, 2016.
[20] R. L. de Queiroz and P. A. Chou. Transform coding for point clouds using a gaussian process model. *IEEE Transactions on Image Processing*, 26(7):3507–3517, July 2017.
[21] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou. 8i voxelized full bodies - a voxelized point cloud dataset. ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006, January 2017.
[22] D. C. Garcia and R. L. de Queiroz. Intra-frame context-based octree coding for point-cloud geometry. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, 2018.
[23] D. C. Garcia and R. L. de Queiroz. Intra-frame context-based octree coding for point-cloud geometry. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 1807–1811, 2018.
[24] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai. An overview of ongoing point cloud compression standardization activities: video-based (v-pcc) and geometry-based (g-pcc). *APSIPA Transactions on Signal and Information Processing*, 9:e13, 2020.
[25] Y. Guan, C. Zheng, X. Zhang, Z. Guo, and J. Jiang. Pano: Optimizing 360° video streaming with a better understanding of quality perception. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM'19, pages 394–407, New York, NY, USA, 2019. Association for Computing Machinery.
[26] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han. Rubiks: Practical 360-degree streaming for smartphones. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, 2018.
[27] M. Hosseini and C. Timmerer. Dynamic adaptive point cloud streaming. In *Proceedings of the 23rd Packet Video Workshop*, 2018.
[28] E. S. Jang, M. Preda, K. Mammou, A. M. Tourapis, J. Kim, D. B. Graziosi, S. Rhyu, and M. Budagavi. Video-based point-cloud-compression standard in mpeg: From evidence collection to committee draft [standards in a nutshell]. *IEEE Signal Processing Magazine*, 36(3):118–123, 2019.
[29] H. Joo, T. Simon, X. Li, H. Liu, L. Tan, L. Gui, S. Banerjee, T. S. Godisart, B. Nabbe, I. Matthews, T. Kanade, S. Nobuhara, and Y. Sheikh. Panoptic studio: A massively multiview system for social interaction capture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.
[30] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach. Real-time compression of point cloud streams. In *2012 IEEE International Conference on Robotics and Automation*, 2012.
[31] J. Koo, J. Yi, J. Kim, M. A. Hoque, and S. Choi. REQUEST: Seamless dynamic adaptive streaming over http for multi-homed smartphone under resource constraints. In *Proceedings of the 25th ACM international conference on Multimedia*, 2017.
[32] Z. Lai, Y. C.Y. Hu, Y. Cui, and N. D. L. Sun. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, 2017.
[33] X. Liu, C. R. Qi, and L. J. Guibas. Flownet3d: Learning scene flow in 3d point clouds. *CVPR*, 2019.
[34] A. Mahzari, A. Taghavi Nasrabadi, A. Samiei, and R. Prakash. Fov-aware edge caching for adaptive 360 video streaming. In *2018 ACM Multimedia Conference on Multimedia Conference*, 2018.
[35] D. Meagher. Geometric modeling using octree encoding. In *Computer Graphics and Image Processing*, 1982.
[36] R. Mekuria, K. Blom, and P. Cesar. Design, implementation, and evaluation of a point cloud codec for tele-immersive video. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(4):828–842, 2017.
[37] R. Mekuria, K. Blom, and P. Cesar. Design, implementation, and evaluation of a point cloud codec for tele-immersive video. *IEEE Transactions on Circuits and Systems for Video Technology*, 2017.
[38] N. Memon, D. L. Neuhoff, and S. Shende. An analysis of some common scanning techniques for lossless image coding. *IEEE Transactions on Image Processing*, 9(11):1837–1848, 2000.
[39] A. Narayanan, E. Ramadan, J. Carpenter, Q. Liu, Y. Liu, F. Qian, and Z.-L. Zhang. A first look at commercial 5g performance on smartphones. In *Proceedings of The Web Conference 2020*, WWW'20, pages 894–905, New York, NY, USA, 2020. Association for Computing Machinery.
[40] A. Nguyen, Z. Yan, and K. Nahrstedt. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *2018 ACM Multimedia Conference on Multimedia Conference*, 2018.
[41] P. Nowacki and M. Woda. Capabilities of arcore and arkit platforms for ar/vr applications. In *Engineering in Dependability of Computer Systems and Networks*, pages 358–370, Cham, 2020. Springer International Publishing.
[42] J. Park, P. A. Chou, and J. Hwang. Volumetric media streaming for augmented reality. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2018.
[43] J. Park, P. A. Chou, and J. Hwang. Rate-utility optimized streaming of volumetric media for augmented reality. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):149–162, 2019.
[44] G. Pavlakos, V. Choutas, N. Ghorbani, T. Bolkart, A. A. A. Osman, D. Tzionas, and M. J. Black. Expressive body capture: 3d hands, face, and body from a single image. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.
[45] F. Qian, B. Han, J. Pair, and V. Gopalakrishnan. Toward practical volumetric video streaming on commodity smartphones. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, HotMobile '19, 2019.
[46] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018.
[47] M. Quach, G. Valenzise, and F. Dufaux. Improved Deep Point Cloud Geometry Compression. *arXiv e-prints*, page arXiv:2006.09043, June 2020.
[48] R. J. S. Shi, V. Gupta. Freedom: Fast recovery enhanced VR delivery over mobile networks. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, 2019.
[49] S. Schwarz and M. Personen. Real-time decoding and ar playback of the emerging mpeg video-based point cloud compression standard. Technical report, IBC, 2019.
[50] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, J. L. Z. Li, K. Mammou, R. Mekuria, O. Nakagami, E. Siahaan, A. Tabatabai, A. M. Tourapis, and V. Zakharchenko. Emerging mpeg standards for point cloud compression. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.
[51] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, 2016.
[52] S. Subramanyam, J. Li, I. Viola, and P. Cesar. Comparing the quality of highly realistic digital humans in 3dof and 6dof: A volumetric video case study. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 127–136, 2020.
[53] D. Thanou, P. A. Chou, and P. Frossard. Graph-based compression of dynamic 3d point cloud sequences. *IEEE Transactions on Image Processing*, 25(4):1765–1778, 2016.
[54] J. van der Hooft, T. Wauters, F. De Turck, C. Timmerer, and H. Hellwagner. Towards 6DoF http adaptive streaming through point cloud compression. In *Proceedings of the 27th ACM International Conference on Multimedia*, 2019.

[55] D. W Walker. Morton ordering of 2d arrays for efficient access to hierarchical memory. *The International Journal of High Performance Computing Applications*, 2018.
[56] C.-C. J. K. Y. Huang, J. Peng and M. Gopi. Octree-based progressive geometry coding of point clouds. In *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, 2006.
[57] K. Zhang, W. Zhu, and Y. Xu. Hierarchical segmentation based point cloud attribute compression. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3131–3135, 2018.
[58] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *Int. J. Comput. Vision*, 1994.
[59] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.
[60] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 371–378, New York, NY, USA, 2001. Association for Computing Machinery.