M.S. THESIS

Volumetric Video Delivery on Mobile Devices

모바일 볼류메트릭 비디오 스트리밍 시스템

BY

LEE KYUNGJIN

FEBRUARY 2020

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Abstract

A volumetric video consists of three-dimensional data, thus providing the user with six degrees of freedom (6 DoF) to fully immerse himself into the media. Combined with the recent advancement in virtual reality (VR), augmented reality (AR), and mixed reality (MR), volumetric video can bring out numerous applications, such as teleconferencing, remote collaboration, and mixed reality, and can be applied to different fields including medical, architecture, education, and arts. Furthermore, the newly released evolution of the wireless network, 5G, and state-of-the-art mobile devices equipped with multi-core CPUs and GPUs open up the possibility of delivering these contents to mobile devices so that the users can interact with the media anytime and anywhere.

Despite the promising future of streaming volumetric videos to mobile devices, there exist multiple challenges. First, the data size of volumetric videos is much bigger than high-resolution 2D or 360°videos. However, volumetric video compression is not optimized yet for real-time systems and there is no dedicated hardware for the purpose. Lastly, adopting approaches from streaming high-resolution video or 360°videos, such as Adaptive Bit Rate(ABR) algorithms or user view adaptive methods have not been studied extensively. Motivated by the aforementioned challenges, we aim to develop the first volumetric video streaming system that enables real-time streaming. We propose novel techniques that can reduce the data size while maintaining user perception quality. We optimize every step of our system to implement a prototype and conduct experiments on the state-of-the-art datasets.

keywords: Volumetric video, interactive media, point cloud, heterogenous computing

student number: 2018-22221

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Volumetric video is an emerging type of media that can provide a new level of immersive and interactive user experience. It consists of 3D data that can be viewed from any angle, thus providing the users with six-degrees-of-freedom (6DoF). Compared to 2D videos and 360 videos, volumetric video allows the user to become a part of the media rather than simply observing them. Volumetric video not only brings out a variety of new applications, such as teleconferencing, mixed reality, and remote collaboration, but it can also be applied to many different fields, including architecture and construction, medical, education, and arts. Volumetric video market growth is projected to reach $2.8 billion by 2023 [1] and companies such as Microsoft, Intel, Facebook, and 8i are actively researching on capturing high-quality volumetric videos. Furthermore, major wireless network companies such as Verizon and AT&T consider volumetric videos as the ”killer application” of their 5G network [2].

Delivering a stream of 3D data to mobile devices is important since viewing these contents on desktop computers or wired head mounted displays will restrict the user's movement, thus limiting the experience of 6 DoF. Recently, major manufacturing companies for mobile devices, such as Samsung, Apple, Google, and Huawei, revealed that their next-generation devices will come with a depth sensor [3]. This will allow mobile devices to track the 3D space as well with high accuracy. When user interaction is

integrated seamlessly to the streamed volumetric video, it will be possible to ultimately break the boundaries between reality and the virtual world.

In this paper, we propose a system that enables streaming volumetric videos, in particular, point cloud videos, to mobile devices. We first propose a file format to store compressed point cloud video at the server which will be streamed to mobile clients. During the online stage, the server reads the encoded frames and extract spatial information of the point cloud data in real-time to apply perception adaptive methods to reduce the data size. Finally, when the encoded point cloud arrives at the client mobile device over the wireless network, it is efficiently decoded on the client device to regenerate raw point cloud data for rendering. We identify that it is possible to leverage how the user perceives a 2D projected view of the 3D model at a single timestamp to generate and send a 2D video stream tracking the user's viewpoint as in [4]. However, for the users to be able to perceive a continuous view of the 3D model, a projected view with an angular resolution of approximately 0.3 degrees should be provided [5]. The difficulty in predicting 6 DoF and/or fast user movements results in unnecessary bandwidth consumption and degradation in user perception quality. Therefore, we direct our focus on delivering 3D data itself for high accuracy and seamless integration with the perspective of the real world.

Despite the simplicity of point cloud representations, it generates multiple challenges when it becomes a continuous stream. Unlike 2D images, it is difficult to store a point cloud frame in a fixed 3D array of pixels (more precisely volume pixels, or voxels) since lots of pixels correspond to empty space, making the frame size unnecessarily large (e.g., $1Gbps \approx 1024 \times 1024 \times 1024$, 10-bit voxel grid). Therefore, the coordinates of each point are saved individually but still comprise relatively large data size (i.e. Gigabytes per second). It requires compression techniques to reduce the size of volumetric videos as in conventional videos. There has been active research on point cloud compression along with standardization activities by MPEG. However, most of the solutions are not applicable to real-time systems, especially in resource-

constrained mobile devices. Achieving intra-frame compression is challenging since there is no spatial information or order between the points. Also, designing an inter-frame compression is difficult due to the fact that the number of points can vary for each frame and there is no correspondence between points in successive frames. Furthermore, decoding compressed point cloud videos is slow on resource-limited mobile devices since there are no existing hardware acceleration techniques. Lastly, adopting conventional user perception adaptive approaches in video streaming [6, 7] has not been properly investigated to leverage the unique characteristics of 3D data.

To address the aforementioned challenges, we design a user perception adaptive volumetric video streaming system that delivers 3D data to mobile devices in real-time. We first design an effective point cloud compression scheme that can both minimize the resource consumption of wireless network bandwidth and mobile computation. Then, we develop user view adaptive algorithms that allow the system to run in real-time. We aim to optimize the end-to-end system starting from encoding raw point cloud video, sending them through the wireless network, to decoding and rendering them on client devices.

The major contributions of this work is as follows:

- To the best of our knowledge, this is the first end-to-end mobile volumetric video streaming system which delivers 3D data to open up new applications. We first identify the challenges and obstacles that need to be tackled in designing a volumetric video streaming system. We propose how to adopt conventional solutions from video streaming systems taking account the unique characteristics of volumetric video.
- We propose a real-time point cloud sampling scheme that adapts to the current user's viewpoint to reduce the data size but maintain the user experience quality.
- We design a real-time point cloud decoding technique using heterogenous computing power on mobile devices.
- We implement a prototype of our proposed system and conduct experiments with

the available state-of-the-art datasets.

Chapter 2

Background

2.1 Capturing Volumetric Video

Volumetric videos can be captured by using multiple RGB-D cameras (e.g., Intel RealSense, Microsoft Kinect) to acquire depth images from multiple viewing angles and synchronizing them. Several open-source volumetric videos captured from studio settings are available [8,9] (examples shown in Figure 2.1), as well as open-source capture libraries for general developers [10, 11].

2.2 Representations of Volumetric Video.

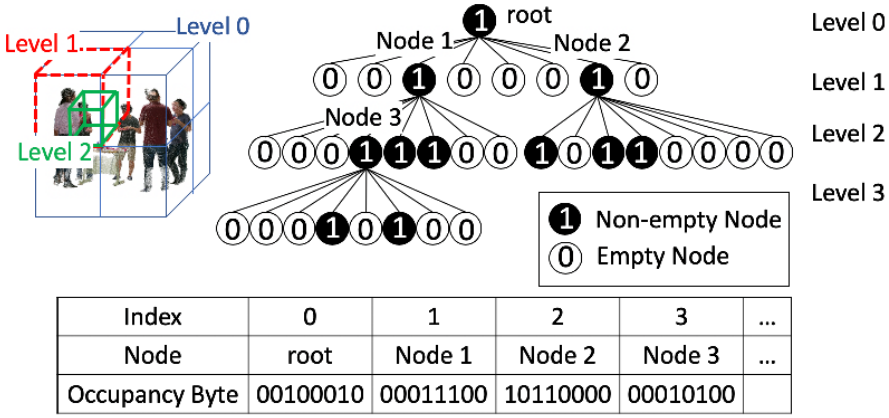
As raw data, volumetric videos are represented as point clouds, where each point is represented by a 3D coordinate (X, Y, Z, 4 bytes each) and its corresponding attributes such as color value (R, G, B, 1 byte each) or normals. Point clouds can be rendered as individual points or reconstructed as 3D meshes or voxels (volume pixels). To efficiently process point clouds, data structures such as octree [12] or k-d tree [13] are commonly used. In octree, the 3D coordinates of the point cloud are recursively encoded as a tree composed of a parent node and 8 child nodes (which correspond to 8 sub-spaces), represented by an 8-bit occupancy map as shown in Figure 2.2. This ap-

(a) 170915_office1 video from CMU Panoptic Dataset [8].

(b) 160906_pizza1 video from CMU Panoptic Dataset [8].

(c) Longdress video from 8i Voxelized Full Bodies [9].

Figure 2.1: Sample volumetric videos.



Index	0	1	2	3	...
Node	root	Node 1	Node 2	Node 3	...
Occupancy Byte	00100010	00011100	10110000	00010100	

Figure 2.2: Octree data structure.

proach utilizes the spatial relationship between adjacent points to store 3D coordinates with smaller memory (i.e., each child node is represented as 1-bit occupancy instead of 12 bytes XYZ coordinates). kd-tree is similar except that it is a binary tree which cycles between the axes and makes half-spaces.

2.3 Existing Libraries

Several open-source libraries have been released for processing volumetric videos, including Point Cloud Library (PCL) [14], Google Draco [15], and Intel Open3D [16].

However, they are mostly optimized for rendering purposes in desktop PCs and are ill-suited for streaming purposes in mobile devices. Most importantly, they only provide single thread CPU implementation, which incurs significant processing latency as will be shown in Section 3.

Finally, with the increasing interest in point cloud volumetric video, there have been movements in the standardization of compressing point cloud video by MPEG V-PCC [17]. It proposed to use 2D projected patches of the 3D point cloud model to utilize the existing video codecs. Due to the information loss after the projection process, the reconstructed point cloud degrades in quality which can affect user perception quality. Performance analysis of the current test model of MPEG V-PCC is handled in Section 3.2. Furthermore, there is no open-source library that can be directly adopted into other systems.

Chapter 3

Motivational Studies

To motivate our system, we first investigate why naive approaches fail to achieve real-time streaming performance. Furthermore, we also present why it is non-trivial to apply existing techniques utilized for high-resolution 2D and 360° to necessitate a need for a specialized design for 3D volumetric videos.

3.1 Raw Data Streaming

The most trivial way to stream volumetric video is to send raw point cloud data. However, this is often infeasible due to the large data size that needs to be sent over a bandwidth-limited wireless network. Specifically, with each point represented as 15 bytes, the data size of a raw point cloud video can be extremely large. For instance, Table 3.1 shows that it requires Gbps of network bandwidth for streaming typical 30 fps volumetric videos, whereas conventional 802.11ac Wi-Fi speed is only 400 Mbps [18].

Table 3.1: List of datasets.

Name	Avg # of Points	Required Bandwidth (Gbps)	Avg Rendering FPS
office	223k	0.80	555
pizza1	344k	1.23	751
pizza2	689k	2.48	262

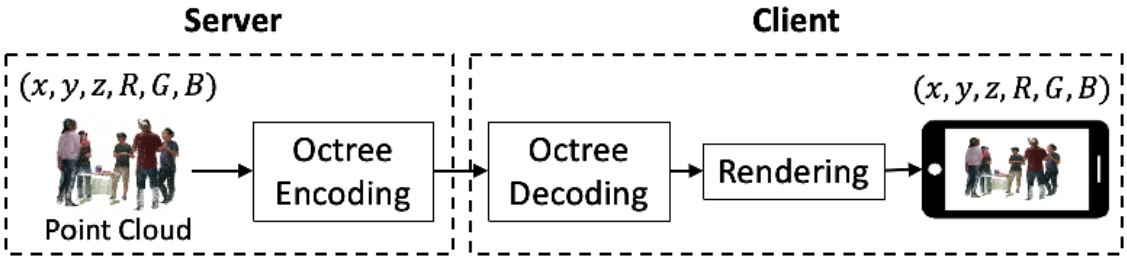


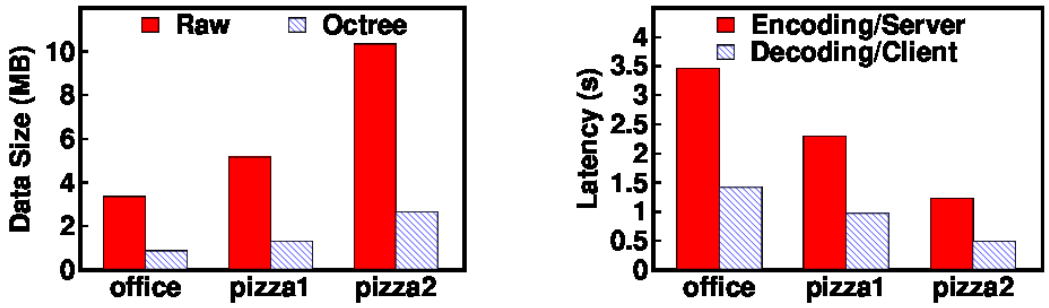
Figure 3.1: Octree-encoded baseline streaming pipeline.

3.2 2D Projection-based Streaming

It is adopted by MPEG V-PCC to support real-time playback on mobile devices. The key idea is to decompose the point cloud into multiple patches based on their surface information and project them onto a single 2D image. Then, the 2D image is sent to the mobile client where the corresponding 3D point cloud is reconstructed with auxiliary metadata such as the projection plane, 3D location, and 2D image bounding box for each patch [19]. This approach is intuitive, and it can benefit from existing 2D video compression techniques. However, the 3D to 2D projection inevitably loses information, and thus affects user-perceived quality. In particular, it is hard to quickly adapt to the fast-changing viewpoints of a user, diluting the key strength of the volumetric video.

3.3 Octree-Based Streaming

Another approach is to use a tree data structure such as octree to utilize the spatial relationships between adjacent points and store 3D coordinates using much less



(a) Octree encoding data size.

(b) Octree encoding and decoding latency.

Figure 3.2: PCL octree-based streaming pipeline performance.

memory space. There has been a number of 3D point cloud compression methods based on octree [20–22]. The basic principle of the methods is to construct an octree data structure from the list of points and generate a byte stream of the occupancy bytes. Then, entropy-encoding methods such as Huffman codes, arithmetic coding, and LZW are applied to compress the byte stream. In this paper, we first focus on the byte stream without entropy encoding. Figure 3.1 depicts the baseline octree-based streaming pipeline: raw point cloud stored at the server is encoded to an Octree byte stream, transmitted over the wireless link, decoded back to point cloud on the client, and rendered on screen.

We implement a baseline octree-based streaming pipeline using PCL, measured on iPhone XS and desktop PC equipped with Intel Core i7-8700 3.2GHz CPU running on Ubuntu 16.04 OS, to show its performance. While Figure 3.2(a) shows that Octree efficiently reduces the large data size to 25% , while Figure 3.2(b) shows that processing latency incurred due to Octree encoding and decoding severely limits the streaming latency. Specifically, we analyze each of the components in the streaming pipeline and identify the performance bottleneck as follows:

• **Octree Encoding.** The octree encoding process at the server involves two steps: 1) octree construction, which adds raw point cloud data into octree data structure defined in PCL, 2) octree serialization, which generates a byte stream from the constructed octree. Figure 3.2(b) shows the encoding latency of *encodePointCloud()* in

PCL, which is responsible for the octree encoding, runs at the scale of 1 fps, limiting the end-to-end latency. The main cause of the latency is in constructing the octree structure from raw point clouds, which requires tree insertion operation for every single point.

• **Octree Decoding.** The Octree decoding process at the client involves decoding the octree-encoded byte stream back to XYZ coordinates. To evaluate the decoding performance, we port the octree decoding function *deserializeTree()* in PCL and measure its latency. Figure 3.2(b) shows that octree decoding only supports 1 to 3 fps, which is far below real-time (30 fps). This is mainly because PCL supports single thread CPU implementation, although the decoding process is highly parallelizable (i.e., calculating XYZ coordinates of each point is independent and can be done simultaneously).

• **Rendering.** It is fast to render 3D models on current state-of-the-art mobile devices with the support of mobile GPUs. As shown in Table 3.1, the rendering time is sufficiently fast to achieve the 30 fps frame rate. Each point in the point cloud can be rendered in a completely parallel manner since there is no dependency between different points.

3.4 Applicability of Existing Techniques

The aforementioned studies show that large data size of point cloud videos makes real-time streaming very challenging, and naively employing octree for data size compression is also infeasible due to large processing overhead. There have been many approaches to tackle a similar challenge in high-resolution 2D or 360° video streaming, it is non-trivial to apply them directly for point cloud videos. Specifically, we analyze why the two most used techniques, *viewport adaptation* and *resolution adaptation* are infeasible as follows:

(a) 2D field-of-view.

(b) 3D view frustum.

Figure 3.3: Field of view in 2D and 3D.

• **Viewport Adaptation.** Another conventional approach employed in 360° video streaming is to selectively stream only a part of the frame within the user's field of view [6,7]. Similar can be achieved by sending only the points that are included in the user view frustum (an intersection of half-spaces of six planes, which are determined by the user's view and projection matrices) as shown in Figure 3.3 (b). However, such a process involves calculating the six planes and conducting a series of matrix multiplication for each point to determine whether or not the point falls in the intersection of the six planes, also incurring significant processing overhead. It is more complex compared to the 2D case, which selects tiles of 2D grid blocks from a fixed configuration as in Figure 3.3 (a).

• **Resolution Adaptation.** To overcome limited network bandwidth or decoding complexity challenge, a common approach taken in 2D and 360° videos streaming is to dynamically adapt the video resolution. A corresponding approach for point cloud videos is to adapt point density by sampling. However, naive point sampling fails in terms of both image quality and processing latency. Figure 3.4, Figure 3.5 and Table 3.2 shows the sampling performance of random and uniform point sampling in PCL applied on 8i longdress dataset. It is shown that random sampling incurs annoying distortions due to randomness in the sampling manner without regarding the spatial density of points. Uniform sampling in PCL uses a similar approach to the spatial division of the octree data structure. It iterates over the points and approximates them

Figure 3.4: Random sampling of 221k(left), 318k(middle), 595k(right) points with PCL.

Figure 3.5: Uniform sampling of 221k(left), 318k(middle), 595k(right) points with PCL.

to the centers of 3D voxel grids which can have a different radius. Although it takes account the spatial distribution of the point cloud unlike random sampling to generate better quality downsampled results it incurs long latency due to high processing overhead of iterating over the points. Most importantly, to maintain user perception quality, the allocated resolution to different parts of the point cloud is highly dependent on the current user's viewpoint which has 6 DoF. Thus, it is difficult to compute and store the different versions of the point clouds for every user viewpoint. Therefore, it is important to develop a runtime sampling scheme.

Table 3.2: Uniform sampling time of PCL.

Avg # of Sampled Points	Latency(ms)
221k	56
381k	78
595k	132

3.5 Summary

Octree structure allows effective and simple compression of 3D point cloud geometry. Also, it generates the spatial relationship between points which is useful in applying user-adaptive techniques without dealing with individual points. However, constructing an octree from raw point clouds and reconstructing the 3D model from the octree structure incurs significant overhead. Therefore, the existing octree-based streaming system cannot support real-time volumetric video delivery to mobile devices. Also, due to the complexity of handling a large number of points in point cloud videos, conventional user-adaptive methods in 2D and 360° videos cannot be applied directly.

Chapter 4

Overview

4.1 Design Considerations

3D Data Delivery To enable fully immersive and interactive user experience, our primary goal is to deliver volumetric video on mobile devices in the form of 3D data. Accordingly, we leave prior approach of streaming 2D projected videos [4] out of scope.

Real-Time Streaming Over Wireless Network We aim at delivering volumetric videos at low-latency, so as to enable real-time streaming on mobile devices (e.g., 30 fps). Furthermore, our target is to support such real-time streaming in an untethered environment (i.e., without a wired link), so that users can move freely around the space exploring the 3D point cloud media without being restricted by wires.

Minimal Video Quality Degradation While supporting real-time delivery, we aim at minimizing the video quality degradation in order not to harm the user quality of experience. We carefully design our system to take into account user behavior and video content to compress the data size without severely harming the user perception quality.

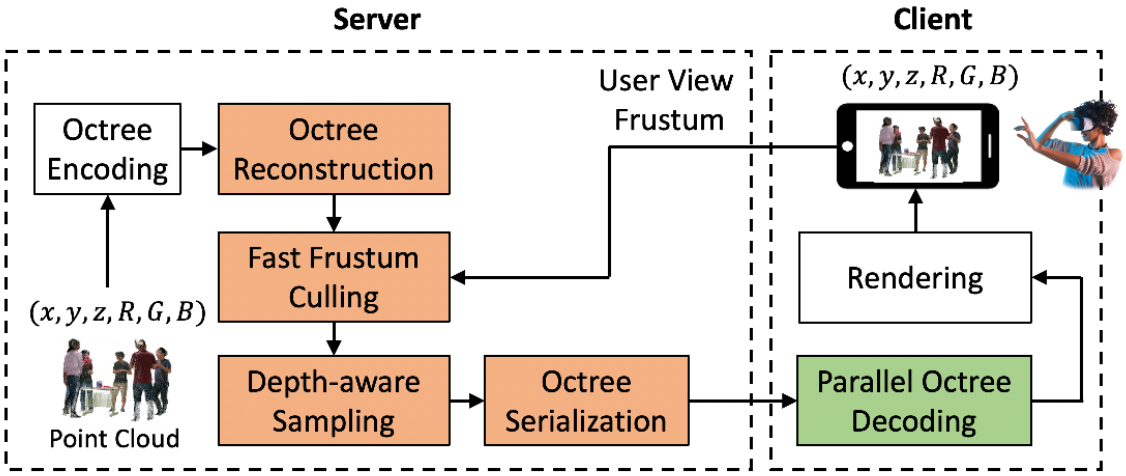


Figure 4.1: Overall system architecture.

4.2 System Architecture

Our proposed end-to-end mobile volumetric video streaming system architecture is shown in Figure 4.1. The system consists of the server part and the client part. The server applies user perception adaptive schemes, frustum culling, and depth-aware sampling, directly on the encoded volumetric video to minimize the runtime latency and the data size. The client utilizes its heterogeneous computing power to decode and render the content in real-time.

First, the x, y, and z coordinates of raw point cloud frames are encoded into an octree structure in advance to its maximum depth offline. During the online stage, the octree structure should be reconstructed from the encoded byte stream to apply the user perception adaptive schemes. To minimize the overhead of this step, we run octree reconstruction where the server does not necessarily decode the byte stream but rather extracts the index information by traversing it only once. Since this step is not dependent on the client's operation, it can be executed in the background process continuously.

When the server receives updates of the user's viewpoint, it predicts the viewpoint of the next few frames. With the predicted user viewpoint, fast frustum culling

is applied to remove points that will not be included in the user's view. To further reduce the number of points, we leverage the unique characteristic of 3D data and apply depth-aware sampling. To further reduce the number of points, two different sampling techniques are applied. Leveraging the unique characteristic of 3D data, we implement a fast and effective point cloud sampling method that adaptively selects the resolution for each octree node regarding the distance from the user in runtime. Also, as in foveated imaging, peripheral points in the current user's view frustum will have less effect in user perception quality compared to points near the user's focus [23]. It is important that the sampling process is done in real-time to provide the users with maximum quality by adapting to their viewpoint continuously. We leverage the multicore processing power at the server to further accelerate the process.

Finally, a frustum-culled and sampled list of points encoded in an octree data structure and colors are sent to the client. When the client device receives the octree byte stream, it performs parallel decoding of the octree byte stream using its heterogeneous processing power. The final output of the decoding process is a list of xyz coordinates perfectly aligned to the color list to be sent to the rendering pipeline.

Handling point cloud data is usually computationally expensive since the data is not organized or fixed in a grid as in 2D images. Therefore, each step of our system is optimized and tightly pipelined to meet the frame rate. With the aforementioned techniques, we present an end-to-end library that fully utilizes heterogeneous computing power both in the server and the mobile device to support real-time streaming of volumetric videos.

Chapter 5

Implementation

In this section, we provide the details of each component in our system depicted in Figure 4.1. For the server-side components, octree reconstruction, fast frustum culling, and depth-aware sampling, and the client-side component, parallel octree decoding, we detail the concepts and algorithms followed by how it is optimized to meet the latency requirement.

5.1 Server: Perception-Adaptive Streaming

5.1.1 Offline Octree Encoding

The point cloud data is typically stored in a raw format, which composed of a long list of xyz coordinates. Not only the size but also any processing considering the spatial context can be bottlenecks for real-time streaming with the unorganized data. We construct the octree as an acceleration geometric structure, and utilize the tree to implement fast frustum culling or depth-aware sampling of the frames of point cloud data which will be detailed in Section 5.1.3 and 5.1.4. However, the octree construction process cannot run in real-time as shown in Section 3.3 and is not dependent on the user viewpoint. Therefore, the octree data structure is pre-processed and separately stored beforehand.

There is no existing standard on saving compressed point cloud data using the octree structure which motivates the design of Modified Point Cloud Data (.mpcd) as in Figure 5.1. The suggested format in heart implements the general octree data structure. The points are first saved into an octree structure to its maximum depth and the individual points in the leaf nodes are quantized to be represented as the center value of the smallest octree node. The quantization effect is trivial when the octree is constructed to its maximum depth where only a single point resides in the leaf node. By traversing this byte stream the relationship between parent and child nodes can be reconstructed and with the initial root node center, side length, and the occupancy bytes, the xyz coordinates of the leaf nodes can be regenerated. The .mpcd file format is designed to parallelize the decoding process to achieve the real-time frame rate. Since calculating the location of octree nodes at the same depth can be parallelized, the octree is encoded in the breadth-first order as a serial byte stream, where the occupancy byte at each octree node indicates which out of the eight child nodes is empty, as described in Section 3.3. However, since the position of a child node depends on its parent node's position, the calculation should be done sequentially for each octree depth which becomes a major bottleneck in the decoding process. To resolve the issue, the .mpcd file format also indicates the number of occupied nodes per depth. The decoding process can utilize the information and correctly find the list of points that are descendants of the nodes that need to be rendered.

The file format is detailed in Figure 5.1. Specifically, we generate a header, which includes the total number of octree nodes, total number of points, maximum octree depth, and number of octree nodes per depth. The header is followed by a payload, where the actual byte stream which represents each octree node is saved. The information can be combined to calculate the offset of individual nodes, and the decoding process can correctly locate any intermediate node considering occupancy of higher levels. The process is further described in the following sections.

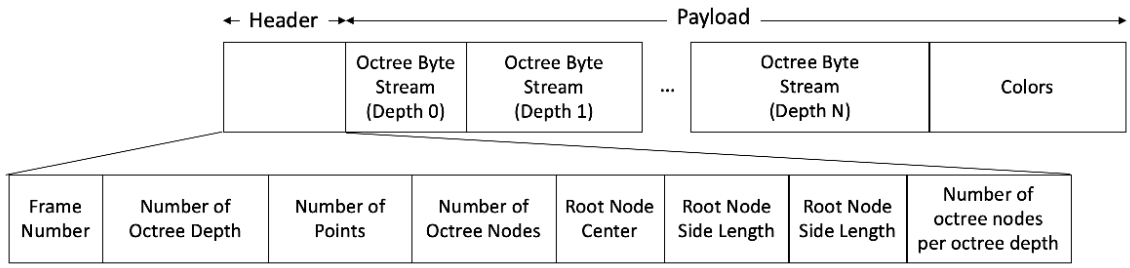
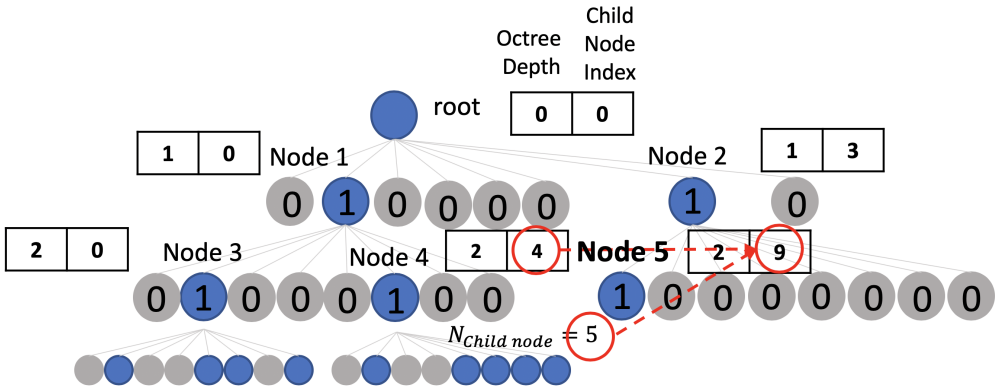


Figure 5.1: Modified point cloud data file format.

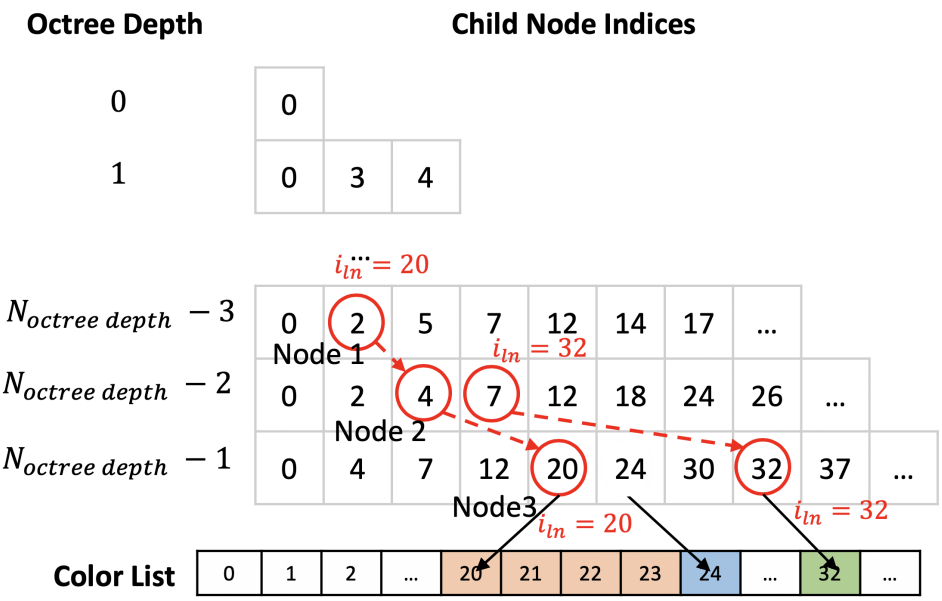
5.1.2 Zero-Conversion Octree Construction

During the runtime process, the octree structure should be reconstructed from the saved file and modified according to the suggested perception-adaptive approaches (Section 5.1.3 and 5.1.4). We design a method to traverse the byte stream a single time to extract the key information such as child node indices and leaf node indices without reconstructing the entire octree structure. When a modified point cloud data file is read, the entire octree is saved in a two-dimensional vector, where nodes of each octree depth is read into a separate vector, as shown in Figure 5.2(b), top. This allows easier indexing to each depth and efficient access to the desired data location out of the irregular structure in the end. This 2D octree vector is traversed a single time and the indexing information is saved to accelerate the following processes.

We store two types of indexing information, child node indices (i_{cn}) and leaf node indices (i_{ln}). The child node index (i_{cn}) of the intermediate branch nodes indicates the index of its first child node in the next octree depth. In our breadth-first tree structure, the child node index is dependent on the preceding nodes in the same octree depth. Specifically, the child node index of each octree node can be saved as a cumulative sum of the occupied preceding nodes of the same depth. If we utilize the auxiliary information stored in the header, we can calculate the child node index after going through the byte stream once. For example, in Figure 5.2(a), the child node index of Node 5 is 9 because the two preceding nodes, Node 3 and Node 4, has 4 and 5 child nodes.



(a) Calculating child node indices(i_{cn}).



(b) Calculating leaf node indicesi_{ln}).

Figure 5.2: Octree reconstruction using child node and leaf node indices.

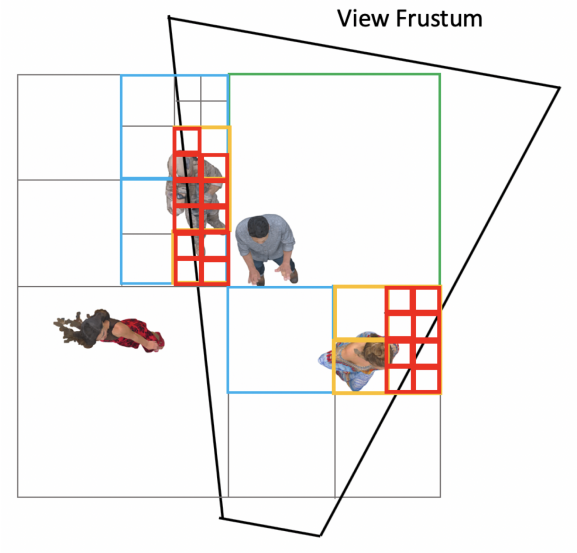


Figure 5.3: Fast frustum culling using octree data structure.

We also save the leaf node index(i_{ln}), which indicates the starting index of the colors that are included in the octree node. Calculating the leaf node index of an octree node is fast since it is the same as its first child's leaf node index as shown in Figure 5.2(b). The leaf node index of the last depth is the same as the child node index. For example, The leaf node index of Node 2 in the figure is the same as the leaf node of its first child node(i_{cn}) which is at the fourth index of the depth below, Node 3. In the same sense, the leaf node index of Node 1 is the same as Node 2. During the frustum culling process, fast indexing with child node indices eliminates redundant tree traversing. Leaf node indices are useful when calculating the average color of the points contained in the last sampled layer during the sampling process.

5.1.3 Fast View Frustum Culling

As mentioned in Section 3.4, the viewport adaptation for volumetric video streaming requires the frustum culling, composed of finding points that lie within the intersection of six half-spaces. In streaming systems, the received updated view at the server might be already a delayed version at the client-side. Therefore, user viewpoint changes should be predicted and applied in advance. User viewpoint prediction in 6

degrees-of-freedom is not in the scope of this work so we assume that it is possible to predict the viewpoint change for the next few frames.

To overcome the challenges, we utilize the octree structure and apply frustum culling at a higher level of octree hierarchy to increase the speed and robustness. Coarser resolution accelerates the frustum culling compared to the case applying the technique into the raw point cloud data. Checking each point independently to determine which points are included in the frustum requires a non-negligible amount of computation. Also, since nearby points have a higher probability of being included or not included in the frustum at the same time, checking every single point has redundant computations. As commonly done in the field of 3D graphics rendering, we make use of the spatial information provided by the octree data structure and perform depth-wise frustum culling as shown in Figure 5.3. If the parent node is not included in the frustum, its child node will not be included either. On the other hand, if the parent node is completely located inside the frustum, all of its child nodes will also be fully included. This approach significantly reduces the amount of computation. Based on the child node indices calculated in the previous step, the location of the child nodes can be calculated fast and efficiently.

Also, to handle user viewpoint prediction errors we set a threshold and stop the frustum culling at a certain depth, which can automatically generate a margin around the edges of the view frustum. The threshold is determined by the side length of the octree node and the width and height of the user view frustum. The spatial margin allows the framework robust to small prediction error of the viewport location, and it is also possible to apply the frustum culling only for every few frames. Leveraging the octree structure, the frustum culling result of a keyframe can be used for successive frames using the AND operation on the octree byte stream rather than performing repetitive matrix multiplication.

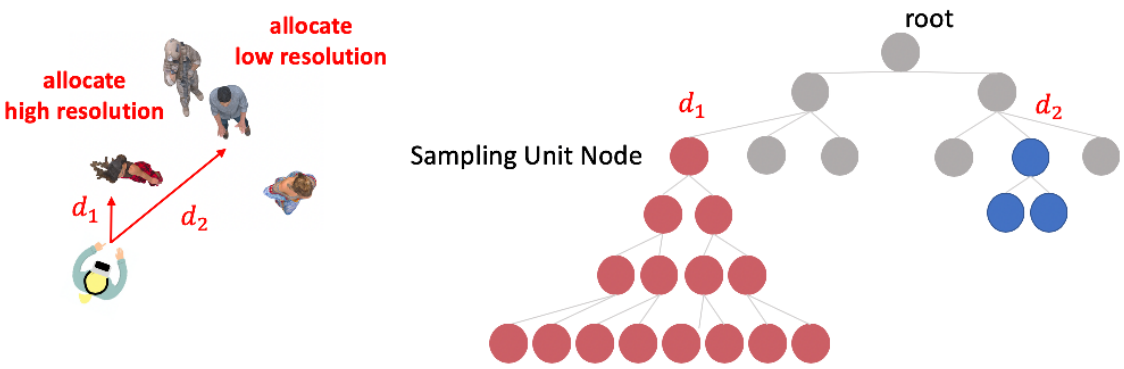


Figure 5.4: Depth-aware point sampling.

5.1.4 Depth-aware Point Sampling

To further reduce the number of points to send while maintaining user perception quality, we perform real-time point sampling based on the distance between the user and the 3D data. Since the perceptual quality depends on the point density per projection on 2D instead of the point density in the 3D space, we allocate a higher density of points to close-by regions and sparse samples of points for further away regions. Runtime sampling can generate the best quality for the current viewpoint of the user and generate a continuous level of resolutions and adapting to the user's viewpoint.

The straight-forward utilization of the octree structure would be selecting the octree depth to display. Each octree node at the sampling depth becomes the unit of resolution allocation, which we name them as sampling unit nodes. This naturally divides the 3D point cloud into small blocks where different resolutions can be allocated. For the center value of each sampling unit node, we apply the view matrix to bring it to the user view coordinate. Based on the depth value and the pixel resolution, we first determine the maximum octree depth. If the maximum octree depth is the same as the original octree's depth, the original colors that are included in the current sampling unit node can be directly used. To display a non-leaf node, the average color of the leaf nodes that are included in the octree node is calculated.

Sampling by the pre-determined octree depth is fast and simple but it results in

(a) Octree depth 8 (60k points). (b) Octree depth 9 (231k points). (c) Octree depth 10 (834k points).

Figure 5.5: Resolution for different octree depths.

a discrete level of resolution. For example, Figure 5.5 shows the rendering results of the longdress dataset represented with three different octree depths. It can be seen that even when there is a single octree depth difference, the number of points increases or decreases by approximately 4 times. This abrupt change of resolution will not be able to maintain user perception quality.

To overcome the above issue, we further separate the level of resolution per depth by determining the maximum nodes at the maximum octree depth. In other words, the sampling density is further refined by selecting how many out of its eight children to use, and the rendering results in the negligible transition of sampling density when observed from the viewpoint. For example, when the maximum number of nodes is set to eight, it means that full resolution at the maximum octree depth is provided. When the maximum nodes are set to four, areas with denser points having more than 4 child nodes will be further subsampled. Through this process, it is possible to perform real-time sampling while providing a continuous level of resolution.

Sampling for each sampling unit node can be executed in parallel. To further reduce the sampling latency, we fully parallelize the sampling process using the multicore-processing power of CPUs.

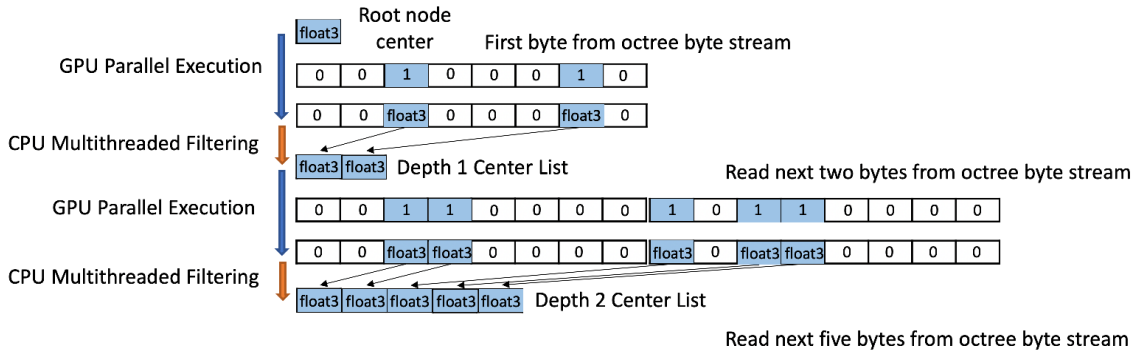


Figure 5.6: Interleaved heterogeneous octree decoding pipeline.

5.2 Client: Real-time Decoding and Rendering

The octree byte stream modified after the frustum culling and sampling process at the server is sent to the client with the list of colors. The client has to regenerate a list of X, Y, Z vertex coordinates of the leaf nodes from this byte stream, which will be aligned with the corresponding color values. Then, these vertices and color values are sent to the rendering pipeline to be visualized on the user's screen.

5.2.1 GPU-based Parallel Octree Decoding

Parallel decoding. The received octree can be decoded in parallel as individual leaf nodes are independent of each other and their locations are only reliant to the location of the ancestors. However, since the octree data structure is saved in a single byte stream we need to figure out which byte represents the child node occupancy of which octree branch node.

Specifically, in Figure 5.6, starting with the first byte of the octree byte stream and the root node center value, the center values of its eight child nodes can be calculated. Since the next bytes in the octree byte stream only consists of bytes representing non-empty nodes, empty nodes should be filtered out. For the two center values of non-empty nodes, the next two bytes from the octree stream is read. Again, calculating the child node centers is done in parallel by the GPU and the empty nodes are filtered out

to calculate nodes in the next octree depth. The parallel computation of center values is fast with GPU programming. However, since the complexity of filtering is $O(n)$, empty node filtering becomes the bottleneck in the decoding pipeline with larger point cloud data. To tackle this problem, we make use of the multi-core processing power of current existing mobile devices.

Heterogeneous Computing. We tackle the problem using the multi-core processing power of current existing mobile devices followed by parallel computations on the GPU to calculate the positions of the leaf node. Performing calculation on the GPU requires a few preparation steps including memory allocation and data copy. When the number of parallel child calculation is small, preparing the pipeline for GPU calculation can become an overhead and thus take longer than doing it on the CPU. When tested on iPhone XS, it is faster to run the process on the CPU than on the GPU when the number of parallel nodes to calculate is smaller than approximately 1000 nodes. Therefore, we set a threshold to determine when to use the CPU or GPU. To accelerate the filtering process, which was the main bottleneck of the decoding process, we implement a multi-threaded filtering function.

5.2.2 Point Cloud Rendering

After the octree is decoded and converted into a list of points, the decoded content needs to be delivered to the user by rendering for the best quality. Point cloud data only have locations without connectivity information between them and rendered individually. Therefore it is very fast to render, but artifacts can be introduced between individual points. We eliminate possible rendering artifacts by adjusting the point size according to the projected distances between neighboring points. For points that are farther away, larger point size is applied to remove the visible spacings between points. On the other hand, for points with higher resolution, smaller point size is used to prevent overlapping of points. The information of the inter-point distances can be inferred from the depth information of each point since depth-aware sampling in Section 5.1.4

was based on them. The point size can be applied for each point individually through the GPU rendering shader function which does not cause an overhead in the rendering process.

Chapter 6

Performance Evaluation

6.1 Experiment Setup

Server. The server-side of our system is implemented on a desktop PC equipped with Intel Core i7-8700 3.2GHz CPU running on Ubuntu 16.04 OS. Functions are all written in C++ in 1,100 LoC. Since network adaptation is not in the scope of this work, we maintain a stable wireless network through connecting a Wireless AP directly to the server computer with a 1Gbps ethernet link.

Client. We implement the client-side system on iPhone XS running on iOS 13.1.3, equipped with a hexa-core CPU(2x2.5 GHz Vortex + 4x1.6 GHz Tempest) and a 4-core GPU with 4GB RAM. Octree decoding and point cloud rendering is implemented based on Apple's ARKit and Metal Framework in Swift with 1,129 LoC.

Dataset. We use two different datasets: CMU Panoptic Dataset [8] and 8i Voxelized Full Bodies [9]. Both of them are open-source datasets captured with state-of-the-art technology but have different characteristics. CMU Panoptic Dataset consists of raw point cloud video data extracted from multiple RGB-D images, with no further processing. Since the visual quality is low due to noisy points, we run some simple denoising processes to generate better quality data using the Open3D library [16]. We choose two sets of videos from the CMU Panoptic Dataset, 170915_office1 dataset, and

(a) office (b) pizza1 (c) pizza2

Figure 6.1: Evaluation dataset sample capture.

160906_pizza1, which consists of 223k and 344k points on average. To test our user-adaptive algorithms, we concatenate the pizza dataset side-by-side to generate a larger scale point cloud video. The dataset configuration is listed in Table 3.1 with sample captures in Figure 6.1. Sample videos from 8i Voxelized Full Bodies are short captures of moving individuals. They are higher in quality with a larger number of points and higher resolution. Four sample videos are provided including longdress, loot, red and black, and soldier. We concatenate longdress and loot to evaluate the visual quality of our depth-based sampling scheme.

Baselines. Since there are no open-sourced volumetric video systems, we implement naive baseline systems for comparison: 1) *Raw*: streams raw X, Y, Z, R, G, B values to be rendered directly on the client device using different wireless network and 2) *Strawman PCL*: implemented based on PCL, the server sends octree encoded frames and client decodes and renders them. We did not apply data compression techniques such as entropy encoding so we skip the process in PCL's pipeline as well for a fair comparison in latency.

6.2 Overall Performance

Figure 6.2 and Figure 6.3 shows the performance of our system compared to the two baseline systems in terms of average streamed data size and end-to-end latency. Strawman PCL resulted in larger latency than streaming raw point cloud, even though the

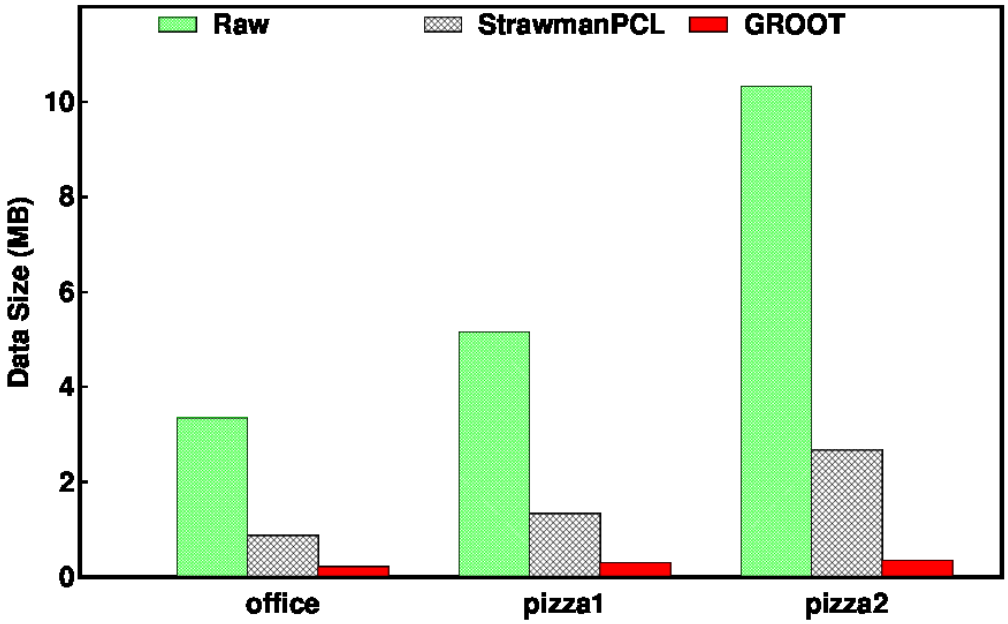


Figure 6.2: Average streamed data size.

streamed data size reduced by 3.87×. At an edge server environment with a single-hop stable wireless network of 5GHz, the end-to-end latency of streaming raw point cloud video was 45ms, 97ms, and 151ms for each dataset. With 2.4GHz wifi, the latency increased by 4.8× while the end-to-end latency of Strawman PCL increased by 13.1×. Compared to Raw 5G, raw 2.4G and Strawman PCL, our system improved the end-to-end latency by 1.3×, 6.7× and 18.0×, respectively. At the same time, our system reduced the data size by 20.6× and 5.3×.

Latency breakdown. Figure 6.4 shows the latency of each component in our system evaluated with the pizza1 dataset. Since the system is pipelined, if each component meets the latency requirement, the total system will be able to deliver point cloud video in 30fps. We observe that octree decoding on mobile devices is the main bottleneck of our system.

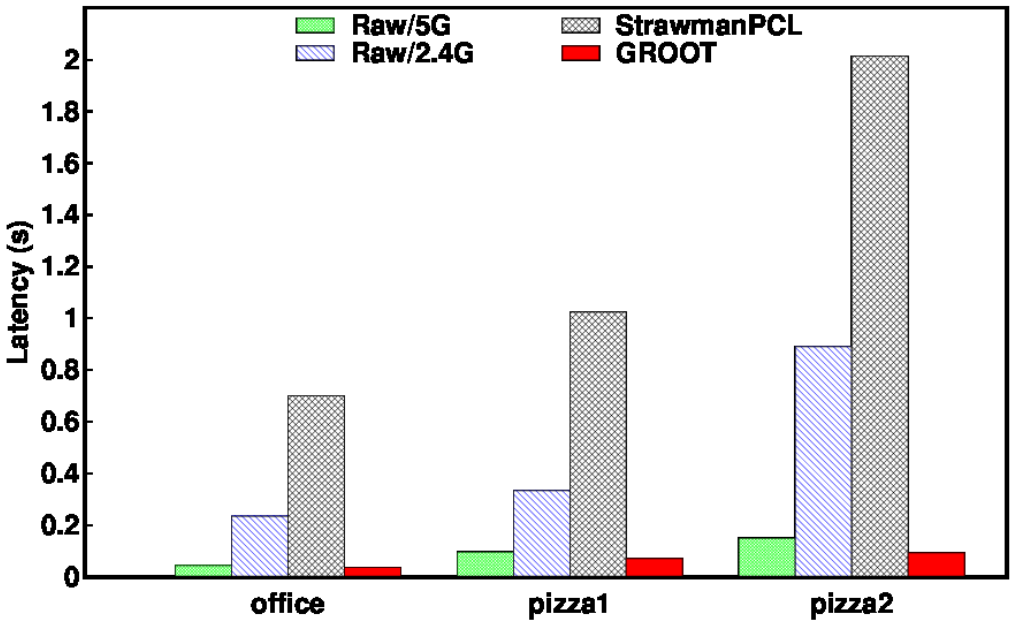


Figure 6.3: End-to-end latency.

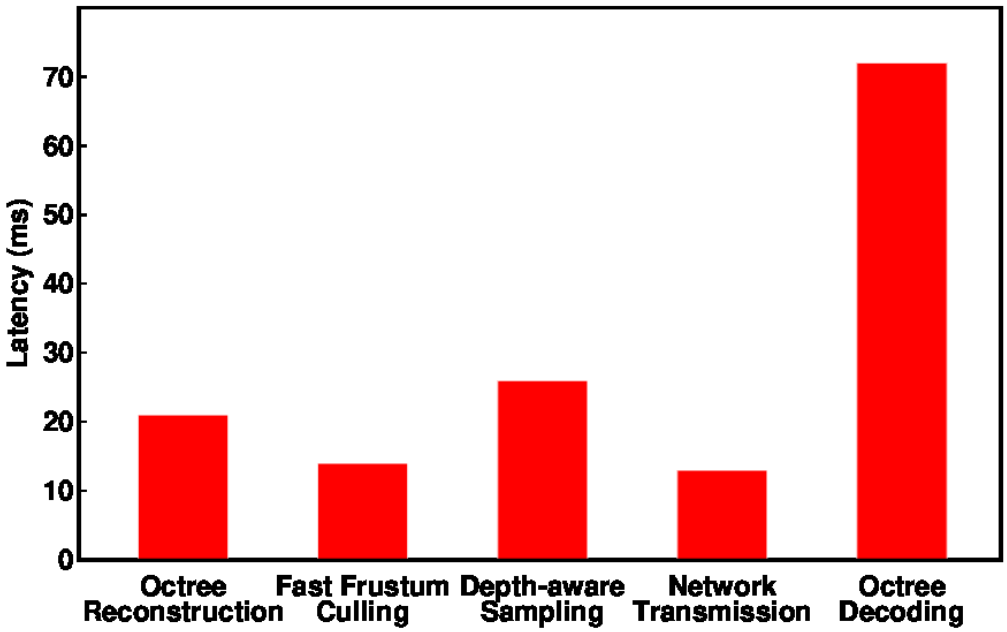


Figure 6.4: End-to-end latency breakdown.

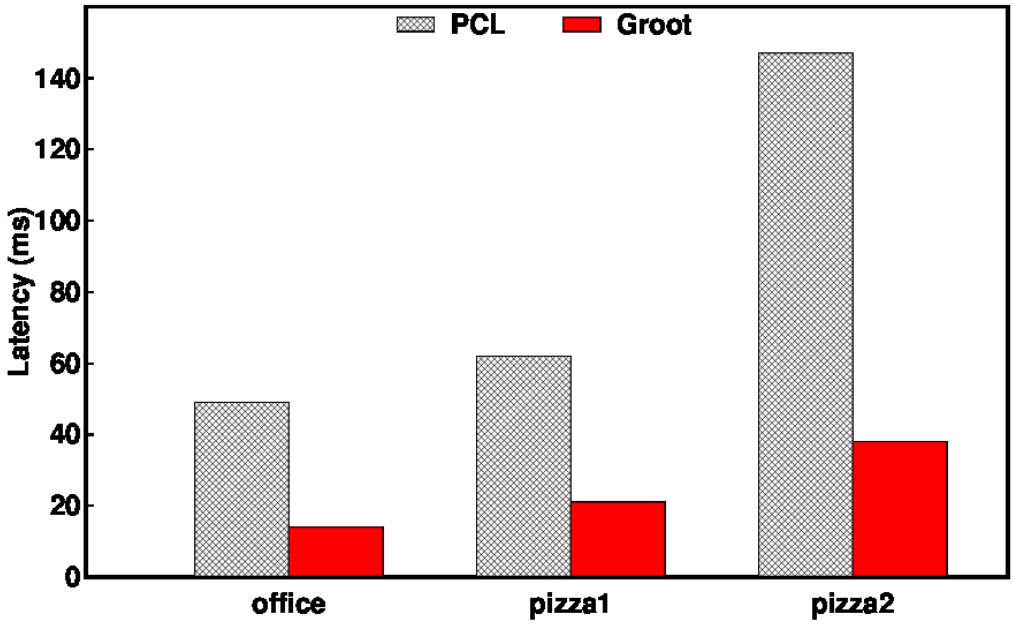


Figure 6.5: Octree reconstruction latency.

6.3 Micro Benchmarks

6.3.1 Octree Reconstruction Overhead

Figure 6.5 shows the runtime overhead of the octree reconstruction process. We compare the results with the octree deserialization function in PCL [14], which reconstructs the octree data structure from the byte stream. Our octree reconstruction process can meet the frame rate of 30fps while extracting necessary information from the encoded byte stream. The octree reconstruction step is fast and is essential to support fast frustum culling and depth-aware sampling

6.3.2 Frustum Culling Latency

As shown in Table 6.1, frustum culling latency is below 10ms. Compared to checking every single point, using the spatial information with the octree data structure can significantly reduce the runtime overhead. The result shows the effectiveness of the octree data structure.

Table 6.1: Frustum culling latency.

Name	Number of Points	Fast Frustum Culling
office	223k	4ms
pizza1	344k	6ms
pizza2	689k	10ms

6.3.3 Depth-aware Sampling Analysis

We evaluate our depth-aware sampling scheme in terms of perceived quality and sampling latency. We compare our results with octree depth-based sampling, which simply generates a different level of detail adjusting the octree depth. From the 8i Voxelized Full Bodies Dataset [9], we concatenate two people and locate them at different distances from the user to test our proposed method.

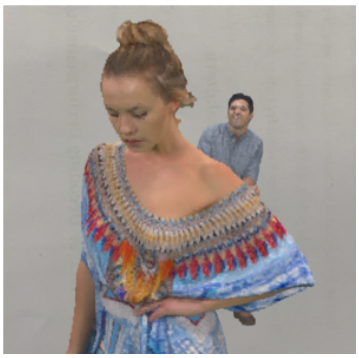
Figure 6.6 shows the sampling results. Even though the octree depth difference between Figure 6.6(a) and Figure 6.6(b) is only one, the resolution degrades significantly. Also, since the sampling is applied to the entire point cloud, the overall frame quality degrades. When our sampling scheme is applied as in Figure 6.6(c) different octree depth is allocated regarding the distance from the user and also by adjusting the number of maximum leaf nodes there are no discrete changes between resolutions.

Lastly, Figure 6.7 shows the time overhead of running sampling in run time. Without parallel computation, the time overhead of serially performing sampling for each sampling unit node is non-negligible. Through multithreading, the process can be accelerated by 2.89$\times$ to 4.81$\times$. It is shown that the runtime sampling process in runtime is possible for frames around 500k points.

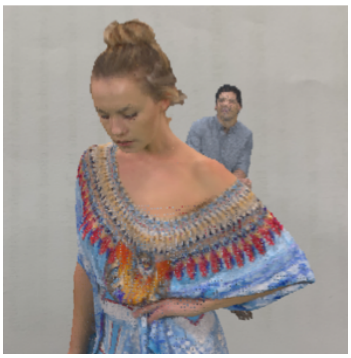
6.3.4 Octree Decoding Performance on Client

We evaluate the octree decoding performance of our system. Figure 6.8 shows how the heterogeneous computing power of current mobile devices can accelerate decoding

(a) Uniform sampling to octree depth 8 (58k)



(b) Uniform sampling to octree depth 9 (230k)



(c) Ours with depth-aware octree depth allocation (162k)

Figure 6.6: Sampling results.

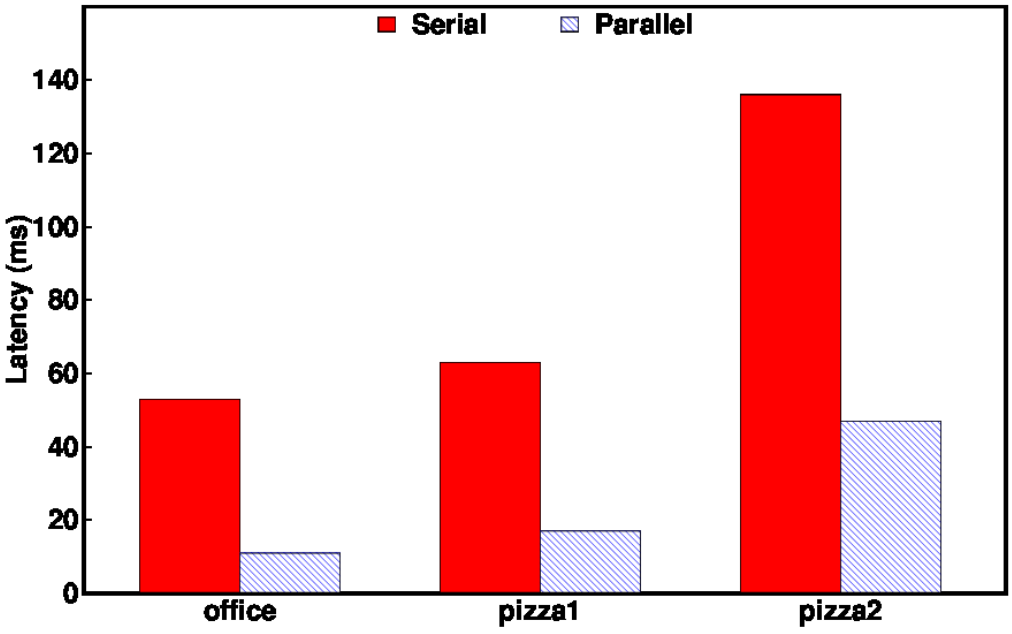


Figure 6.7: Depth-aware point sampling latency.

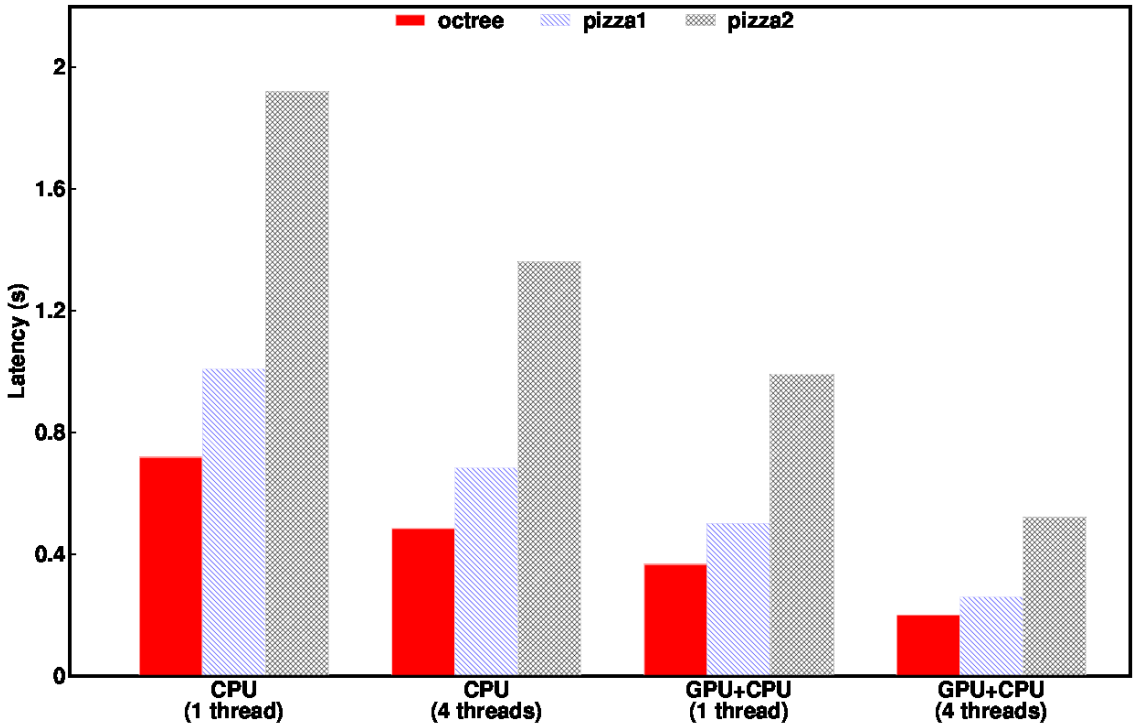


Figure 6.8: Octree decoding latency on mobile devices using heterogeneous computing power.

performance. Without any parallelization, decoding a single frame takes more than 1 second and with multithreading, it can be improved by only 1.4×. For the octree nodes in the same depth, calculating their child node can be done in parallel. Executing this on the GPU can further accelerate the process. Finally, tight cooperation between multicore CPUs and the GPU allowed an overall performance improvement of 2.7× to 4.3×. We also tested on different mobile devices. As shown in Figure 6.9, the decoding speed of iPhone 11, which is the next generation of iPhone XS, improved by 1.2×.

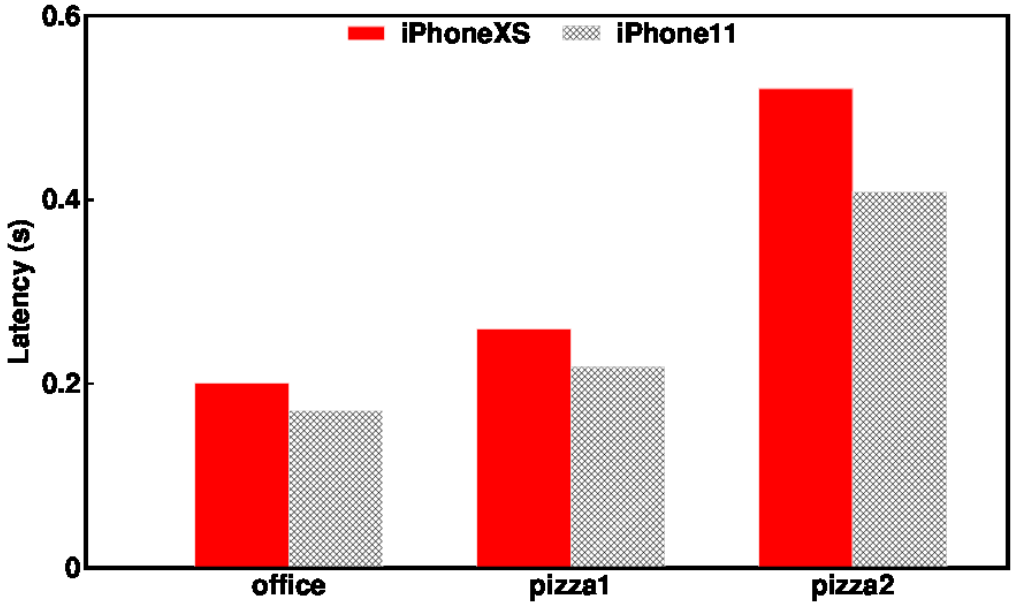


Figure 6.9: Octree decoding latency measured on different mobile devices.

Chapter 7

Discussion

In this section, we discuss the performance results of our system along with the remaining challenges and issues.

7.1 Efficiency of Octree Data Structure

Octree data structure eliminates the need for storing each X, Y, Z coordinates, which reduces the memory size significantly. Also, it is effective in particular since it generates the spatial information of a set of independent and unorganized points. This can accelerate the process of handling and applying different techniques such as frustum culling and sampling. Despite the advantages of the octree data structure to store and process 3D point cloud data, we have seen in Chapter 5 the difficulties in streaming octree encoded point clouds. The difficulties stem from the fundamental difference between 2D and 3D videos. In 2D videos, each pixel value in the list of colors is matched to a specific location in the 2D pixel grid. However, in the 3D video, since the size, the number of points, and orientation differ for every frame, they cannot be stored in a fixed grid. This makes it difficult to figure out which color value in the serial list is matched to which point.

Although the decoding performance of our system outperforms conventional meth-

ods, it still has limited performance to meet the 30fps frame rate. This is because although the parallel computation on the GPU is fast, the multi-threaded filtering process on the CPU throttles the performance. Since GPUs do not provide dynamic buffer functionalities, this process needs to be done on the CPU in serial, which has a complexity of $O(n)$. Therefore, the maximum number of points that can be decoded is bounded to meet the 30fps frame rate. To tackle this problem, one approach can be further reducing the number of points to decode through inter-frame compression. There are a few studies on inter-frame compression of point cloud videos [24] [25], but the compression rate is low and incurs higher complexity to be applied to mobile systems. This is because unlike 2D videos, calculating the motion vector between adjacent point cloud frames is much more complex since the movements of each point are inconsistent and the number of points for each frame differs. Therefore, a lightweight inter-frame compression scheme that can be applied to mobile systems should be developed, which remains as our future work.

7.2 User Adaptive Runtime Sampling.

The user viewpoint in 3D videos can be extremely diverse. To provide the user with the best quality while minimizing the latency and bandwidth consumption, it is important to effectively allocate the limited number of points to different parts included in the user frustum. To achieve this goal, we do not store pre-sampled results but rather execute the sampling process in runtime as opposed to conventional approaches in [26, 27]. Our proposed scheme is lightweight to meet the 30fps frame rate.

In 2D videos, generating different resolutions can be done by interpolating nearby pixels that are fixed in grids. Bilinear or bicubic interpolation are some of the commonly used methods. Since the filtering results in the same size image still having color values for every pixel, the user might not be able to perceive the difference between different resolutions as much. On the other hand, a different resolution in point

cloud video is the process of sampling a different number of points. Due to the discrete nature of point cloud video, sampling might create unwanted visible spacings between points as shown in Figure 6.6, thus degrading the sense of realism. This problem could be solved by our sampling method since the proposed sampling process can provide a continuous level of resolution adaptive to human visual acuity.

7.3 Remaining Issues

7.3.1 Color Compression

In this work, we did not consider any color compression. Since the octree data structure significantly reduced the data size to represent the X, Y, Z coordinates, which was the main burden in terms of data size, now the color data size takes up the main portion in the total streamed data. To solve this problem, we are planning to apply conventional H264 or JPEG image compression techniques by packing colors into a 2D image to further reduce the data size. Even though H264 and JPEG decoding are fast on mobile devices through hardware acceleration, since it will require further mobile processing power, task scheduling between octree decoding, color decoding, and rendering remains as a challenging issue.

7.3.2 Viewport Estimation

To the best of our knowledge, estimating viewports with 6 DoF is still a remaining challenge. Thus, this will be a new research direction on its own. Since the movement in 3D space consists of rotation and translation, it will be much challenging than viewport estimation in 3 DoF, which were studied in the context of 360 video streaming. The first step would be to perform an extensive user study to collect a large amount of data. Then, applying methods from the recent advancement in machine learning and deep learning, we expect the accuracy of viewport estimation in 6 DoF to reach a sufficient level in the near future.

Chapter 8

Conclusion

In this paper, we proposed an optimized end-to-end system for streaming 3D point cloud videos to mobile devices. Streaming point cloud video is rather an under-explored area of research with few existing works. Therefore, the challenges and motivations of developing such a system have not been investigated enough. We identified some of the challenges and presented a novel end-to-end pipeline that can effectively deliver point cloud videos to mobile devices.

We designed and presented how point cloud data should be saved and read with minimal overhead, how to leverage the depth information to reduce the data size while maintaining user experience quality, and a mobile platform for decoding encoded point cloud data. For our future work, we plan to further optimize each component in our system to achieve the 30fps frame rate. Furthermore, we aim to integrate this system into other promising applications, such as mixed reality and interactive media.

Bibliography

[1] "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/.

[2] "Verizon 5g labs show off what's next in tech," https://www.verizon.com/about/news/verizon-5g-labs-show-whats-next-tech.

[3] "What is a tof camera and which phones have it? time-of-flight sensor explained," https://www.pocket-lint.com/phones/news/147024-what-is-a-time-of-flight-camera-and-which-phones-have-it.

[4] 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*, ser. HotMobile '19, 2019.

[5] P. A. Kara, A. Cserkaszky, M. G. Martini, A. Barsi, L. Bokor, and T. Balogh, "Evaluation of the concept of dynamic adaptive streaming of light field video," *IEEE Transactions on Broadcasting*, vol. 64, no. 2, pp. 407–421, June 2018.

[6] 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*, ser. MobiSys '18, 2018.

[7] 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*, ser. MobiCom '18, 2018.

[8] 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.

[9] Eugene d'Eon, Bob Harrison, Taos Myers, and Philip A. Chou, "8i Voxelized Full Bodies - A Voxelized Point Cloud Dataset," 2017. [Online]. Available: https://jpeg.org/plenodb/pc/8ilabs/

[10] "Intel LibRealSense," https://github.com/IntelRealSense/librealsense.

[11] "LiveScan3D," https://github.com/MarekKowalski/LiveScan3D.

[12] D. Meagher, "Geometric modeling using octree encoding," in *Computer Graphics and Image Processing*, 1982.

[13] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, Sep. 1975.

[14] "Point Cloud Library (PCL)," https://github.com/PointCloudLibrary/pcl.

[15] "Google Draco," https://google.github.io/draco/.

[16] "Intel Open3D," https://github.com/intel-isl/Open3D.

[17] "Point cloud compression," https://mpeg.chiariglione.org/standards/mpeg-i/point-cloud-compression.

[18] 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 the 23rd*

Annual International Conference on Mobile Computing and Networking, ser. MobiCom '17, 2017.

[19] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li, J. Llach, 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.

[20] Y. Huang, J. Peng, C.-C. J. Kuo, and M. Gopi, "Octree-based progressive geometry coding of point clouds," in *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, ser. SPBG'06. Goslar, DEU: Eurographics Association, 2006, p. 103–110.

[21] 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*, May 2012, pp. 778–785.

[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)*, Oct 2018, pp. 1807–1811.

[23] W. S. Geisler and J. S. Perry, "Real-time foveated multiresolution system for low-bandwidth video communication," in *Human Vision and Electronic Imaging*, 1998.

[24] R. L. de Queiroz and P. A. Chou, "Motion-compensated compression of point cloud video," in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017.

[25] 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.

[26] J. Park, P. A. Chou, and J. Hwang, "Volumetric media streaming for augmented reality," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018.

[27] M. Hosseini and C. Timmerer, "Dynamic adaptive point cloud streaming," in *Proceedings of the 23rd Packet Video Workshop*, ser. PV '18, 2018.

초 록

볼류메트릭 비디오는 공간에 대한 깊이 정보를 담은 3차원 비디오이며 차세대 멀티미디어로 주목 받고 있다. 기존 360도 비디오는 사용자에게 축 회전에 해당하는 움직임을 자유롭게 할 수 있는 3 자유도(3 DoF)를 제공한다. 볼류메트릭 비디오는 한 단계 더 나아가 전후, 좌우, 위아래로 이동할 수 있는 추가적인 자유도와 함께 6 자유도(6 DoF)를 제공한다. 최근 기술 발전이 급격히 이뤄지고 있는 가상 현실, 증강 현실, 혼합 현실 기술과 함께 볼류메트릭 비디오는 사용자가 3차원 공간을 인지하고 직접 참여할 있는 인터렉티브 미디어를 실현한다. 사용자가 볼류메트릭 비디오가 제공하는 6자유도를 실감하기 위해서는 선의 제약 없이 모바일 기기를 통한 미디어를 제공받을 수 있어야 한다. 최근 5G 네트워크 기술의 발전과 고성능 CPU 와 GPU 를 장착한 모바일 기기의 발전으로 이를 실현할 수 있는 가능성이 있지만 여러가지 해결 되지 않은 어려움이 존재한다. 볼류메트릭 비디오는 3차원 공간에 대한 정보를 저장해야하기 때문에 기존 이차원비디오에 비해 훨씬 큰 데이터 크기를 가진다. 또한, 볼류메트릭 비디오를 압축하는 현존하는 기술들은 모바일 기기에서 실시간 처리를 제공할 수 없다. 더 나아가 기존 비디오 스트리밍 시스템에 흔히 사용된 가변 비트레이트 (Adaptive Bit rate) 스트리밍 이나 360도 비디오 스트리밍 시스템의 핵심 기술 중 하나인 사용자의 시선에 해당하는 부분만 적응적으로 보내는 기술은 볼류메트릭 시스템에 그대로 적용할 수 없다. 본 논문에서는 처음으로 앞서 언급된 문제점들을 해결하는 볼류메트릭 비디오 스트리밍 시스템을 제안한다. 기존 볼류메트릭 비디오를 압축하는 방식을 효율적으로 실시간 처리하는 방법을 제안하고, 사용자의 시선뿐만 아니라 위치에 따라 적응적으로 스트리밍하여 데이터 양을 줄이

고 사용자의 경험 품질을 유지하고자 한다. 데스크톱 컴퓨터와 모바일 기기에 직접 프로토타입을 구현 후 최신 데이터셋으로 성능 평가를 하여 해당 시스템의 가능성을 증명한다.

주요어: 볼류메트릭 비디오, 인터렉티브 미디어, 포인트 클라우드, 이기종 컴퓨팅

학번: 2018-22221

ACKNOWLEGEMENT

This dissertation would not have been possible without the support of many people.

It is a privilege for me to have worked with so many amazing professors. I would like to express my deep gratitude to my advisor Professor Youngmin Kim for her patient guidance and encouragement to accomplish this research project. Although I knocked on her door with nothing but a passion to start a new project, she dedicated her time and effort to provide me with insightful comments and to push me further to the next step. Her enthusiasm and dedication inspired and motivated me to actually enjoy doing research.

I cannot go without mentioning Dr. Sunghyun Choi, my former supervisor for three semesters, who embarked on his new journey as a senior vice president at Samsung Research. I was very lucky to have the chance to work with him. He is one of the smartest and yet the most hardworking person I have ever met. His profound and sharp-witted insights broadened my perspective and developed my logical thinking ability. More importantly, many of his advice taught me to always try my best, never give up, and always challenge myself.

I was very lucky to have a chance to work with professor Youngki Lee. I really appreciate his encouragement and practical pieces of advice in every meeting that I frequently asked for. I appreciate professor Saewoong Bahk for being on my defense committee and providing helpful feedback and warm advice. I would also like to thank Dr.Jonghan Lim for teaching me with patience and encouraging me with previous

projects.

I am grateful to all my collaborators from Multimedia and Wireless Networking Lab(MWNL) who I worked closely with. I would like to give a special thanks to Juheon Lee who was a great mentor for this work. Also, Gyujin Lee, who was my first Ph.D. mentor, gave me the chance to have hands-on experience with ongoing projects and provided insightful comments which helped me become a more independent researcher. It was a great pleasure to be on the same team with Kitaek Lee and Heejin Yang. I learned the importance of teamwork and collaboration through them. My first teammate in 3D Vision Lab was Jin Seok Bae. I would like to thank him for his dedication to the project.

I have to thank many people who have contributed to my day-to-day life at SNU. From MWNL: Seongwon Kim, Seungil Park, Kangjin Yoon, Changmok Yang, Sundo Kim, Youngwook Son, Jihoon Kim, Jaehong Yi, Junseok Kim, Hoyoung Yoon, Junyoung Choi, Cheolyoung Kwak, Sunwook Hwang, Jihwan Lee, Kanghyun Lee, Jaewon Hur, Suhun Lim, Hanyeoreum Bae, and Jinmyeong Lee. From 3D Vision Lab: Hyuntae Son, Juhyeon Kim, Cheolhui Min, Dongsu Zhang, Jeonghwan Kim, and Junho Kim.

My parents are my role models in every aspect of life. I would not be where I am today without them. Their strong support, encouragement, love, and faith allowed me to pursue my dreams without hesitation. I am also very grateful to have my one and only sister Kyungyun Lee who is also a researcher in a similar field. Her passion for exploration and challenges with determination always motivates and inspires me. Lastly, nothing would have been possible without the unconditional love and support from Joshua Ngo despite the long distance between us.