Characterizing Real-Time Dense Point Cloud Capture and Streaming on Mobile Devices

Jinhan Hu, Aashiq Shaikh, Alireza Bahremand, Robert LiKamWa
Meteor Studio, Arizona State University
Tempe, Arizona, USA
jinhanhu,ashaik11,abahrema,likamwa@asu.edu

ABSTRACT

Point clouds are a dense compilation of millions of points that can advance content creation and interaction in various emerging applications such as Augmented Reality (AR). However, point clouds consist of per-point real-world spatial and color information that are too computationally intensive to meet real-time specifications, especially on mobile devices. To stream dense point cloud (PtCl) to mobile devices, existing solutions encode pre-captured point clouds, yet with PtCl capturing treated as a separate offline operation.

To discover more insights, we combine PtCl capturing and streaming as an entire pipeline and build a research prototype to study the bottlenecks of its real-time usage on mobile devices, consisting of a depth sensor with high precision and resolution, an edge-computing development board, and a smartphone. In a custom Unity app, we monitor the latency of each operation from the capturing to the rendering, as well as the energy efficiency of the board and the smartphone working at different point cloud resolutions. Results reveal that a toolset helping users efficiently capture, stream, and process color and depth data is the key enabler to real-time PtCl capturing and streaming on mobile devices.

KEYWORDS

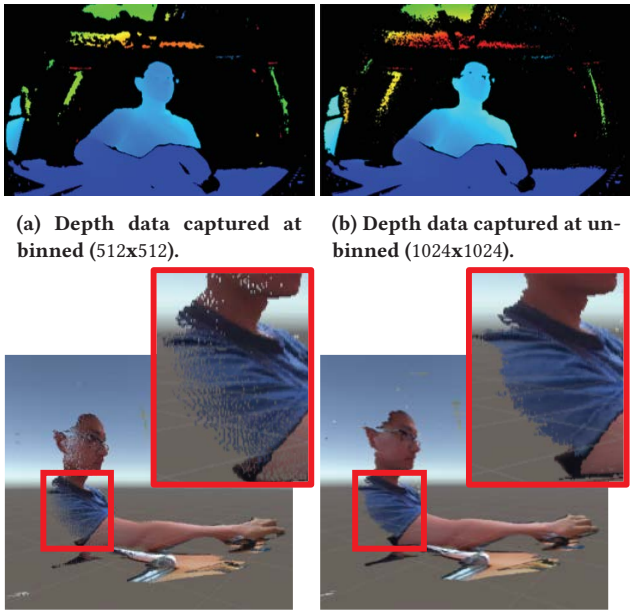
Dense point cloud streaming prototype; Performance and energy characterization; Point cloud rendering on mobile devices

ACM Reference Format:
Jinhan Hu, Aashiq Shaikh, Alireza Bahremand, Robert LiKamWa. 2022. Characterizing Real-Time Dense Point Cloud Capture and Streaming on Mobile Devices. In *3rd ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges (HotEdgeVideo'21), January 31-February 4, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3477083.3480155

1 INTRODUCTION

Point clouds containing 3D spatial data can enable users to interact with the physical world sensed through a digital device in an immersive way to improve performance, enhance knowledge, and spark imagination in various tasks. The ongoing development of cutting-edge sensing hardware, software, algorithms, and associated development kits have enabled the usage of point clouds in a wide range of exciting use cases including computer vision, artificial intelligence, autonomous driving, entertainment, and augmented reality, etc [10, 31, 32, 34, 43].

Despite these advancements, a challenge remains in that point cloud data is either dense with high depth precision that can only be processed on desktops with high-end GPUs, or sparse with low depth precision that is suitable to be processed on mobile devices. The smooth and dense point cloud enables fine-grained data visualization but comes with the cost of limited user mobility and interactivity. On the other hand, the sparse point cloud on mobile devices enables mobility and real-time user interactivity but comes with the cost of impaired visualization quality and task accuracy. As shown in Table 1, a comparison of state-of-the-art 3D sensors on/off mobile devices demonstrates a balanced resolution-power tradeoff [3, 15, 16, 30, 37]. Figure 1 shows a comparison of depth data and PtCl captured at two resolutions. Compared with depth data captured at the binned (512x512) mode (in Figure 1a), the



(a) Depth data captured at binned (512x512). **(b) Depth data captured at unbinned (1024x1024).**

(c) PtCl reconstructed from 720p color data and unbinned depth data. **(d) PtCl reconstructed from 2160p color data and unbinned depth data.**

Figure 1: A comparison of depth data and PtCl captured at two different resolutions.

HotEdgeVideo'21, January 31-February 4, 2022, New Orleans, LA, USA

ACM ISBN 978-1-4503-8700-2/22/01…$15.00
https://doi.org/10.1145/3477083.3480155

Depth Sensor	Resolution	Power
Azure Kinect	512x512	225 mW
	1024x1024	950 mW
Intel RealSense D435i	1280x720	752 mW[1]
ToF on Samsung S21	640x480	160 mW
LiDAR on iPhone 12	256x192	N/A[2]

Table 1: A comparison of state-of-the-art 3D sensors on/off mobile devices.

depth is smoother, richer, and more precise captured at the unbinned (1024x1024) mode (in Figure 1b). Similarly, PtCl reconstructed from the 2160p color and unbinned depth data (in Figure 1d) is more consistent and detailed than it is reconstructed from the 720p color and unbinned depth data (in Figure 1c), especially at the boundary.

To bring dense point cloud (referred to as PtCl in this paper) to mobile devices, related works mostly focused on reducing the amount of data to be transmitted, offline after the data is already captured. For example, some works first encode PtCl offline using state-of-the-art libraries [4, 21] and then stream and decode them on mobile devices [9, 18, 33]. Some other works further divide PtCl into different regions based on the position of each point and then represent different regions using different resolutions to utilize the resolution-based accuracy and performance tradeoffs [35, 40]. Additionally, other existing works utilize dynamic adaptive streaming over HTTP (DASH) to dynamically determine the amount of PtCl to be streamed under different networking conditions [12, 17, 38].

Mobile devices are able to live-stream 4K videos at more than 30 FPS, but why not PtCl? To identify the bottlenecks of real-time PtCl capturing and streaming on mobile devices, we built a research prototype using three off-the-shelf components, consisting of an RGB-D sensor with high depth resolution and precision, an edge-computing development board, and a smartphone, as shown in Figure 2 inspired by [41]. On top of this prototype, we develop and run a Unity application on the Android platform and perform several experiments to monitor the latency of each important operation in the capturing-to-rendering pipeline (including view transformation, data preparation, data transmission, etc.), as well as the energy efficiency (energy consumption per frame) of the board and the smartphone.

The findings from the experimental results are straightforward: *(i)* the throughput of the hardware on mobile devices is limited, therefore mobile devices cannot handle the massive amount of raw PtCl. For instance, one point of RGB-D data captured by an Azure Kinect sensor consists of 3 bytes color values and 12 bytes depth values. This results in 151 MB data to be captured and streamed per frame if working at 4K resolution, requiring a throughput of 36Gbps working at 30 FPS. As shown in our evaluation, rendering PtCl (captured at a resolution as low as 720p) in the Unity app running on a Pixel 3XL struggles to meet real-time specification even with GPU acceleration. *(ii)* the representation of PtCl is not optimized for rendering on mobile devices. The view transformation and XYZ-RGB data format make the streaming of PtCl unnecessarily large in data size. Although point cloud libraries can encode spatial information to reduce the data size, they are not suitable for encoding color info and their latency is unacceptable for live-streaming. In addition, there is an optimization gap between raw PtCl and its representation in rendering components, e.g., particle and its texture material.

These findings motivate the development of a software stack that can help upper level PtCl apps running on mobile devices sense 3D space efficiently, in which these apps can: *(i)* adaptively define the region of interest to capture and stream with the least amount of PtCl needed. Reducing the amount of data captured under the premises of maintaining app performance can reduce operation latency and improve energy efficiency from the root. *(ii)* dynamically determine the operations to be performed on/off device. The entire PtCl capturing-to-rendering pipeline involves multiple operations that can be optimized to balance the performance and energy tradeoffs on mobile devices. *(iii)* drive rendering-oriented PtCl capturing and processing. PtCl can be tailored based on the component used in the rendering engine to enable geometry-based tradeoffs. *(iv)* plug and play in 5G environment. We look forward to applying and testing our work in emerging 5G-based interactive PtCl applications to unlock more possibilities that are currently limited by the processing power of mobile devices.

2 RELATED WORK

2.1 Adaptive Resolution

Resolution-based accuracy and performance tradeoffs exist in both 2D color images and 3D depth data [14, 22]. Hu *et al.* introduced Banner [13, 23] which is a media framework that enables fast sensor resolution reconfiguration such that AR apps can make use of resolution-based tradeoffs to extend battery life while maintaining a smooth AR experience. In [7], Ha *et al.* studied the improvements in energy usage and response time after lowering the sensor resolution for object detection tasks on a Google Glass wearable device. In [24], LiKamWa *et al.* verified the energy proportionality to image sensor resolution and frame rate. In [40], Wang *et al.* proposed a voxel-based representation of the LiDAR point cloud, in which a voxel is reconfigured with neighbors having different resolutions such that the convolution is more efficient and the 3D object detection accuracy is improved. Riegler *et al.* introduced OctNet [35], in which less important regions are represented with less data, while important regions are represented with more data resulting in improved overall learning accuracy.

2.2 Dynamic Point Cloud Streaming

Dynamic adaptive streaming over HTTP (DASH) [1] is widely adopted in video streaming to dynamically adjust the video quality under various network conditions. In [17], Lederer *et al.* introduced a public DASH dataset as well as a DASH encoder to enable the comparison between different adaptation algorithms. On top of DASH, Hosseini *et al.* presented DASH-PC [12] to adaptively stream point clouds, with several optimization techniques integrated such as exploiting layer of density (to dynamically spatial subsampling the point cloud). In [38], Subramanyam *et al.* introduced a tiling-based approach exploiting user movement to reduce the bit rate for

[1]The power consumption of D435i module is calculated by combining the power consumption of dual wide imagers and IR.
[2]LiDAR component cannot be treated as a standalone module to be measured due to core underlying modules of the ARKit SDK.

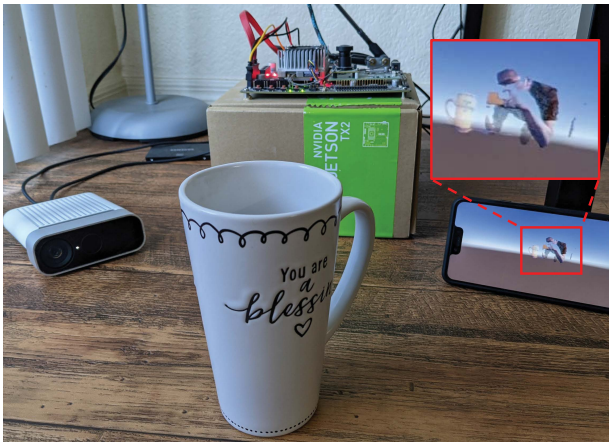


Figure 2: Our research prototype (an Azure Kinect, a Jetson TX2, and a Pixel 3XL from left to right) for characterizing real-time PtCl capturing and streaming on mobile devices.

streaming point clouds with user perceptual quality maintained. DASH can be used as a fundamental component to facilitate real-time PtCl capturing and streaming on mobile devices.

2.3 Point Cloud Compression

Because of the massive amount of PtCl to be streamed, state-of-the-art solutions take advantage of compression techniques. For instance, Google Draco [4] and PCL [21] are two tree-based processing libraries that can compress point clouds by orders of magnitude. Utilizing these libraries, Han *et al.* presented ViVo [9], a visibility-aware mobile volumetric streaming system that dynamically optimizes the fetching and rendering of volumetric data during AR experiences by incorporating visibility-aware optimizations Lee *et al.* introduced GROOT [18], a framework that employs PD-Tree (parallel decodable tree) to remove the data dependency in tree structure by using octree-breadth bytes and octree-depth bytes. Despite promising results, these works focus on pre-processing PtCl offline that has already been captured.

3 RESEARCH PROTOTYPE

To characterize the performance and energy budget for PtCl live-capturing and streaming on mobile devices, we set up a research prototype using three off-the-shelf components, consisting of a Microsoft Azure Kinect sensor [27] to capture PtCl at a high resolution and precision, an NVIDIA Jetson TX2 development board [29] to process PtCl as an edge-assistant, and a Google Pixel 3XL [6] to enable PtCl visualization and interaction with high user mobility. The research prototype is shown in Figure 2. Though built around those three components, this pipeline is generic to other components such as an Intel RealSense sensor (which also produces depth data at a high resolution and precision), other edge-computing modules (no matter faster or slower), and other mobile platforms such as iOS devices. From capturing to rendering, this prototype can be divided into three subcomponents, capturing and processing as the data producer, streaming as the data porter, and rendering as the data consumer. Operations in this pipeline are shown in Figure 3.

Capturing There are multiple ways to produce PtCl. For instance, Microsoft Azure Kinect equips a ToF sensor and Intel RealSense employs a dual-camera system to sense real-world points in millimeters from the camera center in 3D space, with their color values sensed by the RGB camera module. Although there are many public point cloud datasets [2, 8], we focus on live capture and streaming. In our prototype, an Azure Kinect sensor is directly connected to a Jetson TX2 through a USB interface. We use the Azure Kinect SDK to capture depth images and color images in the C++ environment through APIs provided by the Open3D library [42], which is a popular library for 3D data processing. The Azure Kinect captures color images in the MJPEG format and supports resolutions ranging from 720p to 4K. It captures depth images in the DEPTH16 format and supports various modes, e.g., NFOV or WFOV and unbinned (1024x1024) or binned (512x512). It supports a frame rate as high as 30 FPS.

Processing To mimic and explore PtCl applications in the edge-computing context, especially for future deployment under the commercial 5G environment, we use a less-powerful Jetson TX2 (CPU/GPU equivalent to mobile devices) with higher mobility to host and process data retrieved from the Kinect sensor. As shown in Figure 3, generating PtCl consists of decompressing the color image, converting the color image format, transforming depth image views, and stitching depth and color images. To convert MJPEG to RGB/YUV, we used two libraries, CPU-based libjpeg-turbo [20] and GPU-based nvJPEG [28]. Then, we use APIs provided by the Kinect SDK to transform the depth image to color image view and generate 3D depth data. To stitch depth and color data, we follow popular 3D file formats such as PLY, resulting in XYZ and RGB values for each point that are interpreted as bytes, concatenated, then stored in an `unsigned char` buffer. To encode PtCl data, we use Google Draco.

Streaming To send PtCl, we implement a C++ TCP server that transmits all the data through a `Socket`. On the receiving end, we implement a C# TCP client in the Unity app to retrieve data iteratively with an arbitrary buffer size of 8192, looping in the background. PtCl is transmitted from the Jetson TX2 to the Pixel 3XL through a local WiFi connection. Both devices support 802.11ac standard, with a throughput as high as 867Mbps.

Rendering We focus on rendering PtCl in AR apps that run on untethered mobile devices, such as a smartphone, a Google Glass [5], or a Microsoft HoloLens [26]. To conduct experiments, we develop an app using the Unity game engine [39], which is one of the most favorable platforms for AR developers and content creators [19]. In this Unity app, PtCl is first received as byte arrays, then interpreted into XYZ and RGB values, and finally fed into the main Unity thread for rendering. To evaluate the capability of today's mobile devices when rendering PtCl, we implemented three different methods. The first implementation naively renders PtCl in the Unity main (single) thread. The second implementation utilizes CPU multithreading to divide and process PtCl in chunks. For both CPU-based implementations, PtCl is drawn in meshes made from points, with the `MeshTopology.Points` API. The third implementation offloads all the data to the on-device GPU with the help of a `ComputeBuffer`.

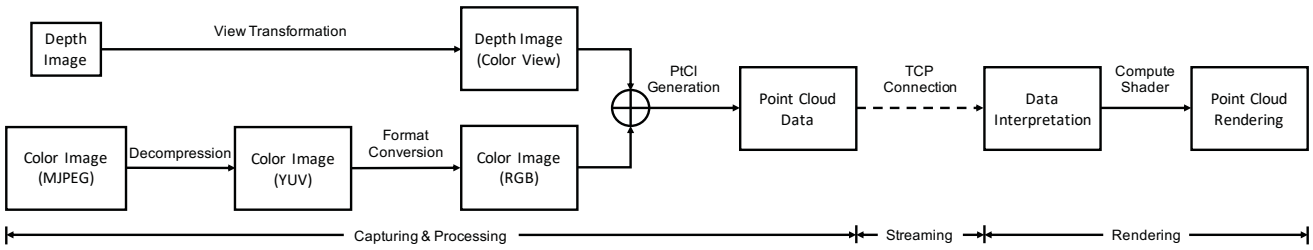


Figure 3: A sequence of operations performed and monitored in the PtCl capturing and streaming pipeline on our research prototype.

4 RESULTS

Evaluation workloads Using our research prototype, we capture the depth image at a frame rate of 15 FPS for two different resolutions, WFOV 512x512 (binned) and 1024x1024 (unbinned), which are represented in 2 bytes `int16_t`. Additionally, we capture the color image at a frame rate of 15 FPS for two different resolutions, 1280x720 (720p) and 1920x1080 (1080p). Then, we stream PtCl (720p-binned/unbinned and 1080p-binned/unbinned interpreted as image resolution-depth resolution) generated from the transformed depth and color image (to color image view) to the mobile devices. The amount of data captured and streamed per frame is shown in Table 2.

Evaluation metrics We care about app performance and energy efficiency. Thus, we monitor the *latency* when performing each operation within the PtCl capturing-to-rendering pipeline, such as converting MJPEG to RGB, manipulating buffers, computing shader, etc., as introduced in §3. Additionally, we monitor the *energy consumption per frame* (calculated as a product of power and frame rate) on both the Pixel 3XL client and the Jetson TX2 server. On Jetson TX2, power consumption is read from the power rail sys files, including SOC, DDR, CPU, GPU, etc. On Pixel 3XL, power consumption is measured by reading the voltage and current sys files of the battery. For all experiments, results are averaged across multiple samples as retrieved during a one-minute period when running the app.

4.1 PtCl Capturing & Processing

In this stage, we monitor four operations on Jetson TX2: *(i)* MJPEG decompression, *(ii)* format conversion, *(iii)* view transformation, and *(iv)* PtCl generation.

Decompressing MJPEG images at 720p with libjpeg-turbo takes 20 ms, followed by another 16 ms to convert BGRA to RGB. Meanwhile, with nvJPEG, decompressing MJPEG images takes 16 ms, followed by another 23 ms to convert YUV to RGB. Increasing the resolution to 1080p doubles the latency for both decompression and conversion for both approaches. On top of these two latency, transforming depth image to 720p and 1080p color image view takes 29 ms and 42 ms for unbinned depth image along with 27 ms and 41 ms for binned depth image. PtCl generation for 720p-binned/unbinned takes 15 ms; 33 ms for 1080p-binned/unbinned. Draco compression (compression level at 7 and quantization at 11 bits) takes 914 ms to encode 720p-binned/unbinned (600K points) PtCl and 2238 ms to encode 1080p-binned/unbinned (1.36M points) PtCl. The resulting output is one magnitude smaller in file size compared with the original PtCl containing both XYZ and RGB values.

In all experiments, we can identify a substantial improvement in energy efficiency if the system is working at a lower resolution. For instance, energy consumption per frame of the GPU is 0.89 J working at 720p-binned, which is 62% less compared against 2.37 J working at 720p-unbinned and 46% less compared against 1.64 J working at 1080p-binned. Note that, by reading the I/O power rail sys file, we find that the power draw from the Kinect sensor is stable regardless of working at different resolutions.

4.2 PtCl Server→Client Streaming

In this stage, we monitor two operations, the `Socket` send on Jetson TX2 and the `Socket` receive on Pixel 3XL.

The average latency to send the compressed PtCl on Jetson TX2 is 84 ms at 720p-binned, 91 ms at 720p-unbinned, 257 ms at 1080p-binned, and 274 ms at 1080p-unbinned. Accordingly, the average latency to receive the compressed PtCl on Pixel 3XL is 143 ms at 720p-binned, 146 ms at 720p-unbinned, 297 ms at 1080p-binned, and 303 ms at 1080p-unbinned. In both cases, streaming latency varies with a high deviation. If sending raw PtCl, the latency is one magnitude higher.

4.3 PtCl Rendering

Within the Unity app deployed on a Pixel 3XL, we monitor two operations, PtCl interpretation and rendering. Note that those two operations are based on the streaming of raw PtCl because of the high latency to compress it. As discussed in §3, the rendering is implemented with three different approaches. A comparison of the latency for each method is shown in Figure 4.

In the naive implementation, interpreting PtCl in the background TCP client thread takes 127 ms at 720p-binned/unbinned and 238 ms at 1080p-binned/unbinned, followed by 128 ms and 287 ms for data rendering in the Unity main thread. With the CPU multi-threaded implementation, spawning 14 threads (each thread responsible for one mesh object) results in a mere 5 ms processing time for each thread. However, spawning multiple threads incurs latency overhead. The overall latency is 87 ms and 185 ms to render PtCl at two resolutions accordingly. In the GPU `ComputeBuffer` implementation, rendering PtCl at two resolutions takes 80 ms and 180 ms accordingly. Though PtCl cannot be rendered at real-time on mobile devices, they can still be rendered on a good GPU machine with

Color Image		Depth Image		PtCl (without transformation)		PtCl (transformed to color image view)	
720p	1080p	Binned	Unbinned	720p + Binned	720p + Unbinned	720p + Un/Binned	1080p + Un/Binned
2,764,800	6,220,800	3,145,728	12,582,912	5,910,528	15,347,712	8,285,400	18,662,400

Table 2: The size of the data captured by the sensor and the generated PtCl data per frame (in # of bytes).

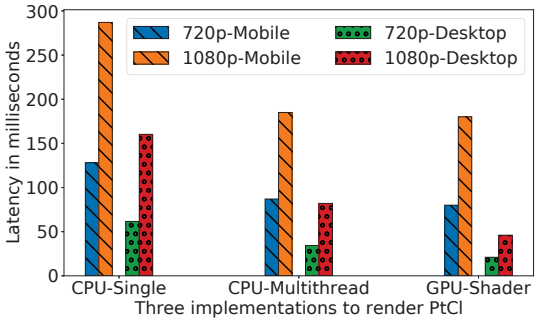


Figure 4: The latency to draw and render PtCl using three different approaches in the Unity app running on a Pixel 3XL and on a desktop.

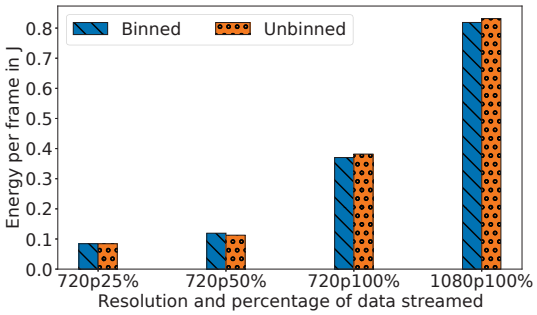


Figure 5: The system energy efficiency of interpreting and rendering PtCl on Pixel 3XL.

real-time performance. Also shown in Figure 4, a laptop with a RTX-3070 GPU can render PtCl captured at 720p-binned/unbinned in as low as 21 ms and 1080p-binned/unbinned PtCl in 46 ms.

Figure 5 shows the energy efficiency result. Interpreting and rendering PtCl consumes 0.37 J and 0.81 J per frame captured at 720p-binned/unbinned and 1080p-binned/unbinned accordingly.

4.4 Summary

- **Latency and energy consumed** in each stage of the capturing-to-rendering pipeline is **proportional to the PtCl resolution**; thus, presenting an opportunity to adaptively balance resolution-based tradeoffs.
- **High resolution XYZ representations are more expensive than raw depth data.** Transforming raw depth data into 3D XYZ format (four bytes each) and further into the color image's view makes PtCl data magnitudes larger. Decoupling the processing of color image and depth image is needed. Furthermore, determining the resolution of PtCl can be driven by the consumer, e.g., performing format optimization based on the requirements of the rendering component, such as particles.
- **Encoding PtCl captured at a high resolution to tree structure with libraries like Draco on an edge device is not feasible for real-time performance (at least 30 FPS).** In our experiment, live-encoding with state-of-the-art libraries takes seconds to compress raw PtCl captured at a high resolution (more than 1M points). However, if solely working on spatial data or reduced amount of PtCl, then encoding can still be utilized. In addition, an obvious compressing and networking latency tradeoff can be identified between streaming raw PtCl and compressed PtCl, upon which the overall throughput in our current prototype is determined.
- **Network would not be the bottleneck** to stream PtCl. In particular, one to two magnitudes higher throughput (i.e., the streaming latency will decrease by two magnitudes) brought by the upcoming 5G technology coupled with advanced streaming protocols such as DASH can transmit PtCl in real-time [11, 25, 36].
- **Current mobile devices cannot handle unfiltered raw PtCl** even with CPU multithreading and GPU acceleration, not to mention the high power consumption associated which will drain the battery on mobile devices very quickly.

4.5 Capturing & Streaming fewer Points

In another set of experiments, we capture, process, stream, and render only 25% and 50% of the original PtCl, to verify that less PtCl leads to less processing latency and higher energy efficiency. The latency reduction and energy efficiency improvement are substantial across all components, proportional to the percentage of data being processed. For example, even in the single-thread implementation, rendering 25% of the 720p-binned/unbinned PtCl on Pixel 3XL can take as low as 40 ms with a 0.084 J energy consumption per frame. As another example, on Jetson TX2, when capturing and sending 25% and 50% of the 720p-binned/unbinned PtCl, the transmission latency can be less than 1 ms after TCP connection is setup due to the window size negotiation.

5 CONCLUSION

In this work, we built a research prototype with three off-the-shelf components (an Azure Kinect sensor, a Jetson TX2 board, and a Pixel 3XL smartphone) to study the bottlenecks in real-time dense point cloud capturing and streaming on mobile devices. The findings around throughput limitation and data representation motivate the development of a software system stack to realize user-defined dense point cloud sensing on mobile devices.

REFERENCES

[1] Adobe. 2021. High-quality, network-efficient HTTP streaming. https://www.adobe.com/products/hds-dynamic-streaming.html.

[2] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 2016. 3D Semantic Parsing of Large-Scale Indoor Spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1534–1543. https://doi.org/10.1109/CVPR.2016.170

[3] Cyrus S. Bamji, Swati Mehta, Barry Thompson, Tamer Elkhatib, Stefan Wurster, Onur Akkaya, Andrew Payne, John Godbaz, Mike Fenton, Vijay Rajasekaran, Larry Prather, Satya Nagaraja, Vishali Mogallapu, Dane Snow, Rich McCauley, Mustansir Mukadam, Iskender Agi, Shaun McCarthy, Zhanping Xu, Travis Perry, William Qian, Vei-Han Chan, Prabhu Adepu, Gazi Ali, Muneeb Ahmed, Aditya Mukherjee, Sheethal Nayak, Dave Gampell, Sunil Acharya, Lou Kordus, and Pat O'Connor. 2018. IMpixel 65nm BSI 320MHz demodulated TOF Image sensor with 3um global shutter pixels and analog binning. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. 94–96. https://doi.org/10.1109/ISSCC.2018.8310200

[4] Google. 2021. Draco. https://github.com/google/draco.

[5] Google. 2021. Glass. https://www.google.com/glass/start/.

[6] GSMARENA. 2021. Google Pixel 3 XL. https://www.gsmarena.com/google_pixel_3_xl-9257.php.

[7] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2014. Towards Wearable Cognitive Assistance. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '14)*. ACM.

[8] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan Dirk Wegner, Konrad Schindler, and Marc Pollefeys. 2017. Semantic3D.net: A new Large-scale Point Cloud Classification Benchmark. *CoRR* abs/1704.03847 (2017). arXiv:1704.03847 http://arxiv.org/abs/1704.03847

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

[10] Lei Han, Tian Zheng, Yinheng Zhu, Lan Xu, and Lu Fang. 2020. Live Semantic 3D Perception for Immersive Augmented Reality. *IEEE Transactions on Visualization and Computer Graphics* 26, 5 (2020), 2012–2022. https://doi.org/10.1109/TVCG.2020.2973477

[11] Chris Hoffman and Craig Lloyd. 2018. The Best (Actually Useful) Tech We Saw at CES 2018. https://www.howtogeek.com/339206/the-best-actually-useful-tech-we-saw-at-ces-2018/.

[12] Mohammad Hosseini and Christian Timmerer. 2018. Dynamic Adaptive Point Cloud Streaming. *Proceedings of the 23rd Packet Video Workshop* (Jun 2018). https://doi.org/10.1145/3210424.3210429

[13] Jinhan Hu, Alexander Shearer, Saranya Rajagopalan, and Robert LiKamWa. 2019. Banner: An Image Sensor Reconfiguration Framework for Seamless Resolution-Based Tradeoffs. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services* (Seoul, Republic of Korea) *(MobiSys '19)*. Association for Computing Machinery, New York, NY, USA, 236–248. https://doi.org/10.1145/3307334.3326092

[14] Jinhan Hu, Jianan Yang, Vraj Delhivala, and Robert LiKamWa. 2018. Characterizing the Reconfiguration Latency of Image Sensor Resolution on Android Devices. In *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications* (Tempe, Arizona, USA) *(HotMobile '18)*. Association for Computing Machinery, New York, NY, USA, 81–86. https://doi.org/10.1145/3177102.3177109

[15] Intel. 2021. RealSense Depth Camera D435i. https://www.intelrealsense.com/depth-camera-d435i/.

[16] James Carroll. 2020. Time of Flight sensors target high-speed 3D machine vision tasks. https://www.vision-systems.com/cameras-accessories/article/14175785/3d-machine-vision-time-of-flight-tof-sensors-and-high-speed-tasks.

[17] Stefan Lederer, Christopher Müller, and Christian Timmerer. 2012. Dynamic Adaptive Streaming over HTTP Dataset *(MMSys '12)*. Association for Computing Machinery, New York, NY, USA, 89–94. https://doi.org/10.1145/2155555.2155570

[18] Kyungjin Lee, Juheon Yi, Youngki Lee, Sunghyun Choi, and Young Min Kim. 2020. GROOT: A Real-Time Streaming System of High-Fidelity Volumetric Videos *(MobiCom '20)*. Association for Computing Machinery, New York, NY, USA, Article 57, 14 pages. https://doi.org/10.1145/3372224.3419214

[19] Germán Leiva, Cuong Nguyen, Rubaiat Habib Kazi, and Paul Asente. 2020. Pronto: Rapid Augmented Reality Video Prototyping Using Sketches and Enaction. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376160

[20] libjpeg-turbo. 2021. libjpeg-turbo. https://libjpeg-turbo.org/.

[21] Point Cloud Library. 2021. Point Cloud Library. https://pointclouds.org/.

[22] R. LiKamWa, J. Hu, V. Kodukula, and Y. Liu. 2021. Adaptive Resolution-Based Tradeoffs for Energy-Efficient Visual Computing Systems. *IEEE Pervasive Computing* (2021), 1–9. https://doi.org/10.1109/MPRV.2021.3052528

[23] Robert Likamwa, Jinhan Hu, Venkatesh Kodukula, and Yifei Liu. 2021. Adaptive Resolution-Based Tradeoffs for Energy-Efficient Visual Computing Systems. *IEEE Pervasive Computing* 20, 2 (2021), 18–26. https://doi.org/10.1109/MPRV.2021.3052528

[24] Robert LiKamWa, Bodhi Priyantha, Matthai Philipose, Lin Zhong, and Paramvir Bahl. 2013. Energy Characterization and Optimization of Image Sensing Toward Continuous Mobile Vision. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '13)*. ACM. https://doi.org/10.1145/2462456.2464448

[25] Zhi Liu, Qiyue Li, Xianfu Chen, Celimuge Wu, susumu ishihara, Jie Li, and Yusheng Ji. 2020. Point Cloud Video Streaming in 5G Systems and Beyond: Challenges and Solutions. https://doi.org/10.36227/techrxiv.13138940.v1

[26] Microsoft. 2021. HoloLens 2. https://www.microsoft.com/en-us/hololens.

[27] Microsoft Azure. 2021. Azure Kinect DK. https://azure.microsoft.com/en-us/services/kinect-dk/.

[28] NVDIA DEVELOPER. 2021. nvJPEG Libraries. https://developer.nvidia.com/nvjpeg.

[29] NVIDIA DEVELOPER. 2021. Jetson TX2 Module. https://developer.nvidia.com/embedded/jetson-tx2.

[30] OmniVision. 2021. OV9282. https://www.ovt.com/sensors/OV9282.

[31] Fabio Poiesi, Alex Locher, Paul Chippendale, Erica Nocerino, Fabio Remondino, and Luc Van Gool. 2017. Cloud-Based Collaborative 3D Reconstruction Using Smartphones. In *Proceedings of the 14th European Conference on Visual Media Production (CVMP 2017)* (London, United Kingdom) *(CVMP 2017)*. Association for Computing Machinery, New York, NY, USA, Article 1, 9 pages. https://doi.org/10.1145/3150165.3150166

[32] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2016. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *CoRR* abs/1612.00593 (2016). arXiv:1612.00593 http://arxiv.org/abs/1612.00593

[33] Feng Qian, Bo Han, Jarrell Pair, and Vijay Gopalakrishnan. 2019. Toward Practical Volumetric Video Streaming on Commodity Smartphones. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, USA) *(HotMobile '19)*. Association for Computing Machinery, New York, NY, USA, 135–140. https://doi.org/10.1145/3301293.3302358

[34] Hang Qiu, Fawad Ahmad, Fan Bai, Marco Gruteser, and Ramesh Govindan. 2018. AVR: Augmented Vehicular Reality. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services* (Munich, Germany) *(MobiSys '18)*. Association for Computing Machinery, New York, NY, USA, 81–95. https://doi.org/10.1145/3210240.3210319

[35] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. 2017. OctNet: Learning Deep 3D Representations at High Resolutions. arXiv:1611.05009 [cs.CV]

[36] Traci Ruether. 2021. The Impact of 5G on Streaming. https://www.wowza.com/blog/the-impact-of-5g-on-streaming.

[37] Sabbir Rangwala. 2020. The iPhone 12 - LiDAR At Your Fingertips. https://www.forbes.com/sites/sabbirrangwala/2020/11/12/the-iphone-12lidar-at-your-fingertips.

[38] Shishir Subramanyam, Irene Viola, Alan Hanjalic, and Pablo Cesar. 2020. User Centered Adaptive Streaming of Dynamic Point Clouds with Low Complexity Tiling. In *Proceedings of the 28th ACM International Conference on Multimedia* (Seattle, WA, USA) *(MM '20)*. Association for Computing Machinery, New York, NY, USA, 3669–3677. https://doi.org/10.1145/3394171.3413535

[39] Unity. 2021. The leading platform for creating interactive, real-time content. https://unity.com/.

[40] Tai Wang, Xinge Zhu, and Dahua Lin. 2020. Reconfigurable Voxels: A New Representation for LiDAR-Based Point Clouds. arXiv:2004.02724 [cs.CV]

[41] Jun Yi, Md Reazul Islam, Shivang Aggarwal, Dimitrios Koutsonikolas, Y. Charlie Hu, and Zhisheng Yan. 2020. An Analysis of Delay in Live 360° Video Streaming Systems. In *Proceedings of the 28th ACM International Conference on Multimedia* (Seattle, WA, USA) *(MM '20)*. Association for Computing Machinery, New York, NY, USA, 982–990. https://doi.org/10.1145/3394171.3413539

[42] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. 2018. Open3D: A Modern Library for 3D Data Processing. *arXiv:1801.09847* (2018).

[43] Yin Zhou and Oncel Tuzel. 2018. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4490–4499. https://doi.org/10.1109/CVPR.2018.00472