Adaptive Volumetric Streaming: Optimizing and Measuring a Multi-Camera Streaming Pipeline

Kevin Cui
Cornell University
Ithaca, New York
kc734@cornell.edu

Jiqing Wen
METEOR Studio
Tempe, Arizona
jwen31@asu.edu

Robert LiKamWa
METEOR Studio
Tempe, Arizona
likamwa@asu.edu

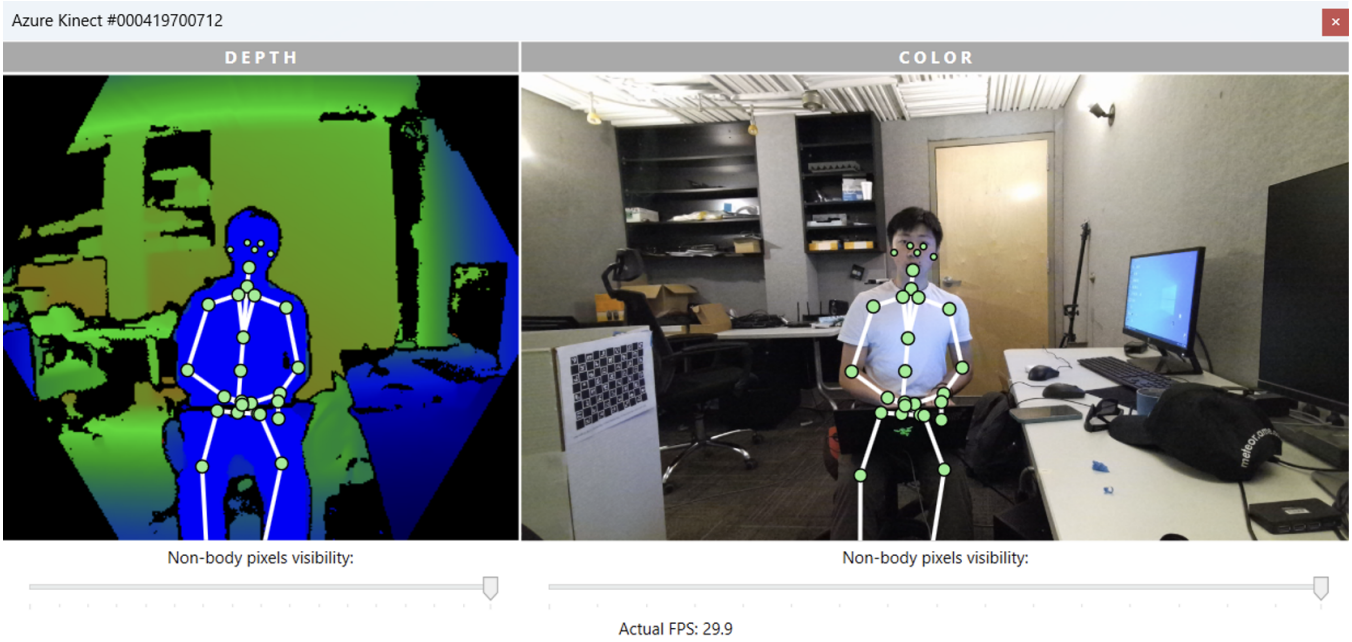

Figure 1: Azure Kinect camera body tracking, depth and color view, 2023.

ABSTRACT

Volumetric streaming, unlike traditional 2-dimensional video, enables users 6 degrees of freedom to explore a scene, providing myriad virtual reality applications in telecommunications, healthcare, and education. The increased viewport mobility, however, comes at the cost of wide bandwidth requirements and high compute power expenses. Therefore to make virtual and augmented reality applications more accessible for public consumption, it is imperative reduce the memory and energy needs of volumetric streaming.

As such, we implement a volumetric streaming pipeline and investigate optimizations through body tracking video region-of-interest filters. More concretely, we integrate two traditional compression techniques, temporal and spatial downsampling, and study their effects on throughput, latency, and quality of experience measurements of a streaming system. Evaluating our pipeline, we find that temporal downsampling decreases data consumption and spatial downsampling decreases execution time. While we found limitations with end-to-end latency, the success of pose-based adaptive filters implies the feasibility of a volumetric streaming system generating point cloud visualizations in real-time. This research paves the way for more extensive study of volumetric streaming and 3-dimensional compression techniques.

KEYWORDS

volumetric streaming, augmented reality, image processing, metrics

1 INTRODUCTION

1.1 Motivation

With developments of Oculus Quest over the past several years and the most recent announcement of the Apple Vision Pro, virtual reality (VR) and augmented reality (AR) technology is fresh in the public mind. Given its novelty, there are many possible applications for VR/AR, including gaming, healthcare, and education. The Augmented Coach project, designed by Arizona State University's METEOR Studio, is one such application, focusing on athletic performance analytics. More specifically, Augmented Coach is a sports training platform that allows players to receive real-time feedback during exercises from coaches who can remotely view and annotate their body movements in a 3D space [13]. The 360 degree viewpoint and low latency allow for on-the-fly feedback, which can not only expedite athletic improvement but also make coaching more accessible to users in remote areas. The latter point

of increasing equitability is a key motivation in the continued development of Augmented Coach, and specifically the optimization of its underlying streaming framework.

Like other VR streaming applications, Augmented Coach is built upon volumetric video capture or volumetric streaming. As Bo Han et al. describes, volumetric video is "truly 3D, allowing six degrees of freedom (6DoF) movement for their viewers" [2]. Consequently however, volumetric streaming requires an extremely high bit rate as voxel chunks must be continuously loaded from a 3-dimensional scene. Part of the challenge comes from stitching together depth and color video streams from multiple cameras to create point clouds for an immersive view; simply transmitting high-quality video streams is half the battle, as Matroska (.mkv) files, which are commonly used for combined color and depth streams, are almost cartoonishly large. Indeed, a single stream from an Azure Kinect camera recorded for one minute is upwards of 1-2 MB.

Thus, on a network with sufficiently wide bandwidth and a computer with sufficiently high processing power, volumetric video becomes feasible. If the streams are transformed into point cloud renderings before they are transmitted over a server, the data load reduces further. Employing such ideas, there has been significant progress in VR streaming technology in the past decade (e.g. VRChat). Still in the case of Augmented Coach, continuously live-streaming high resolution video is a focal point yet to be solved efficiently, so optimizing the volumetric streaming pipeline is essential. Ultimately, our desire is to make VR/AR technologies a readily accessible tool for developers and end-users. Hence, reducing the computational expense of volumetric streaming is vital.

1.2 Our contributions

Our research explores different techniques for optimizing the volumetric streaming pipeline. The methodology builds off an established system that connects Azure Kinect cameras to a Unity client via WebRTC connection. In the Unity client, a separate rendering pipeline is employed to generate point cloud representations from color and depth input. Having this system already in place, the objectives and associated achievements with this research relate to server-side optimizations of an encoded video stream. Notably, temporal downsampling, and spatial downsampling are integrated into the system, using adaptive body tracking filters as a framework and the work of Kodukula et al. [3] as a springboard.

Moreover, we measure the effectiveness of these improvements using relevant metrics. For each added feature, and combination with other features, we test the throughput, latency, and quality of experience (QoE). Since the pipeline already had associated memory and computational requirements, metric measurement was focused on the efficiency of new improvements. However, end-to-end latency and upload bandwidth were still used, to a lesser extent, in understanding the benefit of the implemented features.

In summary, our systems research makes several contributions:

- First, we develop two main strategies for improving a Azure Kinect to Unity volumetric streaming system, based loosely on the work of Kodukula et. al [3] and image compression principles. These strategies are temporal downsampling and spatial downsampling, both framed by adaptive body tracking filters.
- Second, we establish the feasibility of such a system for use in VR technology and widespread consumption. Using throughput, latency, and QoE measurements, we quantitatively determine effects of the strategies and realistically feasibility to VR/AR integration.

2 BACKGROUND/RELATED WORK

Volumetric Streaming. Volumetric video is a relatively new form of multimedia, with 3D, 6DoF video being notably different from traditional formats. With the immersive experience, however, there is a significant requirement for bit-rate, and ViVo [2], a visibility aware VV streaming system for mobile systems, and CaV3 [5], a cache assisted viewport adaptive system, are proposed efficient VV capture pipelines. Kodukula et al. [3] also suggests compression techniques using multi-resolution systems that enhance or downgrade certain video regions. Further, Liu and Zhong et al. [6] explored a remote rendering approach towards VR streaming. Furion [4] employs a similar split-rendering technique to optimize VR streaming. These are mostly theoretical studies, however, and this research will tackle the implementation of a volumetric streaming system and proposed optimizations [12].

Augmented Coach is an AR sports coaching platform developed by the Meteor Studio at Arizona State University. Its novelty lies in the ability for coaches to analyze athletes in a 3D augmented space, allowing for better spatiotemporal contextualization of athletic movement compared to a traditional 2D representation [13]. Channar, Dbeis, and Richards [1] conducted user studies on usability of the framework. Relating to our research, our ultimate aim is to connect a streaming pipeline to Augmented Coach for systems testing of live-streaming capabilities.

Viewport Prediction. A key technique for seamless VR/AR experience involves buffering the video chunks that the user will view next. ViVo [2] tries linear regression and multilayer perceptron approaches towards predicting future user viewports. The CaV3 architecture [5] builds upon this using gaze and current user inertia as features for a transformer-based approach of viewport prediction. Further, research on trajectory-based prediction [10] and cluster-based prediction [9] are suggested means to accurately predict user viewports.

Data compression and more specifically video compression is important in reducing the throughput of media streams. One of the most prevalent techniques is spatial compression [**?**]. In this, a static image is divided into chunks that are individually transformed using algorithms like the Discrete Cosine Transform (DCT) or wavelet transform. Another technique is temporal compression, in which consecutive frames are analyzed for change [**?**]. If a region is static, instructions are sent to maintain the current output, thereby reducing the total quantity of data needing to be transmitted. There are many other compression techniques including Joint Photographic Experts Group (JPEG), Motion JPEG, VP8, and H.264 which are all algorithmically designed [**?**]. There has also been recent work by Google on transformer-based video compression [7], and this research takes inspiration from all compression techniques in developing parallel compression techniques for volumetric video.

Microsoft Azure Kinect. Finding sensible cameras are a crucial part in developing a streaming pipeline. Microsoft's Azure Kinect

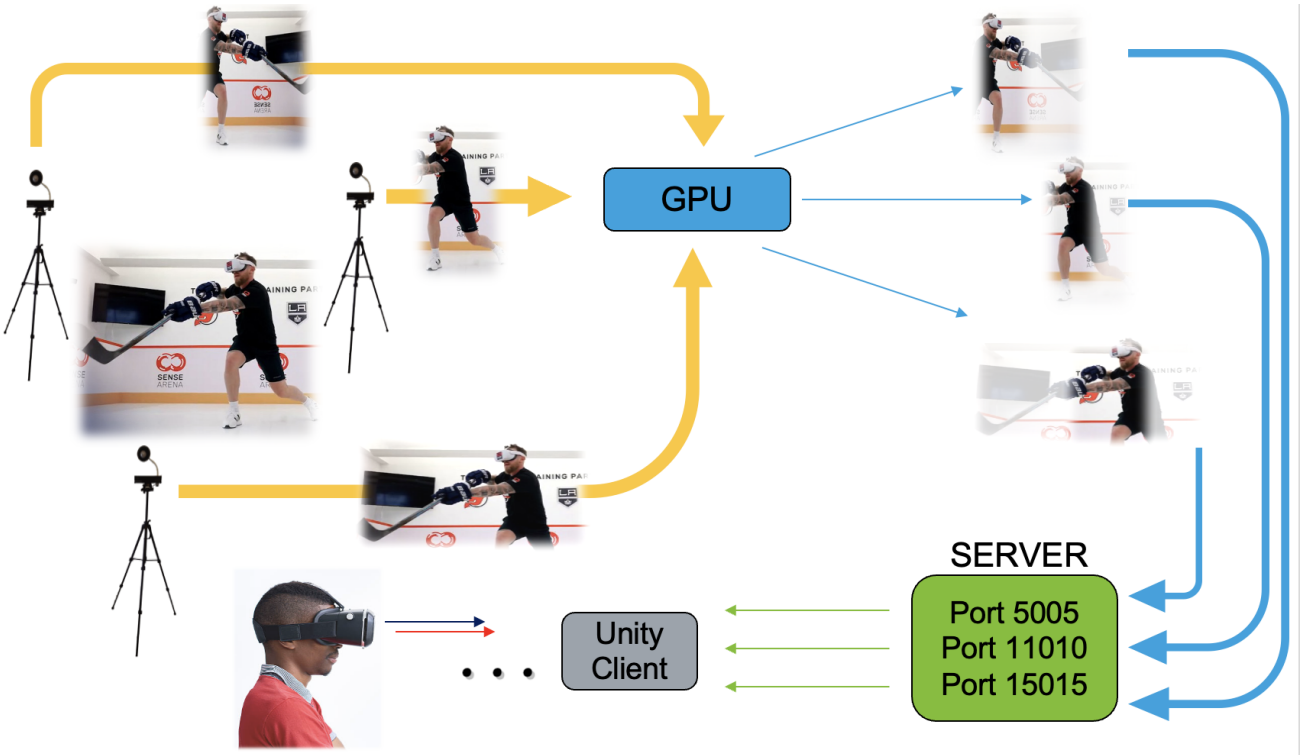

Figure 2: Volumetric streaming pipeline, end-to-end from camera to end-client

is a relatively robust piece of hardware that can simultaneously capture depth and color streams at variable resolutions [8, 11]. Using time-of-flight (ToF) sensors, the depth camera can record in 4 field-of-view (FOV) modes: NFOV unbinned (640x576), NFOV 2x2 binned (320x288), WFOV 2x2 binned (512x512), and WFOV unbinned (1024x1024) [11]. The RGB camera, recording in MJPEG, YUY2, or NV12 formats, similarly has many resolution modes, from 1280x720 (16:9 aspect ratio) to 4096x3072 (4:3 aspect ratio) [11]. The wide array of features enabled preliminary spec testing for our research. As of this paper's submission, metrics relating to these measures have yet to be thoroughly explored.

3 METHODOLOGY

3.1 Systems overview

The pipeline can be broken down into a sequence of processes as seen in Figure 2. Before any camera data can be transmitted, the server and client must be initialized. Then, output from Azure Kinect cameras can be streamed through an image processing pipeline that applies a selection of our implemented downsampling techniques and encodes video data to be sent to a remote server. Once the client is connected to the server, the encoded data is piped into Unity where it is interpreted as color and depth textures. These rendered textures are and can be fed into other Unity pipelines, including a point cloud renderer, and the output is ultimately displayed to an end-user wearing a headset or directly monitoring the Unity client.

3.2 Pipeline implementation

Having now a high-level overview of the volumetric streaming pipeline, let us go into the specifics how each process is implemented and how they function. We start with step zero: initializing the server and client network. We implemented Flask servers in Python that are listening to different local ports, and before starting the camera stream, we spin them up and have these servers online and ready to receive and transmit data. In our implementation, we have three Flask servers running concurrently (one for each camera), and they are connected to local ports 5005, 11010, and 15015. Simultaneous to our initialization of the servers, we start up the Unity project and register the client to listen for requests on any of the aforementioned ports. The Unity to server connection is via WebRTC and Unity's built-in streaming capabilities. It is important to note that the client does not have to be on the same machine as the server, but simply on the same broadband network with a discovereable IP.

Once the network connection is established, evident with client registration POST requests, we can commence with Azure Kinect video streaming. This process is fairly straightforward as we simply call an executable that toggles Kinect camera parameters (like FoV dimensions, frame rate, color image format, connecting port, etc.) and image compression options. This executable also allows for .mkv video playback, in addition to Kinect streaming, which can be helpful if there are hardware access limitations.

When the executable launches the camera, it immediately connects to the indicated port (either 5005, 11010, or 15015), using

SipSorcery. If no image processing or downsampling specifications are selected, the color and depth video streams, encoded with the VP8 protocol, are sent directly to the server. If, however, spatial or temporal downsampling with an adaptive filter is chosen, then the system's attached GPU will use runtimes from the Azure Kinect Body Tracking SDK to pre-process the image before encoding. It is at this step, transmission between GPU to server, that the bulk of our optimization research is implemented. Additionally, we would like to note that the output stream is completely wireless from this point onward, and we only need connected Kinect cameras to upload to a streaming client.

Finally, after the compressed video data is streamed through the specified port and available on the Flask server, a GET request from Unity is sent, allowing for the data to come through and be processed by additional pipelines in Unity for end-user consumption. This final step of connection is triggered by specific key bindings, and given the capacity for multiple servers for multi-camera streaming, there is one event per camera stream. It is all handled by WebRTC. For a summary of the specifications used in the pipeline, see Table 1.

The system as a whole can be thought of as parallel pipelines for each individual Azure Kinect camera. It is all handled by the same web transport and protocol, but there is a unique port, server, and receiving client for each video stream. For these reasons, we found it helpful to connect each Kinect server-client pipeline individually instead of progressing concurrently step-wise for a multi-camera streaming system. Given its complexity, however, the resulting color/depth stream and point cloud rendering output in Unity is simple.

Table 1: Primary specifications of pipeline components (determined via preliminary testing though other options could be tried)

Component	Specification
Camera(s)	Azure Kinect
Depth resolution	320 x 288
Color resolution	1280 x 720
Color image format	ColorBGRA32
Frame rate	30 fps
Web Transport	SIPSorcery, WebRTC
Connection port(s)	localhost:5005 (primary), 11010, 15015
GPU	nVidia GeForce RTX 3060 Ti
Server type	Flask
Client	Unity (v2021.3.25f1)

3.3 Video compression techniques

Transmitting the video stream directly from the Kinect cameras to a server is technically possible but not a good idea due to the high bit rate required for such transport. Therefore, we integrated several video compression techniques based on traditional 2D principles. Essentially, we identified regions of interest (ROI) that maintained continuous high-resolution throughput and then down-sampled non-ROI pixel regions. There were two main approaches to down-sampling: temporal and spatial. Regardless of the specific down-sampling technique, however, the manipulations were framed by adaptive filters, with three possibilities in total.

One was a skeleton tracker provided by the Azure Kinect Body Tracking SDK that drew bounding boxes around all regions in a frame containing a human body. The other main filter was a head tracker we developed based on joint data (specifically shoulder points, nose, and chest) supplemented by the Body Tracking SDK that drew bounding boxes all supposed "heads" in a region. For use as a control, we also developed a "none" filter that treate the entire frame as a region of interest, and therefore did not perform any downsampling. While we did preliminary testing of the impact of skeleton, head, and none filters on streaming performance, we primarily used them to contextualize the temporal and spatial down-sampling effects.

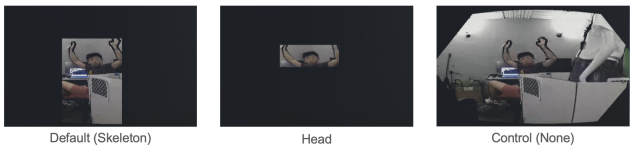

Figure 3: Comparison of different framing with adaptive filters, including the default skeleton model provided by the Azure Kinect Body Tracking SDK, a head tracking model, and control "None" filter model.

Temporal downsampling compresses video by *reducing the frequency* of data transmission. After applying an adaptive filter to generate regions of interest in the video frame, we can use temporal downsampling to change the refresh rate of the background. Instead of updating the video at the same frequency as the ROIs, using a variable background refresh rate enables reliable data compression. In this research, we toggled the temporal downsampling from 1 frame per second (fps) to 30 fps, or as fast as the pixel ROI was being updated. The actual implementation of temporal downsampling was through a systems timer that executed a background refresh at a selected frequency.

Figure 4: Temporal downsampling background refresh rate comparison of 1 fps vs. 30 fps updating; increased temporal downsampling or a lower refresh rate increases the visual choppiness of playback.

Spatial downsampling works by *reducing the complexity*of certain pixel regions; essentially, it is acting like a background blur. Similar

to temporal downsampling, we applied spatial downsampling only to non-ROIs in a video frame. However, unlike temporal downsampling, with spatial downsampling, data is being transmitted at the same frequency, merely at a lower resolution. We implemented spatial downsampling using a non-sliding NxN box blur algorithm that used the average of box-based background pixels to create a lower-resolution, and therefore more compressible, video stream. To ensure the dropoff in user experience was not too noticeable, our system only allows up to 4x4 pixel regions for spatial downsampling, although it would have been possible to extend this into higher dimensions.

Figure 5: Spatial downsampling background blur comparison of 1x1, 4x4, and 16x16 adaptations; expectedly, higher spatial downsampling (increased background blur blocks) diminishes the image resolution.

3.4 Metrics

There are several metrics relevant to the measurement of our streaming pipeline. Firstly, we care about the memory usage or footprint of the video data. Consequently, we have a bandwidth or, more accurately, **throughput** metric. This is all contextualized by execution time, so we define throughput as simply data processed per unit time. Our findings primarily concern the throughput of individual processes, most notably the GPU to server data transmission because that was what our improvements were targeting, but we also observed upload throughput data, or more aptly upload bandwidth requirements, as a sanity check.

Another key metric we use for studying the pipeline is **latency**. Latency relates to execution speed, and it is more specifically as the lag time between when an instruction is sent and when it actually finishes executing. Similar to throughput, we can measure latency of individual tasks, which we focus on in the systems paper, but we can also measure end-to-end latency of the full pipeline. Both approaches are helpful in quantifying the impact of our improvements.

One final metric that we consider is **quality of experience (QoE)**. This is fairly subjective, and we define it as a "what looks good" or "what is acceptable" statistic. Without thorough user study, QoE was not heavily used; rather it was simply suggested as another dimension to supplement our quantitative results.

4 EXPERIMENTAL RESULTS

In testing the system, combinations of one or multi-camera streaming systems were set up. Specifically, temporal downsampling was tested from 1 fps refresh rate up to 30 fps, with 5 fps increments; spatial downsampling was tested from 1x1 box blur up to 4x4, with 2x2 as the only intermediary. These combinations of refresh rate and blur factor were paired with the three adaptive filters: skeleton, head, and none (with none serving as a control for no data compression). Following are the main takeaways from systems measurements - corresponding figures are provided in the appendix.

4.1 Temporal downsampling decreases throughput

We found that an increase in temporal downsampling decreased the average throughput for GPU-server data data transmission. This trend is evident with both single Kinect and multi-camera streaming. Additionally, both novel adaptive filters (skeleton and head) had similarly trends. As expected, the control filter, which treated the entire frame as a region of interest, and therefore did not employ temporal downsampling, showed no trend with change in refresh rate. Considering the effects on latency, we observed little to no trend in that regard with changes in temporal downsampling.

This result is expected and promising; with temporal downsampling, data should be sent at a lower frequency, and therefore average throughput should decrease. The GPU is still transmitting the same quality of data but less times per second, so we are effectively flattening the curve by spacing out encoded data transmission. This in turn allows for a slower processor as it has more time in between update tasks to execute image processing and encoding.

4.2 Spatial downsampling decreases latency

With spatial downsampling, we observed noticeable trends with latency; specifically there was a negative relationship between spatial downsampling and encoding task latency. Put another way, this means that as the background blur is increased, the time it takes to an encoded video stream decreases. Like temporal downsampling, the trends for single Kinect and multi-camera streaming were quite similar, and again the skeleton and head tracking filters showed correlation, unlike the none filter. More quantitatively, there was nearly a 2 times reduction in latency from 1x to 4x blur while minimal change in QoE.

These results confirm our belief that decreasing the background complexity lends for faster data transmission. With spatial downsampling, there is no longer need to iterate through each pixel in encoding, but rather compression can be completed chunk-wise. It is interesting to note, however, there was little significance measured between spatial downsampling and encoded data throughput, which would have been expected. Still, the general notion of latency decreasing with spatial downsampling has promising implications.

4.3 Volumetric streaming pipeline has feasibility

Considering the system as a whole, measurements of individual parts do not necessarily imply the feasibility of success. Thus, we considered a notable bottleneck (upload bandwidth) and full system

experience (end-to-end latency) in evaluating our system. Very unexpectedly, there was almost no trend in upload throughput with respect to temporal downsampling, spatial downsampling, and our adaptive filter frameworks. This is not all bad news, however, as the overall upload bandwidth requirement for 3 simultaneous Kinect streams was under 1 Mbps, a quite reasonable figure.

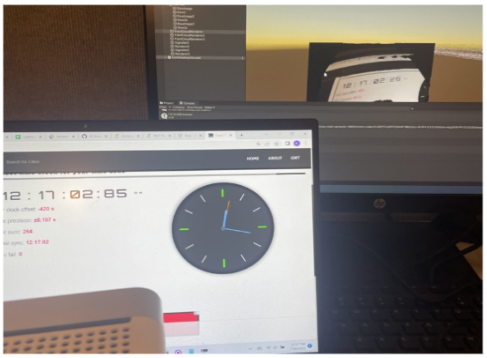

Figure 6: End-to-end latency measured using millisecond clock output for multi-camera Kinect streaming; latency is roughly 600 ms.

Despite the memory benchmark satisfaction, there were still concerns with latency that were observed. Streaming with a single camera to a remote client had upwards of 200ms of end-to-end latency. Using multiple cameras, we measured average end-to-end latency to be almost threefold at around 550ms. Given patency of latency greater than 200ms, the tri-Kinect streaming is not up to commercial standards. Further, streaming using 640x576 depth resolution or greater led to even more noticeable lag.

5 DISCUSSION

5.1 Key Takeaways

The results were mostly as expected, with temporal and spatial downsampling optimizing the volumetric streaming pipeline in different ways. Summarizing the ideas, we found:

- Temporal downsampling decreases throughput, with upload bandwidth is maintained at a reasonable sub-1 Mbps rate.
- Spatial downsampling decreases latency, and end-to-end latency increases at a rate of about 200ms per additional Kinect camera.

Beyond these two main concepts, we surmise that the adaptive filters are a viable tool for ROI framing. Piggybacking off the built-in Azure Kinect Body Tracking SDK, we have proven the possibility of creating other relevant bounding box procedures that can be applied with background downsampling and compression techniques to reduce data transmission.

5.2 Future Directions

Building on this work, there are many avenues for further development. First, resolving the high end-to-end latency issue is a top priority in making volumetric streaming more feasible. With further optimizations and additional novel techniques, it might also be possible to reduce the upload bandwidth at a controllable rate. Even without new techniques, progress can be made via deeper exploration of temporal and spatial downsampling and more comprehensive metrics.

Another direction for further research involves modularizing the volumetric streaming pipeline. Before starting on any developed, we faced many challenges in integrating the current system onto a new machine. A systematized manual for setup as well as decoupling of pipeline processes will enable expedited progress with volumetric streaming.

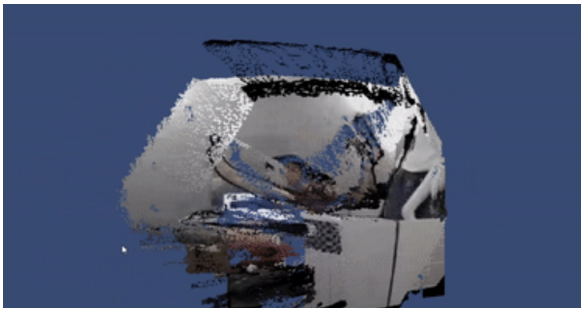

Figure 7: Proof-of-concept output in Unity from point cloud rendering of single Kinect camera streaming.

One final area for future work centers around connecting the system to data visualization pipelines, like point cloud renderers. The ultimate goal of having a volumetric streaming system is to serve as a backbone for VR/AR applications, and now that a proof-of-concept is working with live video feed, stitching together an actual volumetric video is next. We started briefly on this work, and a single camera point cloud stream has been generated, but more work must be done to calibrate multi-camera point cloud renderings in a virtual scene for use in projects like Augmented Coach. modularize pipeline

5.3 Conclusion

Volumetric streaming is a crucial yet challenging aspect of virtual reality and augmented reality technologies. We implement a volumetric streaming pipeline and demonstrate several downsampling optimizations to reduce the data transmission and time constraints of this new media format. With this work, we open the door to future measurement and optimization of volumetric streaming. Given its applicability with Augmented Coach and a new wave of technology, our keystone work will hopefully bring about further study and a brigther future with VR/AR systems.

ACKNOWLEDGMENTS

We would like to thank Dr. Robert LiKamWa, Jiqing Wen, Aashig Shaikh, and the METEOR Studio for their support in conducting this research. Additionally, appreciation goes out to Dr. Suren Jayasuriya for organizing the 2023 Visual Media REU program that stimulated the project. Finally, we want to thank the National Science Foundation for funding the program and its associated research.

APPENDIX

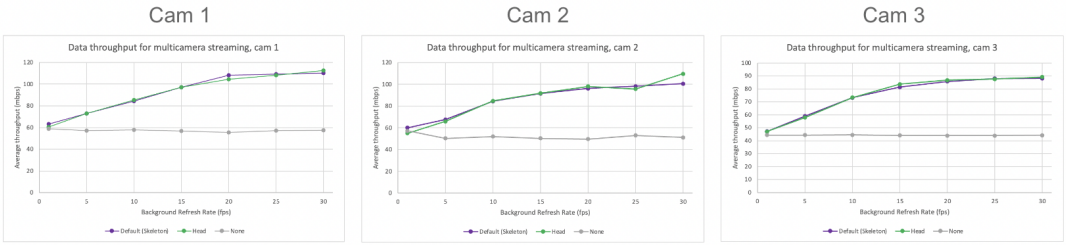

Figure 8: Temporal downsampling decreases throughput - as refresh rate is increased (equivalent to temporal downsampling *decreasing*) data processed per unit time increases. This trend holds true for single Kinect and multi-camera streaming.

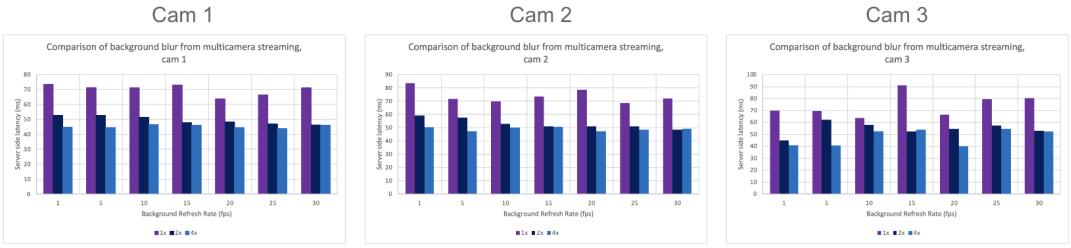

Figure 9: Spatial downsampling decreases latency - as background blur is increased (equivalent to spatial downsampling *increasing*) delay between execution and output time increases. This trend holds true for all cameras.

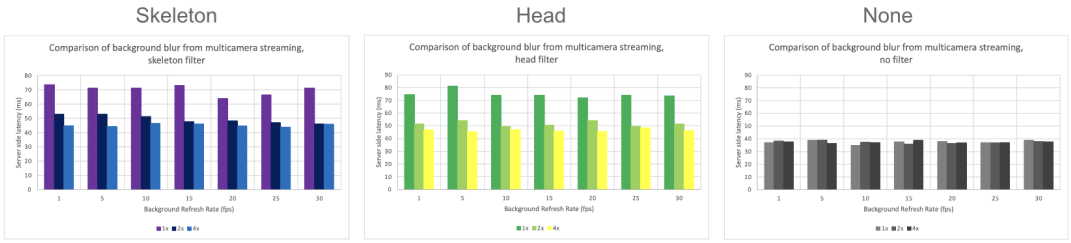

Figure 10: Temporal downsampling has no considerable impact on latency; with non-control filters, the decreasing trend is evident with variable blur factor and not variable refresh rates.

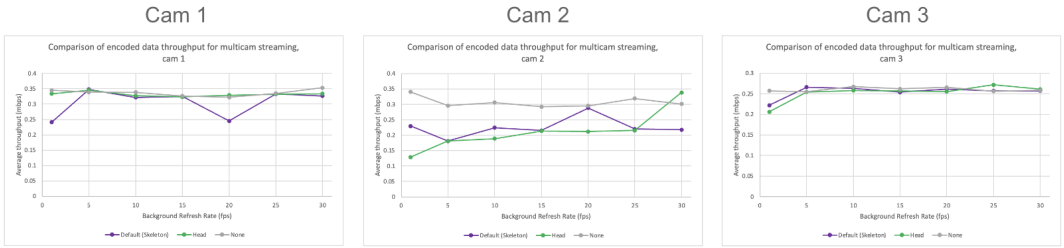

Figure 11: Temporal and spatial downsampling have little effect on upload throughput. As an upper bound, the requirement for uplaod bandwidth is under 1 Mbps and roughly 200-300 Kbps per Kinect camera.

REFERENCES

[1] Sameer Channar. 2022. Augmented coach: An augmented reality tool for immersive sports coaching. https://keep.lib.asu.edu/items/165564

[2] Bo Han, Yu Liu, and Feng Qian. 2020. Vivo. *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (2020). https://doi.org/10.1145/3372224.3380888

[3] Venkatesh Kodukula, Alexander Shearer, Van Nguyen, Srinivas Lingutla, Yifei Liu, and Robert LiKamWa. 2021. Rhythmic pixel regions: Multi-resolution visual sensing system towards high-precision visual computing at low power. *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2021). https://doi.org/10.1145/3445814.3446737

[4] Zeqi Lai, Y. Charlie Hu, Yong Cui, Linhui Sun, Ningwei Dai, and Hung-Sheng Lee. 2020. Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices. *IEEE Transactions on Mobile Computing* 19, 7 (2020), 1586–1602. https://doi.org/10.1109/TMC.2019.2913364

[5] Junhua Liu, Boxiang Zhu, Fangxin Wang, Yili Jin, Wenyi Zhang, Zihan Xu, and Shuguang Cui. 2023. CAV3: Cache-assisted viewport adaptive volumetric video streaming. *2023 IEEE Conference Virtual Reality and 3D User Interfaces (VR)* (2023). https://doi.org/10.1109/vr55154.2023.00033

[6] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. 2018. Cutting the Cord: Designing a High-Quality Untethered VR System with Low Latency Remote Rendering *(MobiSys '18)*. Association for Computing Machinery, New York, NY, USA, 68–80. https://doi.org/10.1145/3210240.3210313

[7] Fabian Mentzer, George Toderici, David Minnen, Sergi Caelles, Sung Jin Hwang, Mario Lucic, and Eirikur Agustsson. 2022. VCT: A Video Compression Transformer. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). https://openreview.net/forum?id=lme1MKnSMb

[8] Microsoft. [n. d.]. Microsoft/Azure-Kinect-sensor-SDK. https://github.com/microsoft/Azure-Kinect-Sensor-SDK

[9] Afshin Taghavi Nasrabadi, Aliehsan Samiei, and Ravi Prakash. 2020. Viewport Prediction for 360° Videos: A Clustering Approach. In *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video* (Istanbul, Turkey) *(NOSSDAV '20)*. Association for Computing Machinery, New York, NY, USA, 34–39. https://doi.org/10.1145/3386290.3396934

[10] Stefano Petrangeli, Gwendal Simon, and Viswanathan Swaminathan. 2018. Trajectory-Based Viewport Prediction for 360-Degree Virtual Reality Videos. In *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. 157–160. https://doi.org/10.1109/AIVR.2018.00033

[11] qm13. [n. d.]. Azure Kinect DK Hardware Specifications. https://learn.microsoft.com/en-us/azure/kinect-dk/hardware-specification

[12] ASU Meteor Studio. [n. d.]. ASU Meteor Studio codebase. https://github.com/asu-meteor/

[13] Jiqing Wen, Lauren Gold, Jinhan Hu, Alireza Bahremand, Aashiq Shaikh, Charmaine Farber, Yasser Dbeis, Sameer Channar, Connor Richards, Ryan Hoang, and et al. 2022. Adaptive 5G systems for interactive volumetric sports analysis in augmented reality. *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services* (2022). https://doi.org/10.1145/3498361.3538660