VVGLTF: EFFICIENT STREAMING OF VOLUMETRIC VIDEO WITH GLTF

Lam Kit Yung, Simone Croci, Philipp Haslbauer, Aljosa Smolic

Lucerne University of Applied Sciences and Arts, Switzerland

ABSTRACT

Volumetric Video (VV) lets viewers interact with realistic virtual 3D scenes in real-time. It enhances the quality and realism of video broadcasting and conferencing, delivering a vivid and immersive 3D experience. Holographic communication, concert and VR experiences benefit from it. VV data is often recorded as point clouds or meshes with shape and texture data. This leads to a substantial volume of data that requires efficient compression and streaming. Nevertheless, there is a lack of a universally accepted standards for VV file formats, resulting in many organizations implementing their own individualized approaches. Hence, we suggest VVglTF, a specific extension that supports the glTF file format and ensures VV compatibility. The open-source 3D content file format glTF is optimized for the Internet. We utilize glTF's rendering workflow to play VV of any duration or file size efficiently. Custom extensions in glTF expand the functionality of the glTF model format and offer customization of VV playback. We also developed a simple and efficient VVglTF streaming system based on HTTP Live Streaming technology for video texturing. It is designed to efficiently play VV content across different network conditions by adjusting the frame rate and number of frames per glTF file. In our experiments we validate the efficiency of our approach.

INTRODUCTION

Volumetric video (VV) is a type of video, typically created by simultaneously capturing and reconstructing 3D scenes from various perspectives, often including humans or objects, with the aim of producing an authentic and engaging experience for its viewers. VV technology brings real people into virtual spaces and facilitates 3D interaction inside the Metaverse. VV pipelines can include four major modules as shown in Figure 1, from Volumetric Capture, Volumetric Processing, Volumetric Encoding to Decoding and Rendering. VV is produced by utilizing cameras, sensors and software, which collaborate to acquire and analyse data from several viewpoints.

VV files are typically stored as dynamic 3D point clouds (1, 2) or dynamic 3D meshes (3, 4, 5) which necessitate the storage of geometric information in addition to texture (i.e. colour information). This results in an enormous quantity of data that must be compressed and transmitted efficiently. Nevertheless, there is no universal standard for VV file formats. Various companies and organizations have developed their own proprietary solutions. As a result, the majority of VV content is currently not accessible across several platforms to the users, while Metaverse applications require cross platform support, and users with different devices should have the same content and same user experience. To address this limitation, some initiatives have been made to establish interoperability and open specifications for VV compression, packaging, and transmission. An example of such a specification is the Volumetric Player Format Specification, published by the standards consortium Volumetric Format Alliance (20). The specification outlines standardized methods for capturing and compressing extensive scenes containing more than 100 individuals, with a data range of 9 megabits to 120 megabits. The document further delineates techniques for establishing a linkage between a volumetric playback application and a backend server, therefore enabling streaming of volumetric data.

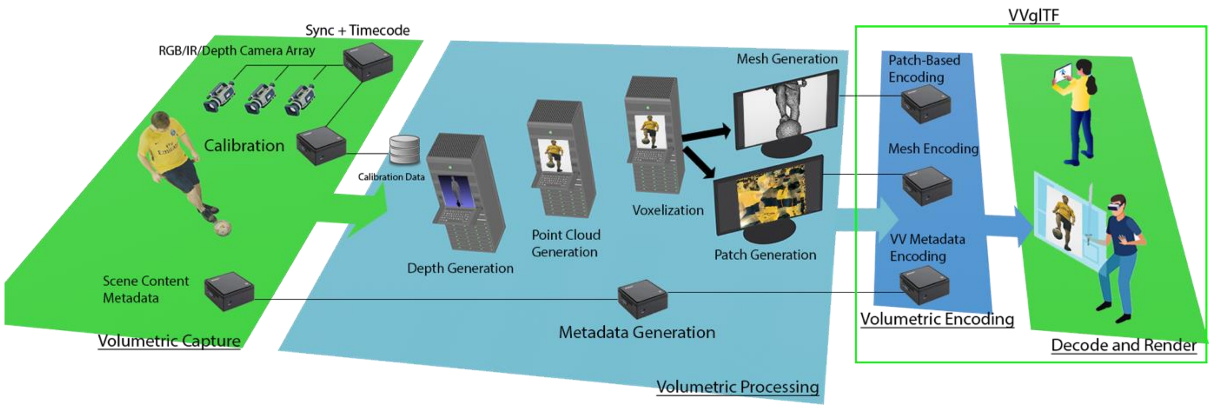
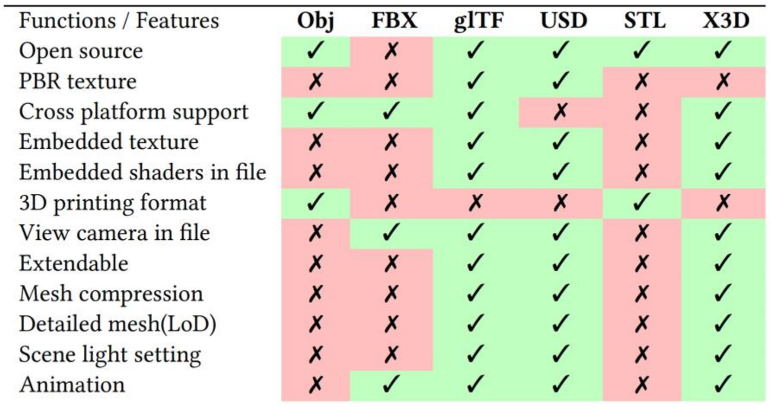


Figure 1 – Typical volumetric video pipeline, VVglTF is the solution for volumetric encoding, decoding and rendering.

Another example is MPEG-5 Part 2 LCEVC, published by the Moving Picture Experts Group (MPEG), an international standardization organization. The specification defines a low-complexity enhancement video codec (LCEVC) that can be used to enhance the quality and efficiency of any existing or future video codec. It also supports VV coding using point clouds or meshes as input formats.

To attain the interoperability of VV data, a potential approach involves enabling VV playback using existing 3D file formats. We examined six widely used 3D file formats (OBJ, FBX, USD, glTF, X3D, STL). glTF emerges as the most viable standard file format when compared to the other formats (Table 1). Similar to 2D video, VV has multiple frames. Each VV frame is a 3D reconstruction of the real world, and the data should be visualized frame by frame. glTF is a JSON-based 3D scene description file format, which contains a whole scene, including geometry, materials, textures, and animations.

glTF extensions expand the functionality of the underlying glTF baseline format. Extensions provide the capability to add novel attributes, such references to other data. Additionally, extensions can describe the structure and format of this external data. Furthermore, extensions can offer new meanings for parameters, establish reserved identification numbers, and create fresh container formats. Extensions have the potential to be included into the core glTF specification in a subsequent version of glTF.



Functions / Features	Obj	FBX	glTF	USD	STL	X3D
Open source	✓	✗	✓	✓	✓	✓
PBR texture	✗	✗	✓	✓	✗	✗
Cross platform support	✓	✓	✓	✗	✗	✓
Embedded texture	✗	✗	✓	✓	✗	✓
Embedded shaders in file	✗	✗	✓	✓	✗	✓
3D printing format	✓	✗	✗	✗	✓	✗
View camera in file	✗	✓	✓	✓	✗	✓
Extendable	✗	✗	✓	✓	✗	✓
Mesh compression	✗	✗	✓	✓	✗	✓
Detailed mesh(LoD)	✗	✗	✓	✓	✗	✓
Scene light setting	✗	✗	✓	✓	✗	✓
Animation	✗	✓	✓	✓	✗	✓

Table 1 – Comparison of 3D file formats.

To enable the VV features in glTF, we developed a custom extension following the current architecture of the glTF standard. In this specification, every frame of a video sequence has a corresponding mesh stored in the VVglTF file. This way the video provides a texture for every 3D mesh of the VV sequence. Using our VVglTF file format enables us to simply apply any codec with the portable file format and to support multi-platform.

The contributions of our work are as follows. First, we designed and implemented a novel cross platform glTF file format with corresponding VV player, i.e. VVglTF. Second, we developed a novel VV streaming system based on VVglTF that allows playback with adaptive streaming. This enables live streaming of VV in glTF. Finally, we conducted an experimental evaluation with VV examples, validating the efficiency of our approach.

RELATED WORK

Volumetric Video Content Creation

VV (6) allows to reconstruct real world scenes and objects in 3D, enabling visualisation and interaction with 6 DoFs, namely, interactive selection of the location and the direction of the viewpoint. This results in high interactivity and realism. In order to capture volumetric video, typically dedicated studios with multiple fixed cameras are used (3, 4). Nevertheless, handheld cameras and capture in-the-wild has also been done (5). In addition to cameras often also depth sensors are used (3).

After the acquisition with colour and optional depth cameras, usually the scene is reconstructed independently for each frame obtaining temporally inconsistent reconstructions. Then, the reconstructions are processed to get temporally consistent reconstructions (3, 4, 5). For temporal consistency, the reconstructions are tracked obtaining registered reconstructions. In the case of meshes, the registered reconstructions have the same topology and connectivity over a certain time, which helps to define a fixed texture atlas. Temporal consistency is useful since it improves storage and streaming of the content.

There are also systems, that use an image-based rendering approach instead of meshes. For example, in Project Starline (8), colour and depth images are streamed using a custom format.

Related to these VV generation methods, different storage and file formats are adopted. A universal display and data storage file format is missing, limiting interoperability and data sharing among them. VVglTF aims to provide a standardised format for VV display and data storage with our new glTF scene descriptor.

Point Cloud Compression

As an alternative to dynamic 3D meshes, VV can also be represented as dynamic 3D point clouds. Point Cloud Compression (PCC) is a method used to compress such data. MPEG-PCC (17) is an intricately developed standard for point clouds (ISO/IEC 23090-5) that includes a collection of 3D coordinate points (x, y, z) together with reflectance and RGB properties assigned to each point (18). The MPEG-PCC standard facilitates efficient and sustainable distribution of real-world point cloud data. As open standard, MPEG-PCC data can be easily adapted to different platforms. It could also be used in combination with VVglTF.

Point Cloud Streaming

Besides methods for compression there are methods tailored specifically for online streaming. For example, Kammerl et al. (9) encodes differences between frames computed by a double octree structure to leverage temporal redundancies. This approach does not consider user adaptation. A term used to refer to techniques that involve user adaptation is adaptive streaming. These methods first segment the point cloud into non-overlapping regions that are encoded at different quality levels, and only the regions within the viewport are streamed at a high-quality level. Examples of adaptive streaming approaches are DASH-PC (10), Park et al. (11, 12) use 3D tiles, Subramanyam et al. (13) use tiles, and Han et al. presented ViVo(14) where the point cloud is segmented into cells. Recently, machine learning models have been introduced by Huang et al. (15) and Zhang et al. (16).

GLTF EXTENSION DESIGN

glTF is a standard for distribution of 3D content. It is widely adopted and enables interoperability across systems and platforms that use 3D data. The importance is increasing with growing popularity of applications such as XR and the Metaverse. As a 3D content format, glTF follows basic principles from the computer graphics community, being a node-based scene description format. These principles are somewhat different from those in the video communication community, rather thinking in sequences of video frames. VV is both, dynamic 3D geometry with a sequence of video associated as textures. Our extension VVglTF is designed to combine the principles efficiently. A video sequence, which can be encoded in any supported standard format, is combined with a sequence of 3D meshes, which can also be encoded in any supported standard format.

VVglTF is a custom extension to enable VV playback based on the glTF 2.0 standard. It utilizes the same underlying graph structure of glTF as the core specification as depicted in Figure 2. The VV controller uses the 3D render engine, video player and glTF 3D loader API to control the VV playback.

VVglTF uses the video player API to decode the video, get the video duration (total number of frames), frame rate and the video frames. The VV controller converts each video frame to a texture and assigns them to the material of the VV meshes. The glTF render engine only renders and shows the node elements specified in the scene description. It ignores the elements missing in the scene structure, even if the elements are embedded in the glTF file. VVglTF uses this feature to enable efficient rendering of VV with glTF. All meshes of a given sequence of VV (chunk or group-of-frames) of a certain length (number of frames, see below) are stored in the VVglTF file, but only one should be displayed at a time. This is controlled by the video, which activates only one 3D mesh at a time and provides the corresponding texture to it.

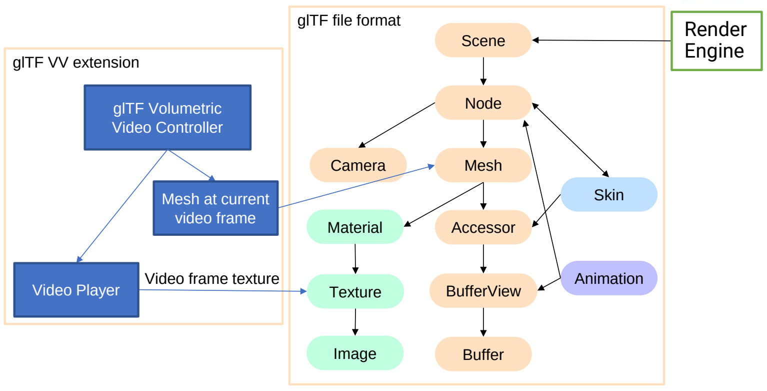


Figure 2 – Design of VVglTF.

Since glTF is a scene description file, it keeps the composition of a scene containing both VV and normal 3D objects. The VV node element contains the extension properties, it keeps the video address and a list of mesh IDs corresponding to the sequence of the video. The VV controller creates the 3D object based on the node configuration and switches the mesh primitive data of the VV following the playback of the video. VVglTF keeps all 3D meshes of the VV sequence in the buffer, like any other 3D mesh in the glTF file. Each VV frame mesh has a unique mesh ID and primitive description data in the glTF file, following the glTF 2.0 specification. Any 3D mesh coding can be applied here in principle to efficiently compress the data. In our experiments below we used Draco.

The mesh primitive data in the glTF data structure can be saved either in a separate binary file or embedded in glTF JSON. The file stream loader retrieves the VV mesh data in response to sequential requests from the VV controller. A delay of VV playback may be caused by loading of mesh buffer data from the binary data. A long VV file has a longer searching time of mesh buffer data from the binary data. Proper alignment of VV frame textures with the corresponding mesh objects is crucial for smooth playback of VV. Nevertheless, the loading time for video textures is often faster than the loading time for mesh primitive data due to discrepancies in resource loading time. This may result in desynchronization between the mesh and texture, as shown in Figure 3. To prevent desynchronization, the VV controller preloads multiple mesh primitive data into memory ahead of the video frame.

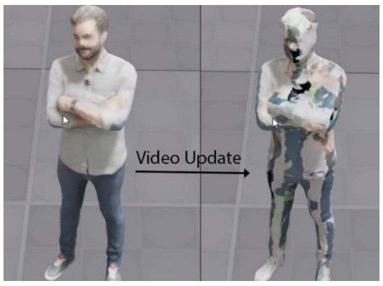


Figure 3 – Desynchronisation of texture and mesh resulting in flickering.

ADAPTIVE SREAMING WITH VVGLTF

We tested and applied VVglTF on different platforms including OpenXR, Unity, and WebXR. The latter for instance allows to run games and other interactive 3D content in any mobile or desktop browser. However, VV content creation can result in a tremendous amount of data (video and dynamic geometry). Long VV sequences could result in prohibitive download times. In order to solve this and even to support close-to-real-time transmission, we developed an adaptive streaming system for VVglTF as illustrated in Figure 4. The VV is divided into chunks of data, each containing a certain number of frames (group-of-frames, GoF), corresponding to a certain duration. The shorter the GoF, the closer the system is to real-time transmission; however, the data overhead increases with decreasing GoF size. We evaluate this delay versus efficiency trade-off in our experiments below.

The VVglTF streaming system includes a simple server with RESTful API, VV capture clients and VV player clients (Figure 4). We support capture clients with different capturing setups, either high-end studios populated with dozens of cameras or mobile phones with mono camera. The capture client must generate 3D meshes of the capturing targets, convert the mesh textures to video, and assemble the data into VVglTF files on a GoF-by-GoF basis.

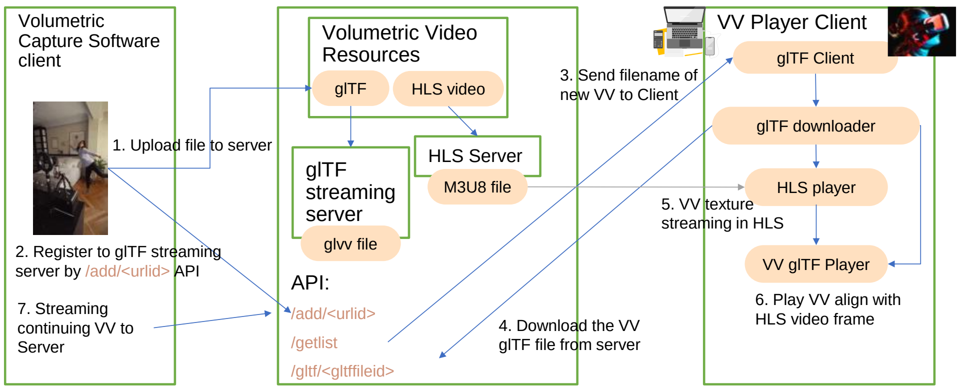

Volumetric Capture Software client
1. Upload file to server
2. Register to glTF streaming server by /add/<urlid> API
7. Streaming continuing VV to Server
Volumetric Video Resources
glTF
HLS video
glTF streaming server
glvv file
HLS Server
M3U8 file
API:
/add/<urlid>
/getlist
/gltf/<gltffileid>
3. Send filename of new VV to Client
4. Download the VV glTF file from server
5. VV texture streaming in HLS
VV Player Client
glTF Client
glTF downloader
HLS player
VV glTF Player
6. Play VV align with HLS video frame

Figure 4 – Architecture and data flow of our VVglTF streaming system.

The capture client uploads VV textures as video (mp4) files to the server. The server generates the HTTP Live Streaming (HLS) m3u8 files for the VV. The VV player client downloads the list of VVglTF files using an HTTP get request. If the VV is separated into multiple GoFs, the volumetric capture client provides a VV configuration file (.glvv file extension) containing the list of all VVglTF file names in the sequence of VV. The volumetric capture client continuously uploads the GoFs and updates the glvv file to the server for live streaming. At the same time, the VV player client downloads the VVglTF files and corresponding texture videos, to decode them and to visualize the content.

EVALUATION

The GoF size is a crucial parameter in our design, as it introduces an inherent playback delay. Therefore, a smaller GoF size is desirable for delay-sensitive applications. On the other hand, it is more efficient regarding data size to pack more 3D meshes into a single GoF, as the relative overhead is reduced this way. In order to evaluate this trade-off, we conducted the following experiments.

Figure 5 – VV examples from the vsenseVVDB2 dataset (19).

We selected 4 VV sequences (AxeGuy, LubnaFriends, Rafa2, Matis) from the vsenseVVDB2 database (19), each containing 300 VV frames at 30fps. Example views are shown in Figure 5. We encoded them into VVglTF files, first without any 3D mesh compression. We varied the GoF size as 300, 150, 60, 30, 25, 15, 5, 1, which corresponds to delays of 10s, 5s, 2s, 1s, 0.83s, 0.5s, 0.17s, and 0s, respectively. We then calculated the total file size that is needed to represent the corresponding VV sequences for each delay. The results are shown in Figure 6. We find that from a delay of 1s the data size practically saturates. Even at a delay of 0.5s the overhead is still very reasonable, a finding that is in line with common video coding standards and typical GoP sizes used there.

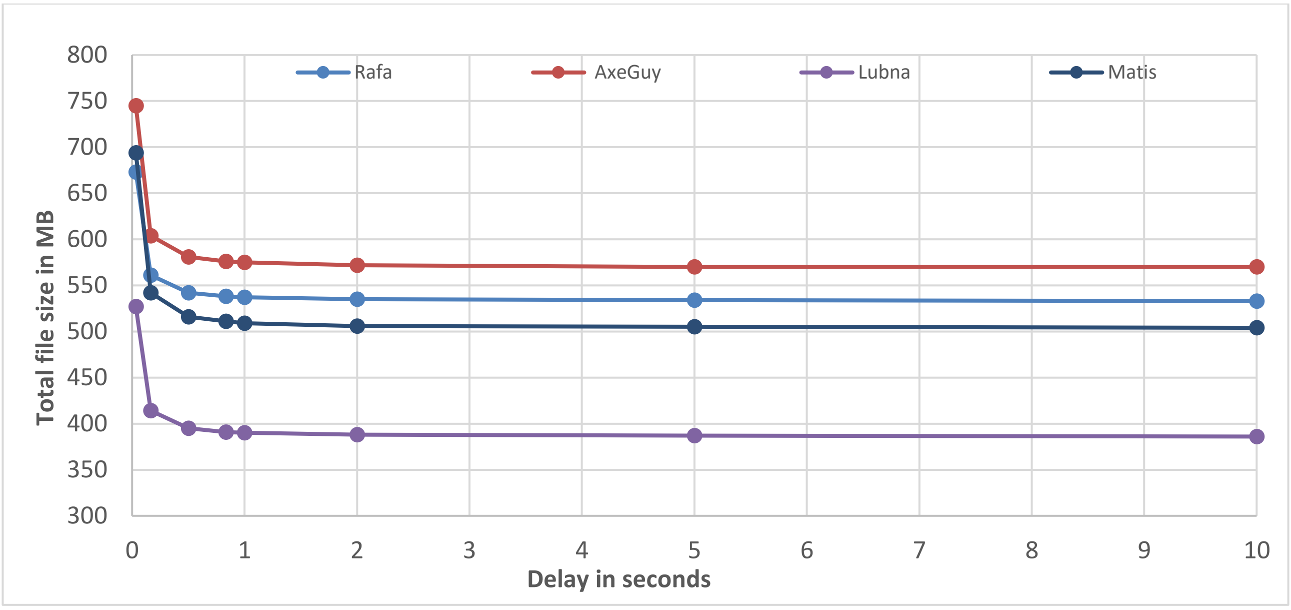


Figure 6 – Total VVglTF file size (300 frames) vs. delay (based on GoF size), uncompressed.

As a second step we encoded the meshes in the VVglTF files using the commonly used 3D mesh codec Draco, which is available as standard in glTF. For that we used typical settings of Draco (level 5) providing good visual quality (i.e. almost no coding artifacts visible). The results are shown in Figure 7 (please note different scaling of the x-axis). Again, we find that reasonable delays of 0.5s or less are possible without substantial increase of the data size. We further notice that the overall data rate is decreased by a factor of 10 compared to the uncompressed version. The additional data for the separate video files at good visual quality are: Rafa 44.6MB, AxeGuy 45MB, Lubna 48.8MB, Matis 49.9MB. This validates that VVglTF is well suited for efficient representation of VV.

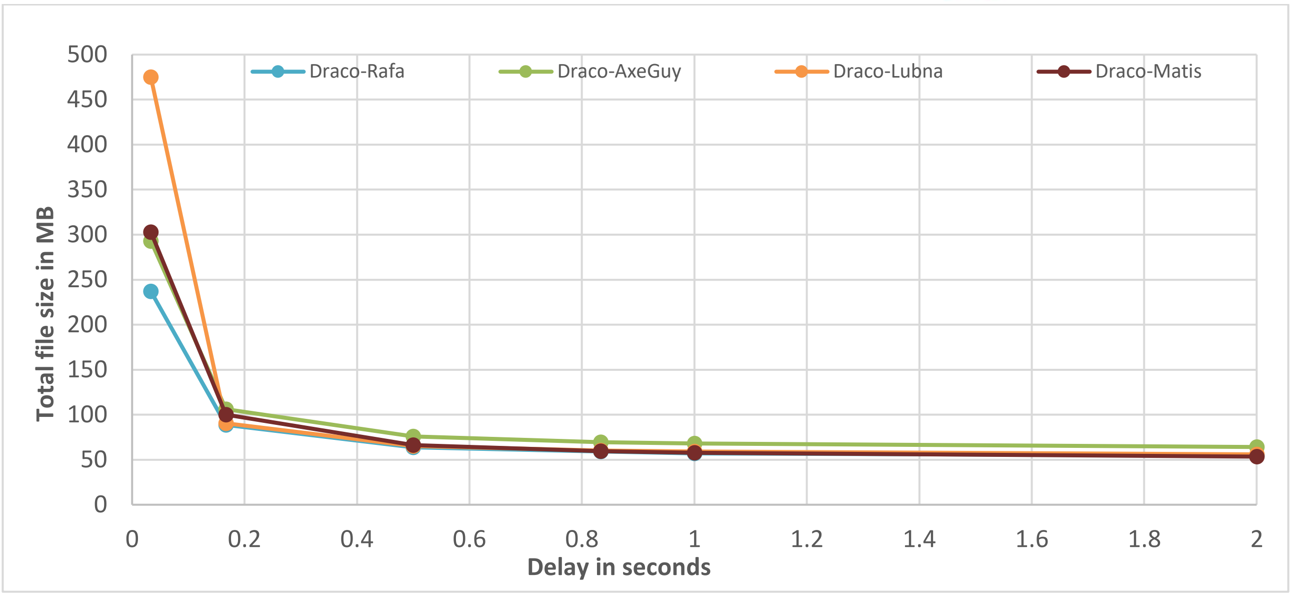


Figure 7 – Total VVglTF file size (300 frames) vs. delay (based on GoF size), Draco compressed.

CONCLUSIONS

This paper presents VVglTF and serves as a groundwork for enabling VV in glTF, which is an important standard for 3D content, ensuring interoperability and efficiency across platforms and systems. We tested and applied it via OpenXR, Unity and WebXR. With VVglTF it is possible to bring VV for example into mobile and desktop browsers and to use it in corresponding 3D applications. For efficient streaming, we developed an adaptive approach, including standard video streaming components such as HLS. Our experiments validate that VVglTF is efficient regarding delay and compression.

This evaluation does not consider limitations arising from terminal devices such as mobiles or head mounted displays, which may cause additional delays or performance problems. In our future work, we will consider such difficulties on low performance devices (in particular standalone XR headsets). We will further investigate live streaming for real-time holographic communication, and combination with other types of VV representations such as point clouds or Gaussian splatting.

REFERENCES

1. Reimat, I., Alexiou, E., Jansen, J., Viola, I., Subramanyam, S., & Cesar, P., 2021. Cwipc-sxr: Point cloud dynamic human dataset for social xr. Proceedings of the 12th ACM Multimedia Systems Conference. pp. 300 to 306.

2. Sterzentsenko, V., Karakottas, A., Papachristou, A., Zioulis, N., Doumanoglou, A., Zarpalas, D., & Daras, P., 2018. A low-cost, flexible and portable volumetric capturing system. 2018 14th international conference on signal-image technology & internet-based systems, pp. 200-207.

3. Collet, A., Chuang, M., Sweeney, P., Gillett, D., Evseev, D., Calabrese, D., ... & Sullivan, S., 2015. High-quality streamable free-viewpoint video. ACM Transactions on Graphics (ToG).

4. Hilsmann, A., Fechteler, P., Morgenstern, W., Paier, W., Feldmann, I., Schreer, O., & Eisert, P., 2020. Going beyond free viewpoint: creating animatable volumetric video of human performances. IET Computer Vision, 14(6), pp. 350 to 358.

5. Pagés, R., Amplianitis, K., Monaghan, D., Ondřej, J., & Smolić, A., 2018. Affordable content creation for free-viewpoint video and VR/AR applications. Journal of Visual Communication and Image Representation, 53, pp. 192 to 201.

6. Valenzise, G., Alain, M., Zerman, E., & Ozcinar, C. (Eds.)., 2022. Immersive Video Technologies.

7. Adelson, E. H., & Bergen, J. R., 1991. The plenoptic function and the elements of early vision (Vol. 2). Cambridge, MA, USA: Vision and Modeling Group, Media Laboratory, Massachusetts Institute of Technology.

8. Lawrence, J., Goldman, D., Achar, S., Blascovich, G. M., Desloge, J. G., Fortes, T., ... & Tong, K., 2021. Project starline: A high-fidelity telepresence system. ACM Transactions on Graphics (TOG), 40(6), pp. 1 to 16.

9. Kammerl, J., Blodow, N., Rusu, R. B., Gedikli, S., Beetz, M., & Steinbach, E., 2012, May. Real-time compression of point cloud streams. 2012 IEEE ICRA, pp. 778 to 785.

10. Hosseini, M., & Timmerer, C., 2018, June. Dynamic adaptive point cloud streaming. Proceedings of the 23rd Packet Video Workshop, pp. 25 to 30.

11. Park, J., Chou, P. A., & Hwang, J. N., 2019. Rate-utility optimized streaming of volumetric media for augmented reality. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 9(1), pp.149 to 162.

12. Park, J., Chou, P. A., & Hwang, J. N., 2018, December. Volumetric media streaming for augmented reality. 2018 IEEE GLOBECOM, pp. 1 to 6.

13. Subramanyam, S., Viola, I., Hanjalic, A., & Cesar, P., 2020, October. User centered adaptive streaming of dynamic point clouds with low complexity tiling. Proceedings of the 28th ACM international conference on multimedia, pp. 3669 to 3677.

14. Han, B., Liu, Y., & Qian, F., 2020, April. ViVo: Visibility-aware mobile volumetric video streaming. Proceedings of the 26th annual international conference on mobile computing and networking, pp. 1 to 13.

15. Huang, Y., Zhu, Y., Qiao, X., Tan, Z., & Bai, B. (2021, October). Aitransfer: Progressive ai-powered transmission for real-time point cloud video streaming. Proceedings of the 29th ACMMM, pp. 3989 to 3997.

16. Zhang, A., Wang, C., Han, B., & Qian, F., 2021, February. Efficient volumetric video streaming through super resolution. Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications, pp. 106 to 111.

17.Schwarz, S., Preda, M., Baroncini, V., Budagavi, M., Cesar, P., Chou, P. A., ... & Zakharchenko, V. (2018). Emerging MPEG standards for point cloud compression. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 9(1), 133-148.

18. Ilola, L., Kondrad, L., Schwarz, S., & Hamza, A., 2022. An overview of the mpeg standard for storage and transport of visual volumetric video-based coding. Frontiers in Signal Processing, 2, 883943.

19. Zerman, E., Ozcinar, C., Gao, P., & Smolic, A., 2020, May. Textured mesh vs coloured point cloud: A subjective study for volumetric video compression. Twelfth QoMEX, pp. 1 to 6).

20. Blanderson, M., 2021. Verizon, Canon, RED Among Founding Members of Volumetric Format Association.

ACKNOWLEDGEMENTS

This project has been funded by the European Union as part of the Horizon Europe Framework Program (HORIZON), under the grant agreement 101070109.