# patchVVC: A Real-time Compression Framework for Streaming Volumetric Videos

Ruopeng Chen
Shandong University
Qingdao, Shandong, China
chenrp@mail.sdu.edu.cn

Mengbai Xiao*
Shandong University
Qingdao, Shandong, China
xiaomb@sdu.edu.cn

Dongxiao Yu
Shandong University
Qingdao, Shandong, China
dxyu@sdu.edu.cn

Guanghui Zhang
Shandong University
Qingdao, Shandong, China
gh.zhang@sdu.edu.cn

Yao Liu
Rutgers University
New Brunswick, New Jersey, USA
yao.liu@rutgers.edu

## ABSTRACT

Nowadays, volumetric video has emerged as an attractive multimedia application, which provides highly immersive watching experiences. However, streaming the volumetric video demands prohibitively high bandwidth. Thus, effectively compressing its underlying point cloud frames is essential to deploying the volumetric videos. The existing compression techniques are either 3D-based or 2D-based, but they still have drawbacks when being deployed in practice. The 2D-based methods compress the videos in an effective but slow manner, while the 3D-based methods feature high coding speeds but low compression ratios. In this paper, we propose patchVVC, a 3D-based compression framework that reaches both a high compression ratio and a real-time decoding speed. More importantly, patchVVC is designed based on point cloud patches, which makes it friendly to an field of view adaptive streaming system that further reduces the bandwidth demands. The evaluation shows patchVCC achieves the real-time decoding speed and the comparable compression ratios as the representative 2D-based scheme, V-PCC, in an FoV-adaptive streaming scenario.

## CCS CONCEPTS

• **Computing methodologies → Image compression**; • **Computer systems organization → Real-time system architecture**; • **Information systems → Multimedia information systems**.

## KEYWORDS

Point Cloud Compression; Volumetric Video; Video Streaming

**ACM Reference Format:**
Ruopeng Chen, Mengbai Xiao, Dongxiao Yu, Guanghui Zhang, and Yao Liu. 2023. patchVVC: A Real-time Compression Framework for Streaming Volumetric Videos. In *Proceedings of the 14th ACM Multimedia Systems Conference (MMSys '23), June 7–10, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3587819.3590983

*Corresponding author.

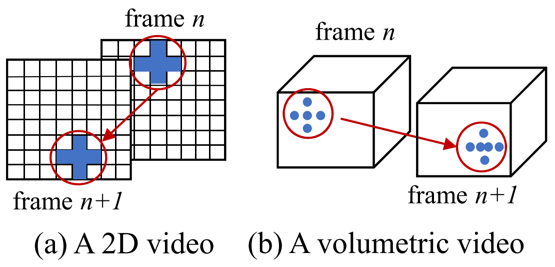

**Figure 1: Two examples of compensating pixels/points in 2D/3D frames. The pixels in the 2D frames are inherently aligned, and thus their color redundancy could be removed easily. The points in the 3D frames are not aligned. We may have different point numbers even for the same 3D object.**

## 1 INTRODUCTION

The recent advances in virtual reality (VR) and augmented reality (AR) technologies have triggered an emerging video type called volumetric video. When watching the video, the user is allowed to freely navigate, i.e., translating and rotating the viewport, in a rendered scene. By providing such a highly immersive watching experience, the volumetric video is expected to bring a new dimension to the industries like filmmaking[1] and interactive advertising.[2] The market for volumetric videos is reported to reach $4.9 billion by 2026 in a recently published market research.[3]

However, storing and streaming volumetric video suffers from its massive volume. A volumetric video is composed of temporally continuous point clouds as its frames, which could be captured by Lidars [38] and RGB-D cameras [32], or be reconstructed from 2-dimensional (2D) images [4]. A point cloud (PC) frame is a set of unordered points, and each point is associated with its Cartesian coordinates $\langle x, y, z \rangle$, the color information, and other optional attributes like curvature and normal. To reconstruct a realistic and immersive enough scene for the viewer, a volumetric video

[1]https://www.youtube.com/watch?v=iwUkbi4_wWo
[2]https://www.anayi.com/include_html/4d/21ss_4d_11.html
[3]https://www.marketsandmarkets.com/Market-Reports/volumetric-video-market-259585041.html

must contain frames with millions of points and reach 30 frames-per-second (FPS). Even if only the coordinates and the colors are preserved, streaming the volumetric video still requires a network bandwidth higher than the scale of Gbps, exceeding the available wireless bandwidth that could be provisioned nowadays [5].

To effectively reduce the volumetric video size, one of the most promising solutions is developing compression techniques for the underlying point clouds. Researchers either exploit the 3-dimensional (3D) data structures [11, 16, 18, 26], e.g., the octree or the kd-tree, to directly compress the point cloud, or project the points onto 2D planes followed by a traditional video codec [20, 28], like H.264 [36] or HEVC [33]. The 3D-based methods generally build a compact tree geometrically representing all points of a frame, and the positional residuals are compressed by an entropy encoder [6] or are directly removed. Without searching and eliminating the inter-frame redundancy, this thread of methods encodes and decodes the PC frames at satisfying speeds but only has a low compression ratio. In contrast, the 2D-based methods take advantage of the traditional codecs that are designed based on inter- and intra-frame redundancy elimination, thus compressing the PC frames more effectively. But the transformation between the 3D space and the 2D space substantially complicates the codec implementation, leading to a slow encoding/decoding speed. Moreover, the 2D-based methods are not friendly to a field-of-view (FoV) adaptive streaming system, which is also a promising technique that reduces the bandwidth requirement for delivering volumetric videos. To transmit the visible content only, the projected points must be rearranged on the 2D frames, and the video must be compressed again.

As 3D-based schemes are fast, the major challenge of improving its compression ratio is effectively locating inter-frame redundancy. It is not straightforward to migrate the motion compensation techniques in conventional 2D codecs to the 3D space. Unlike the 2D images, the points in the neighboring PC frames are not spatially aligned, and their numbers are not even necessarily the same as shown in Figure 1(b).

In this paper, we propose patchVVC, a **patch**-wise **v**olumetric **v**ideo **c**ompression framework. patchVVC is a 3D-based compression method that features a high compression ratio and fast decoding speed. More importantly, patchVVC encodes the volumetric video based on spatially divided patches so that we can directly figure out the visible data in compressed format, which is friendly to an FoV-adaptive streaming system. Specifically, the video frames are separated into groups of pictures (GOPs), which consist of a leading I-frame and subsequent P-frames, and the compression is carried out in each GOP individually. The encoding pipeline is composed of three stages, the *two-stage motion estimation*, the *patch deformation*, and the *data compression*.

In the two-stage motion estimation, the PC frames in a GOP are separated into patch groups, and for each group we extract a transformation matrix and combine it with the global trasfromation matrix of the GOP to jointly represent the rigid moves of consecutive patches. In patch deformation, we align the points across patches in a patch group by deforming them to a *common patch*, in which the $k$-means algorithm is adopted to find the geometrically aligned positions and they are colored by distance-weighted interpolation. We also exclude patches that are significantly deviated from the common patch and compress them individually. With the

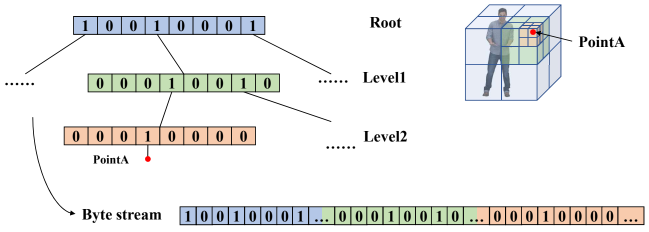

**Figure 2: A three-level octree indexes a point cloud without keeping the residuals.**

aligned points, one octree could represent the geometric structure of the patch group and the color redundancy is removed trivially. In data compression, the transformation matrices and the octrees are compressed by an entropy encoder [6], while the colors and the color residuals are encoded with the region-adaptive hierarchical transform method (RAHT) [7]. The procedures of encoding and decoding patch groups in patchVVC are independent of each other and thus are friendly to both the modern multi-threading architecture and the FoV-adaptive streaming system. We evaluate patchVVC with the 8i dataset [9]. The experimental results show that patchVVC reduces the size of compressed data by up to 7.72x compared to GROOT [18] and up to 5.73x compared to MP3DG [22]. patchVVC also approaches the representative 2D-based methods, V-PCC [28], in terms of the compression ratio when compressing complete videos, and even requires less bandwidth in an FoV-adaptive streaming scenario. It is worth noting that patchVVC decodes the volumetric videos at real-time speed, which is almost 20x faster than V-PCC and MP3DG.

The main contributions of this paper are summarized as follows:

- We propose a 3D-based compression framework that exploits the inter-frame redundancy in both the geometric and color domain to achieve a high compression ratio for volumetric videos.
- Our parallel-friendly method features a fast decoding speed, supporting real-time streaming applications.
- Our patch-based approach is friendly to the FoV-adaptive streaming scenarios, which can further reduce the bandwidth consumption when delivering the video.
- We implement a prototype codec based on the proposed framework, and the comparison results with the peer solutions justify our designs.

The rest of this paper is organized as follows. We first summarize the related work in Section 2, and then introduce background knowledge in Section 3. The encoder and decoder are described in Section 4 and Section 5, respectively. We report the experimental results in Section 6. The discussion is in Section 7 and the conclusion of the work is in Section 8.

## 2 RELATED WORK

### 2.1 Point Cloud Compression

The MPEG standard V-PCC [28] is a 2D-based scheme designed to compress point cloud sequences, where points in the 3D space are projected onto 2D frames. These frames are compressed with traditional codecs. Based on V-PCC, a prior study [20] suggests deriving additional motion vectors from the 3D point clouds for improving the encoding performance of the 2D codec. Or one can recognize the unoccupied pixels on the projected plane and trade their visual quality for a high compression ratio [19]. Considering the narrow scope of the user's FoV, view-PCC [41] introduces multi-view projection and distance-based clustering methods into the V-PCC pipeline so that adaptively streaming the volumetric video in a bandwidth-aware manner becomes possible. Though the 2D-based methods achieve high compression ratios, they suffer from slow coding speeds because of the complex pipelines.

In the 3D-based compression methods, a point cloud is commonly represented by a compact octree, and the tree is compressed by an entropy encoder [16, 26]. GROOT [18] further develops a parallel decoding tree that breaks the dependencies between the tree nodes at the last three layers, which is suitable for GPU and meets the real-time decoding speed. The MPEG standard G-PCC [28] divides a point cloud into blocks, where the encoding and decoding could be performed in parallel. To further improve the coding efficiency of dynamic point clouds, the 3D-based motion compensation [8, 21, 22, 37] that exploits the inter-frame redundancy is proposed. But these methods are based on the blocks, not the patches, which can hardly detect the motions across the block boundaries. On the other hand, the point attributes, especially the colors, are usually projected into a 2D image and use JPEG to compress it [18, 22]. RAHT is designed to compress the colors by manipulating the octree built on the points. RAHT features both high compression ratios and fast coding speeds. Another thread of studies compresses the point cloud sequence with graph-based transformation [14, 30, 40]. But they suffer from high computation overhead.

## 2.2 Volumetric Video Streaming

As an emerging type of multimedia data, the methods of saving the high bandwidth demands in streaming volumetric videos are still being explored. DASH-PC [15] spatially sub-samples the dense point clouds and constructs a point cloud-specific manifest. PCC DASH [34] proposes to stream multi-object scenes with rate adaptation heuristics that are based on the user and buffer status. Another method [31] segments the PC frames into tiles to accommodate the user-centric adaptation. These tiles must be independently codable to allow adaptive streaming based on the user's viewport. Other tile-based methods [23] use the window buffer instead of the queue buffer to handle frequent user interaction. The window buffer method eliminates the sequential concept of queues and can respond with latency to unexpected user interactions. The volumetric video streaming could also be improved by predicting the user's FoV [12, 13], which substantially reduces the motion-to-photon latency. To practically stream the volumetric video on mobile devices, Nebula [24] offloads the video decoding to the edge server that transcodes the 3D point clouds into 2D frames according to the user viewport. YuZu [39] is another streaming system that saves bandwidth with a point cloud upsampling model.

## 3 BACKGROUND

## 3.1 Iterative Closest Point Algorithm

The iterative closest point algorithm (ICP) [2] is widely used for point cloud registration. ICP aims at optimally overlapping a source point cloud over a target point cloud after a rigid transformation. ICP iteratively invokes four steps to build the transformation matrix: 1) the closest points searching, 2) the transformation computing, 3) the registration application, and 4) the convergence judgment. In an iteration, a rotation vector $\vec{q_r} = (q_0, q_1, q_2, q_3)^T$ represented by the quaternion and a translation vector $\vec{q_t} = (q_4, q_5, q_6)^T$ are found to minimize point distances between the source and the target point cloud. The transformation matrix of the iteration $k$ is then

$$M_r^k = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) & q_4 \\ 2(q_1q_2 + q_0q_3) & q_0^2 + q_2^2 - q_1^2 - q_3^2 & 2(q_2q_3 - q_0q_1) & q_5 \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 + q_3^2 - q_1^2 - q_2^2 & q_6 \end{pmatrix}$$

. While the optimal transformation is approached in $K$ iterations, the overall transformation matrix is $M_r = M_r^K \cdot M_r^{K-1} \cdot \ldots M_r^1$, and the transformed point cloud is $\mathbf{P}_s' = M_r \cdot \mathbf{P}_s$, where $\mathbf{P}_s$ is the source point cloud.

## 3.2 Octree-based Point Cloud Compression

Octree [26] is a compact data structure that indexes points in a 3D space. When building an octree, we recursively subdivide the space into 8 subspaces until a pre-defined level is reached or there is no point contained. The root node thus represents the whole space and the leaf nodes may contain any number of points. The intermediate nodes have at most eight children so they can be efficiently stored in a byte, where 1 indicates the corresponding subspace containing points and 0 means an empty subspace. We don't have to store the coordinates of the points anymore because the centers of the leaf nodes approximately represent the points contained. Or we can calculate the residuals of the point coordinates from their subspace centers, which can be more effectively compressed by entropy encoders. Such a tree structure is materialized into a byte stream according to the breadth-first search.

Figure 2 shows how a point cloud is indexed by an octree, and how the octree is represented by a byte stream. Recovering the coordinates of a point is thus calculating the center of the encapsulating subspace through tree traversal. But traversing the octree is hardly parallel-friendly, and a prior study [18] strives to adapt the tree structure to the GPU architecture for fast decoding. If the point residuals are kept, an additional step that adds the residuals to their centers is required.

## 3.3 RAHT

RAHT [7] is the state-of-the-art codec for compressing attributes in a static point cloud. RAHT is based on the Haar wavelet transform. It divides the point cloud into several blocks and constructs an octree on each block. Then, RAHT merges the tree nodes from bottom to top, and the merge operations of each node are carried out along the $x$-, $y$-, and $z$-axis. Taking the $x$-axis as an example, the merge operation follows

$$\begin{bmatrix} g_{l-1,x,y,z} \\ h_{l-1,x,y,z} \end{bmatrix} = \frac{1}{\sqrt{w_1 + w_2}} \begin{bmatrix} \sqrt{w_1} & \sqrt{w_2} \\ -\sqrt{w_2} & \sqrt{w_1} \end{bmatrix} \begin{bmatrix} g_{l,2x,y,z} \\ g_{l,2x+1,y,z} \end{bmatrix}$$

, where $g_{l,x,y,z}$ is the attribute signal of an octree node at layer $l$ and position $\langle x, y, z \rangle$, and $h_{l,x,y,z}$ is the coefficient. $w_1$ and $w_2$ are the point numbers of the octree nodes with $g_{l,2x,y,z}$ and $g_{l,2x+1,y,z}$. After all the merge operations, the root node signal $g_{0,0,0,0}$ and all coefficients $h_{l,x,y,z}$ are divided into sub-bands according to the

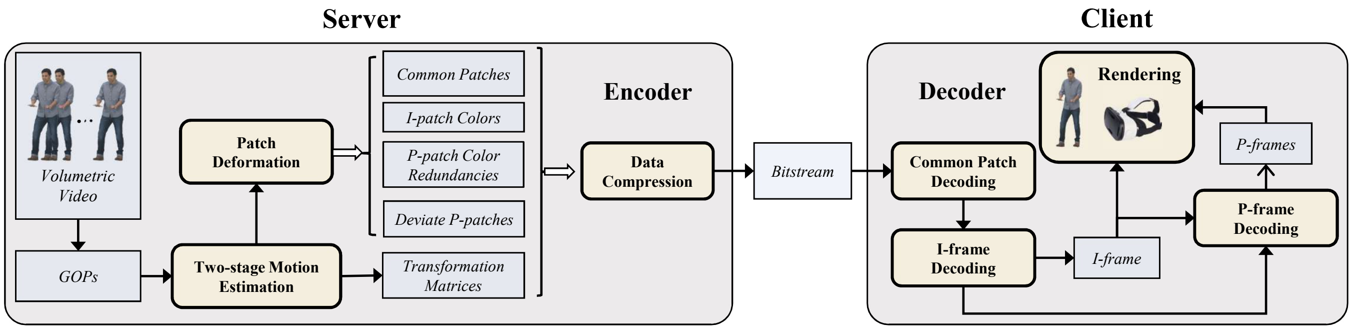

**Figure 3: The overview of patchVVC**

weights. Finally, each sub-band is quantized and encoded by an arithmetic codec.

The encoder is designed to receive consecutive PC frames of a volumetric video as the input and to output a bitstream for efficiently storing and streaming. The input PC frames are divided into groups of pictures (GOPs). A GOP has a leading I-frame and the following $N - 1$ P-frames. The encoder compresses the video one GOP by one GOP. When processing a GOP, the encoding pipeline consists of three phases, namely *two-stage motion estimation*, *patch deformation*, and *data compression*. The left side of Figure 3 shows how the encoder works.

### 3.4 Two-Stage Motion Estimation

In the two-stage motion estimation, our goal is to discover the rigid moves of patches in a GOP. After *global motion estimation* and *patch-wise motion estimation*, the I-frame is separated into $L$ patches, and we jointly use two matrices to characterize how one I-frame patch is transformed to match a P-frame. Based on this, we extract patches from the P-frames and form $L$ patch groups.

## 4 ENCODER

*4.0.1 Global Motion Estimation.* In a volumetric video, the neighboring PC frames are expected to be highly similar. The global motion estimation helps extract a compact transformation matrix $M_g$ that describes the major difference between an I-frame and a P-frame. We start by filling the encoding buffer with a GOP of $N$ frames, and always set up the first one as the I-frame and the others as the P-frames. The global motion estimation is carried out for each P-frame. We first align the centroids of the I-frame and the P-frame, introducing a better initial status to the ICP process. The centroid alignment results in a translation vector $\vec{q}_g$. After ICP is performed on the translated I-frame and the P-frame, a transformation matrix $M'_g$ is calculated. The overall transformation matrix $M_g$ from the global motion estimation is calculated following $M_g = M'_g + (\vec{0}, \vec{0}, \vec{0}, \vec{q}_g)$.

*4.0.2 I-patches Generation.* The global motion estimation calculates a unifying transformation between the PC frames, but the actual movement of a point cloud object is more subtle. For example, a walking person is moving forward but his hand might swing back. Thus, the point moves should be captured at a finer granularity. To

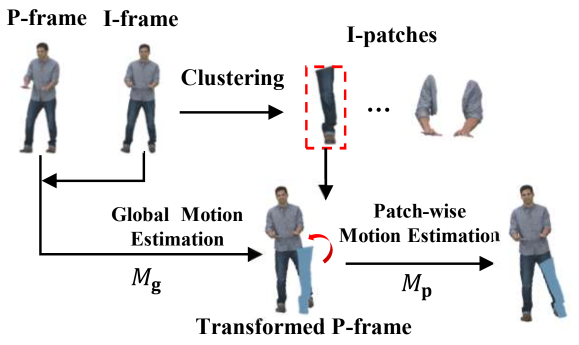

**Figure 4: The pipeline of two-stage motion estimation**

realize this, clustering algorithms could be employed to generate patches from the I-frame, namely *I-patches*.

We employ a straightforward yet effective method to generate I-patches. For a given I-frame, we equally divide it into two blocks along the dimension with the maximum span. The blocks are recursively split in the same manner until the number of points in a sub-block is less than a threshold. The threshold is experimentally determined according to the number of patches we want to generate for an I-frame. We then calculate the centroids of all blocks and classify every point to its nearest centroid. In the end, we separate an I-frame into compact and non-overlapped patches with almost the same number of points. Our experiments show that this method is effective in generating patches for a single 3D object with rigid moves. Additional clustering methods, e.g, DBSCAN [10], could be incorporated to generate I-patches in more complex scenarios.

*4.0.3 Patch-wise Motion Estimation.* By accepting an I-patch as the source and a complete P-frame as the target, the encoder calls ICP to locate where the patch might best cover. We also set up a local centroid alignment process to provide a preferable initial state for ICP: 1) We calculate the centroid of the I-patch; 2) We search the nearest neighbors of the points of the I-patch in the target P-frame, and calculate the centroid of these neighbors; 3) We calculate the translation vector between the two centroids. In the end, combining the translation vector of alignment and the transformation matrix of ICP, a patch-wise transformation matrix $M_p$ is calculated between an I-patch and a P-frame, indicating the subtle movement of a small

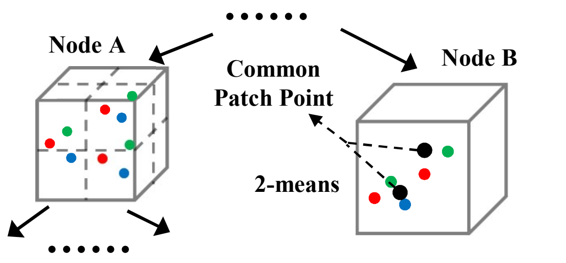

**Figure 5: Node A and Node B are two nodes of an octree. The red, green, and blue points are from three patches in a patch group. *k*-means is carried out in Node B to generate the points of the common patch, which are the black points.**

group of neighboring points. Figure 4 illustrates the processing pipeline of the patch-wise motion estimation.

*4.0.4 Patch Group Generation.* According to the patch-wise motion estimation, an I-patch properly overlaps a part of each P-frame. But we still need to generate patches from the P-frames, namely *P-patches*, which are associated with the I-patch to form a *patch group*. While the patches in a patch group are expected to represent the same part of a 3D object, we can extract the common geometric structure based on them. Specifically, if we have $L$ I-patches, we classify each point in the $n$-th P-frame into $L$ P-patches. The cluster index $l \in [0, L)$ of a point $p$ from the $n$-th P-frame is calculated as

$$\arg\min_{l} \; d(p, \mathbf{P}_l^n)$$

, where $\mathbf{P}_l^n$ is the $l$-th transformed I-patch towards the $n$-th P-frame, and

$$d(p, \mathbf{P}_l^n) = \min_{p' \in \mathbf{P}_l^n} \|p - p'\|$$

. In this way, the $n$-th P-frame is split into $L$ P-patches and each of them is associated with an I-patch. Every I-patch and its $N-1$ associated P-patches form a patch group, and we have $L$ patch groups in a GOP.

## 4.1 Patch Deformation

In the patch deformation stage, we generate a common patch for a patch group to align the points from different frames. After deforming a patch into its common patch, we color the points via a distance-weighted interpolation method. In the end, each patch is represented by an octree of the common patch and the colors (the I-patch) or the color residuals (the P-patches). The deviate patches have their octree and colors.

*4.1.1 Common Patch Construction.* Once we have a patch group, we can attempt to summarize the common geometric structure for them. Inherently, the patches, no matter the I-patch or the P-patches, are expected to represent the same part of a 3D object and thus share a common geometric structure. However, due to the noisy nature of positional signals, the points of these patches are not aligned in the 3D space. Despite that we can build an octree over these points to generalize the geometric structure, we still have to preserve distinguish details for each patch at the leaf nodes, which incurs significant overhead (the number of leaf nodes is almost 8x more than the intermediate nodes in an octree). To summarize the common geometric structure in a more cost-efficient way, we propose to deform every patch into a common patch that is the most geometrically similar to all patches involved.

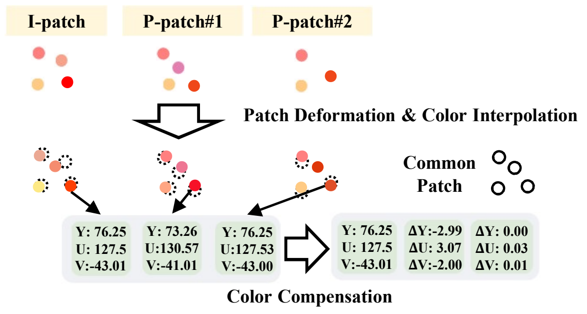

**Figure 6: An example of color interpolation and compensation**

The common patch is generated as follows: we merge all patches of a patch group (the I-patch and the transformed P-patches) into a dense point cloud and start to build an octree over them. In the tree-building process, a branch stops growing if the current level of a node contains exactly one point from any patch or if the tree is deep enough. This ensures the lowest level of tree nodes has information from all frames. While all branches have stopped growing, we conduct the $k$-means algorithm [17] in the leaf nodes individually, where $k$ is the number of points in a leaf node divided by the size of the patch group. The process of generating the common patch for a patch group is shown in Figure 5. After the $k$-means algorithm, all of the clustering centroids form the common patch, and its geometric structure is close to all patches in the patch group. It is worth noting that we still need to build another octree based on the common patch for the later compression.

Occasionally, a P-patch in the patch group has changed shape from the I-patch, e.g., the fabric of a clothes flies in the wind. If we involve such a deviate patch in the common patch construction, the quality of the result is likely to have deteriorated. So we detect a deviate patch in a patch group according to the MSE of its distance from the I-patch. For the $n$-th P-patch $\mathbf{P}^n$, we calculate the distance MSE as

$$\text{MSE}(\mathbf{P}^n) = max(f(\mathbf{P}^n, \mathbf{P}^0), f(\mathbf{P}^0, \mathbf{P}^n))$$

, where $\mathbf{P}^0$ is the I-patch in the same patch group, and

$$f(\mathbf{P}, \mathbf{Q}) = \sum_{p \in \mathbf{P}} \min_{q \in \mathbf{Q}} \|p - q\|$$

defines an asymmetric distance from a patch $\mathbf{P}$ to another patch $\mathbf{Q}$. Only if $\text{MSE}(\mathbf{P}^n)$ is greater than a predefined threshold, the $n$-th P-patch is a deviate patch and is excluded from the common patch construction. These deviate patches are encoded independently as the I-patches.

*4.1.2 Color Interpolation and Compensation.* Though the common patch represents the unifying geometric structure of a patch group, the color information of each patch might be different because of the changing lighting conditions. Therefore, we introduce a distance-weighted interpolation method to color the common patch for $N$ times. Specifically, the color of a point in the $n$-th colored common

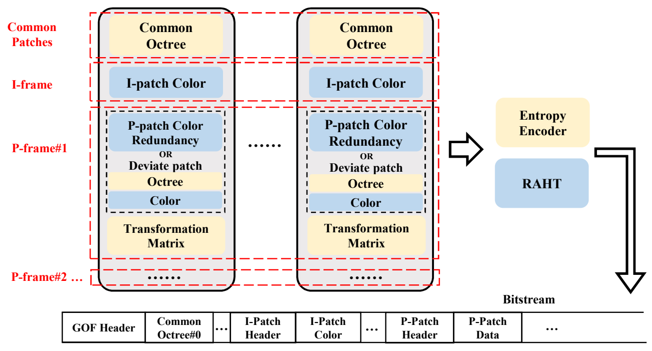

**Figure 7: The data compression pipeline**

patch $cc^n$ is interpolated by its k-nearest neighbors in the same patch

$$cc^n = \sum_{j=1}^{k} \frac{c_j^n \cdot w_j}{w}$$

, where $c_j^n$ is the color of the $j$-th nearest points in the $n$-th patch and $w_j$ is the reciprocal of distance from this neighbor to the point we want to interpolate. As $w_j$ weighs the color of the $j$-th nearest neighbor, $w$ is the sum of all $k$ weights. Figure 6 shows an example of interpolating the colors for different patches, where we use the same method to handle all color channels.

As long as we deform the members of a patch group into a common patch with different colors, we can straightforwardly compensate the colors of P-patches since their points are spatially aligned with the ones in the I-patch. We directly subtract the point colors of a P-patch from those of the I-patch and keep the color residuals with little energy for compression later. An example of color compensation is shown in Figure 6. The color information of the P-patches is reduced to quite small values, which can be more effectively compressed compared to the original color signals.

## 4.2 Data Compression

Until now, the PC frames in a GOP have been converted into transformation matrices (via the two-stage motion estimation), octrees (representing the common patches of patch groups and the deviate patches), colors (for the I-patches and the deviate patches), and color residuals (for the P-patches). Figure 7 shows how we compress and organize the video data into a bitstream. To further reduce the data volume, the colors and the color residuals are compressed with the RAHT [7] method, while the transformation matrices and the octrees are compressed with entropy coding. In the compressed bitstream, we always have a header of encoding parameters, e.g., the GOP size, the number of patches in a frame, etc. After this, the octrees representing the common patches are attached. Then, the compressed data of the I-frame are added followed by those of P-frames in the frame order. For each I-patch, we need to specify the patch group index and the RAHT quantization step in the I-patch header before the compressed colors. The compressed colors/color residuals of a P-patch follow a similar header except for an additional field indicating if it is a deviate patch. We also include the transformation matrices for each P-patch in addition to the compressed color information.

## 5 DECODER

When the decoder receives the bitstream, we follow three phases to decode the PC frames in a GOP: 1) decoding the common patch, 2) decoding the I-frame, and 3) decoding the P-frames. The right side of Figure 3 shows how the decoder works.

### 5.1 Common Patch Decoding

According to the GOP header, the decoder understands the GOP size $N$ and the number of patch groups $L$. With the parameters, the decoder reads $L$ compressed octrees and feeds them to the specified entropy decoder. With these octrees, we are able to recover all points in the common patches by calculating the coordinates of the leaf nodes. The coordinate residuals are added back if the encoder chooses to preserve them. It is worth noting that since the common patches are discovered independently, it is trivial to decode the octrees and recover the coordinates in parallel.

### 5.2 I-frame Decoding

The decoder recovers the I-frame by associating the colors with the common patches. According to the patch index in an I-patch header, the compressed colors are paired with the corresponding common patch. After decoding the colors with the color decoder, the common patch could be straightforwardly colored because the colors are organized in the order of extracting the points from the octree. All the recovered I-patches then form the I-frame to be rendered. Furthermore, as there is no data dependence between the I-patches, they could be also recovered in parallel. After decoding and rendering the I-frame, it should reside in the decoding buffer for the following decoding of P-frames.

### 5.3 P-frame Decoding

Decoding a P-frame demands recovering all P-patches belonging to it. Recovering a P-patch is similar to recovering an I-patch except that two additional steps must be involved: 1) We should use the transformed common patch as the skeleton instead, and 2) the colors are recovered by adding the decoded color residuals to the colors of the corresponding I-patch. For those deviate patches, they are decoded with their own octrees, and the remaining steps are the same as the I-patches. The P-frames are decoded frame by frame, and when decoding a P-frame, we launch multiple threads to decode the P-patches.

## 6 EVALUATION

**Implementation**: We implement our system with roughly 3000 lines of C++ code. We incorporate the ICP algorithm and the nearest neighbor search algorithm from PCL.[4] The state-of-the-art entropy encoder, Zstandard,[5] is employed to compress the octrees and the transformation matrices, and RAHT [7] is integrated to encode the colors and the color residuals. patchVVC always launches 30 threads for decoding because we experimentally notice that more threads contribute little to the decoding speed.

[4]https://github.com/PointCloudLibrary
[5]https://github.com/facebook/zstd

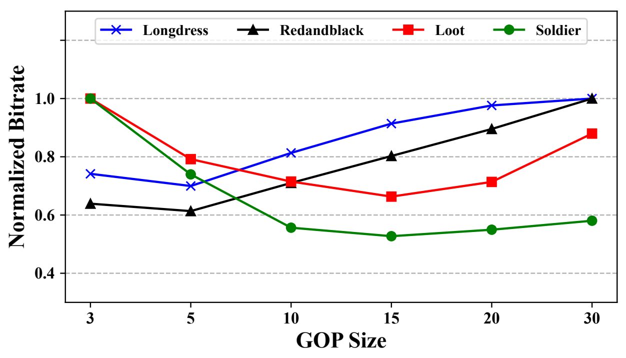

**Figure 8: The normalized bitrates of different videos when varying the GOP size**

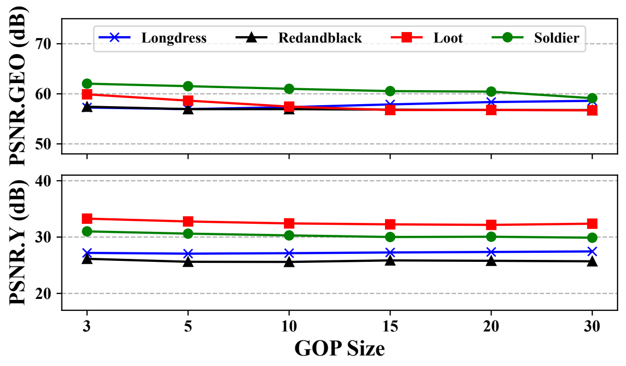

**Figure 9: The visual quality of different videos when varying the GOP size**

**Table 1: A brief introduction to the 8i dataset**

| Video Name | Description | Avg. # of Points/Frame | Required Bandwidth |
|---|---|---|---|
| *Loot* | A man playing piano | 794 K | 2.86 Gbps |
| *Longdress* | A female in a colorful dress | 834 K | 3.00 Gbps |
| *Soldier* | A man on guard with a gun | 1.01 M | 3.87 Gbps |
| *Redandblack* | A lady in a red and black skirt | 707 K | 2.63 Gbps |

**Experimental Setup**: All of our experiments are carried out on a machine having two 2.90GHz Intel Xeon CPUs and 128GB memory. We compare patchVVC with three peer methods, GROOT [18], V-PCC [28], MP3DG [22]. GROOT is a GPU-accelerating 3D-based compression method, which uses PD-tree, an octree optimized for massive parallelism, to compress the geometric structure and JPEG to compress the colors. In the experiments, we implement our own version of GROOT following the published work [18] on one NVIDIA Geforce RTX 3090 GPU. V-PCC is the most widely recognized 2D-based compression solution for volumetric video. It reaches a higher compression ratio than the 3D-based methods but takes more time to encode/decode the data. We use the reference implementation of V-PCC that is publicly available.[6] MP3DG [21] is a point cloud codec designed for 3D immersive videos. MP3DG also exploits inter-frame redundancy and supports multi-threading encoding/decoding. Different from patchVVC we capture the redundancy between patches, and MP3DG carries out the motion estimation between blocks from adjacent frames. In the evaluation, we use the official implementation of MP3DG that is publicly available.[7]

All the experiments are performed on the 8i dataset [9], which consists of four 10-sec 30FPS volumetric videos. A brief introduction to the 8i dataset is given in Table 1. The point cloud frames are presented as text-based `.ply` files, where a line means a point and consists of 3 coordinates and 3 channels of its RGB color. In the experiments, the coordinates are turned into 32-bit integers, and the RGB-formatted colors are converted to YUV colors according to the ITU-R standard [25]. For all 3D-based methods, we set the octree resolution at 1, and discard the residuals because it barely affects the visual quality. The metrics PSNR.GEO and PSNR.Y [27] are defined to represent the visual quality of PC frames. We also measure how much compressed data have to be transmitted in an FoV-adaptive streaming system if different coding schemes are used. The experiments are conducted based on a viewport trace [31] that is recorded from 26 participants watching the videos of the 8i dataset.

[6]https://github.com/MPEGGroup/mpeg-pcc-tmc2
[7]https://github.com/cwi-dis/cwi-pcl-codec

## 6.1 GOP Size

In patchVVC, only the color residuals are compressed for the P-patches. As a result, the compressed size of a P-patch is significantly smaller than that of an I-patch or a deviate patch. Intuitively, we want to define a large GOP size to maximize the compression ratio. However, as GOP size grows, it is increasingly challenging to find a P-patch in the last frame matching the I-patch, and we have to define more and more deviate patches in a patch group. This reduces the compression efficiency since the deviate patches are encoded without any compensation. As a result, the GOP size is critical to determine the compression ratio in patchVCC.

We thus conduct experiments to explore how the GOP size affects the compression efficiency of patchVVC. In the experiments, we change the GOP size from 3 to 30 and encode all videos in the 8i dataset. The bitrate of the compressed data and the visual quality are measured. In Figure 8, we report the normalized bitrates of various videos as the y-axis and the x-axis is the GOP size. We divide the average bitrate of a compressed video by its maximum compressed bitrate to calculate the normalized bitrate. In the figure, we can see an optimal GOP size for each test video that leads to the smallest compressed data. Setting the GOP size to the optimal value reduces the normalized bitrate by up to 47% (*Soldier*, GOP size 15). The optimal GOP sizes for *Longdress*, *Redandblack*, and *Loot* are 5, 5, and 15, respectively.

We also measure PSNR.GEO and PSNR.Y to check if the varying GOP size would affect the visual quality. The results are reported in Figure 9, where the y-axes are PSNR.Y and PSNR.GEO in dB. We can see that, for all videos, the visual quality is not significantly affected

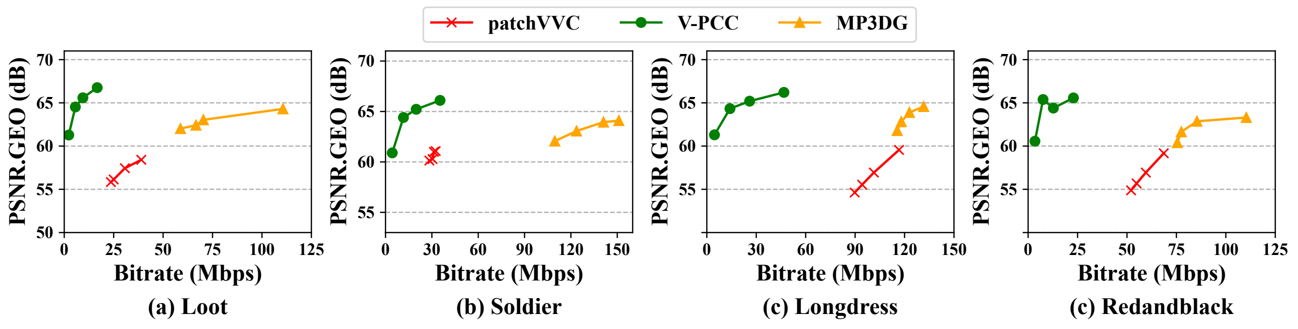

**Figure 10: The RD-curves (geometric fidelity) of various codecs on the 8i dataset**

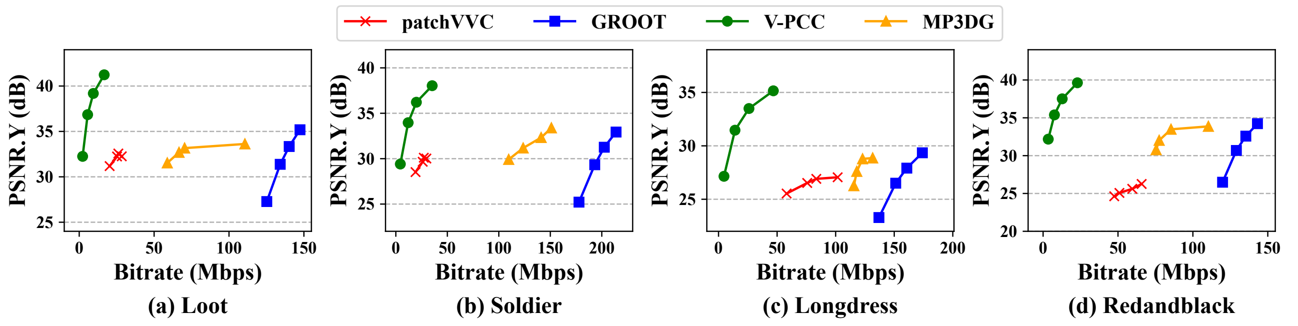

**Figure 11: The RD-curves (color fidelity) of various codecs on the 8i dataset**

by the GOP size. PSNR.GEO is degraded by up to 3.124 (*Loot*) and PSNR.Y is degraded by up to 1.121 (*Soldier*) when increasing the GOP size to 30.

## 6.2 RD-Curve

To more comprehensively evaluate patchVVC as a volumetric video encoder, we plot the RD-curves of patchVVC, V-PCC, MP3DG, and GROOT. As the visual quality of volumetric videos could be measured in PSNR.GEO and PSNR.Y, we plot two versions of the RD-curves to characterize both the geometric and color fidelity of various encoders. When measuring PSNR.GEO, we change the MSE threshold $T$ that controls how to detect the deviate patches from 5 to 20. When measuring PSNR.Y, we change the quantization step $Q$ from 10 (30) to 50 to compress colors (color residuals), which is used to balance the compression ratio and the color fidelity in RAHT. When we scale one of $T$ and $Q$, another is set to a fixed value. $T$ is set to 10 and $Q$ is set to 10 (30). For V-PCC, we measure four preset compression profiles *ctc-r1*, *ctc-r3*, *ctc-r4*, and *ctc-r5* (*ctc-r2* is discarded because it outputs almost the identical results as *ctc-r1*). For GROOT, we only plot its RD-curves on PSNR.Y because it compresses the geometric structures of PC frames in a lossless manner. We adjust the JPEG encoding quality from 10 to 70 in GROOT. For MP3DG, we vary the size of predictive macroblocks among 8, 16, 32, and 64.

The experimental results are reported in Figure 10 and Figure 11, respectively, where the x-axes are the compressed bitrate in Mbps and the y-axes are PSNR.GEO and PSNR.Y in dB. We individually plot the RD-curves of various encoders for all videos. As a 3D-based method, we can see that patchVVC is not as efficient as V-PCC in terms of the compression ratio, because V-PCC takes advantage of the techniques in the conventional 2D codecs that have been developed and optimized for decades. But since we effectively exploit the inter-frame redundancy in terms of the geometric structure and the colors, the compression ratio of patchVVC approaches V-PCC in cases. When compressing *Soldier* with varying quantization steps, the average bitrate of the compressed data of patchVVC is 17.90 Mbps while that of V-PCC is 25.51 Mbps, which is only 1.43x higher. As for GROOT, patchVVC reduces the average bitrate of the compressed data by 5.47x (*Loot*), 7.72x (*Soldier*), 1.95x (*Longdress*), and 2.36x (*Redandblack*). Compared to MP3DG, patchVVC reduces the average bitrate of the compressed data by 3.89x (*Loot*), 5.73x (*Soldier*), 1.99x (*Longdress*), and 1.68x (*Redandblack*). The relatively low compression ratios of the last two videos are due to the content. We notice that the non-rigid transformation is more common and the color variance is higher in the videos. In terms of visual quality, patchVVC reaches slightly lower geometric and color PSNR than V-PCC in most videos. In *Redandblack*, the average color PSNR is as low as 25.39. The reason is that the sharp color variance in this video is not friendly to our color interpolation method.

## 6.3 Decoding Latency

We aim at developing an efficient framework that matches the video decoding to a practical speed, e.g., 30 FPS. Thus, the decoding latency of patchVVC is also measured. We compare patchVVC,

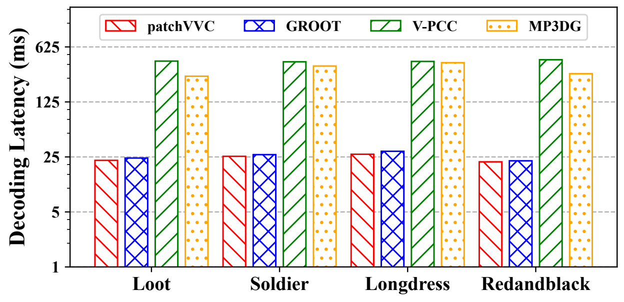

**Figure 12: The average decoding latency per frame of various codecs**

GROOT, V-PCC, and MP3DG, and the results are shown in Figure 12. We can observe that patchVVC has a similar decoding speed as GROOT, even though GROOT is accelerated by GPU. On average, patchVVC decodes a frame from 21.66 ms to 27.10 ms, and GROOT takes 20.84 ms to 29.49 ms. In contrast, V-PCC takes up to 413.81 ms to decode a frame because of its complex processing pipeline, which can hardly satisfy the requirement of fluent playback. We also measure the decoding speed of MP3DG. Even though 30 threads are launched for decoding, MP3DG still takes up to 394.05 ms to decode a frame.

## 6.4 PatchVVC in FoV-Adaptive Streaming

One of the strengths of patchVVC is that an FoV-adaptive streaming system could trivially select the patches in the user's FoV. In contrast, V-PCC can hardly support this because the video content is projected onto a few 2D frames. It is not straightforward to cut the compressed 2D frames according to the varying FoV. In this experiment, we evaluate the compressed data required to be transmitted in an FoV-adaptive streaming system for different coding schemes. For patchVVC, we set $T$ as 10 and $Q$ as 10 for I-frames and 30 for P-frames. V-PCC is set to use the *ctc-r5* compression profile.

We simulate an FoV-adaptive streaming system based on a public trace [31]. Before the experiment, we convert the trace data recorded in the Unity global coordinate system to those in the voxelized coordinate system. Because the 8i dataset uniformly maps 1.8 meters to [0, 1024) in integers, the new camera position $C_v^{x/y/z}$ is calculated by

$$C_v^{x/y/z} = \frac{C_u^{x/y/z}}{1.8} \times 1024$$

, where $C_u^{x/y/z}$ is the Unity global coordinate. We set the FoV to $100° \times 100°$.

In the experiment, we check if any point of a patch falls in the FoV. If so, the patch has to be transmitted, otherwise, the patch is excluded from the network delivery. In addition, we also exclude the occluded patches even if they are in the FoV. We determine a patch as occluded if all its points are occluded by at least one point from another patch. We name the scheme using FoV to exclude the invisible patches as *patchVVC-FoV* and the original scheme as *patchVVC-vanilla*. For V-PCC, all data have to be sent via the network. For GROOT, we also evaluate if it can benefit from the

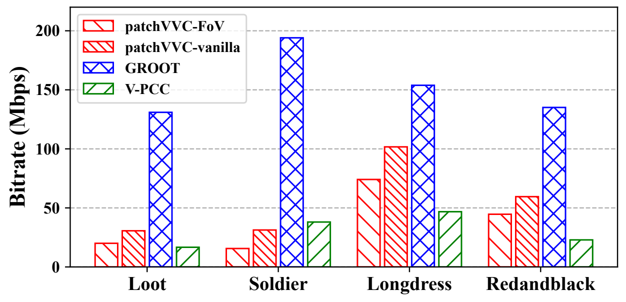

**Figure 13: The bitrates of streaming an object with FoV-adaptive methods**

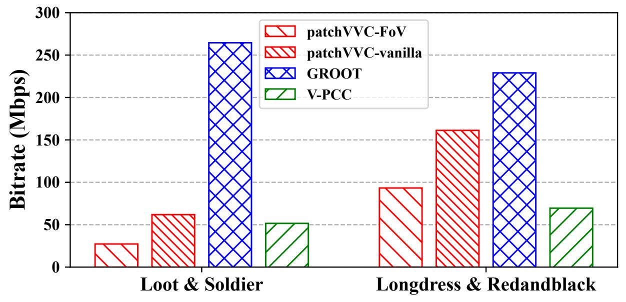

**Figure 14: The bitrates of streaming multiple objects with FoV-adaptive methods**

FoV-adaptive system. In GROOT, we first remove all points outside the FoV and then use the encoder to compress the trimmed point clouds. MP3DG is not optimized for FoV-adaptive streaming, so we will not discuss it here.

*6.4.1 Single Object.* We first conduct the experiment based on the original videos from the 8i dataset that contains only one object. We calculate how much data should be transmitted over the network and report the results in Figure 13, where the y-axis is the required bandwidth in Mbps and the x-axis is the video names. From the figure, we can see GROOT can hardly benefit from the FoV-adaptive calculation. It always has the highest bitrate even though we have removed points outside the FoV. For patchVVC, trimming the invisible data can always reduce the bandwidth required for streaming, and the average bandwidth saving is 17.19 Mbps. For V-PCC, though it has to send all of its data, the bandwidth required is still low due to its high compression efficiency. Overall, patchVVC-FoV further approaches the bandwidth requirement of V-PCC, and even in the case of streaming *Soldier*, patchVVC-FoV demands 58.96% less bandwidth compared to V-PCC.

*6.4.2 Multi-Objects Scene.* We also evaluate the coding schemes in multi-objects scenes. We generate two scenes with the 8i videos, one with *Loot* and *Soldier* in it, while another with *Longdress* and *Redandblack*. To form a multi-objects scene, we translate one object and combine it with the other object. Specifically, we translate *Soldier* and *Redandblack* by $[600, 0, 0]^T$, and combine them with

*Loot* and *Longdress*, respectively. We apply the trace of each object in the scene independently to the entire scene, and then report the average performance. For patchVVC, we additionally incorporate DBSCAN [10] to better split the patches in the I-frames. The evaluation results are shown in Figure 14. Similar trends are observed in single object compression, where GROOT has the worst performance. patchVVC-FoV demands 41.70% less bandwidth than V-PCC in *Loot & Soldier* while demands 34.23% more bandwidth in *Longdress & Redandblack*.

## 7 DISCUSSION

The volumetric video is composed of consecutive and highly similar frames, just like the 2D conventional videos. This emerging video type is also data-intensive and needs compression techniques. It is a natural idea to apply the compression techniques developed for 2D videos to the volumetric video, which have been deployed for decades [33, 36] and are still evolving [3]. Since pixels of volumetric videos are presented in a 3D space, one effective solution is compressing the pixels only after projecting them onto a planar, forming 2D patches embedded in rectangular frames. This solution has been standardized [28]. But treating the 3D objects as 2D patches discards one-dimensional information, and the movement of objects in the 3D space can not be completely captured during the compression process. We believe that directly realizing motion estimation and compensation in the 3D space is another potential solution to compressing volumetric videos since these techniques are originally developed for capturing object movements in the 3D space (though these movements are projected into the 2D space). In this work, we explore if we can build a pipeline to compress the volumetric videos as what has been done in compressing the conventional 2D videos. Despite the high-level idea being similar, challenges still exist in the implementation: 1) What is the basic unit of motion estimation? 2) How can we carry out the motion compensation? And 3) how is the compression pipeline evaluated?

### 7.1 Blocks *vs.* Patches

The motion estimation in 2D video codecs is based on macroblocks, which are small squares or rectangles. To mimic this, the point cloud frames could also be split into non-overlapped cubes [8, 22]. However, there are a few weaknesses in doing this: 1) The searching space of motion vectors is one dimension higher than that of 2D videos; 2) the volumetric video is relatively sparse, which means a cube with most empty voxels may match another empty cube even if the non-empty voxels from two cubes are completely different; 3) A cube might contain voxels from different objects, which have different motion vectors.

The current hardware captures point clouds from the surfaces of objects so that they form thin "shells" in the frames. We expect the 3D patches extracted from the object surfaces can be used as the basic unit of motion estimation. A patch is dense and, more importantly, is extracted from the neighboring area of the same object surface, which is highly likely composed of voxels with similar motion vectors. In this work, we employ a straightforward method to generate patches at a similar size from a single object. But advanced techniques are worth exploring to achieve better segmentation results.

### 7.2 Motion Estimation and Compensation

For a point cloud patch, ICP [2] is a feasible method to estimate the motion vector across frames. Though more accurate methods, e.g., NICP [29] and GICP [? ], have been proposed, they also demand high computation overhead. DNN-based methods [1, 35] further improve the accuracy of motion estimation but only in a low-throughput way. We aim at developing a codec matching the real-time playback speed in this work, thus adopting ICP. But these alternative methods could be further explored if a coding framework with a high compression ratio is desired.

The motion compensation between 3D patches is also challenging, especially the number of points is not identical and their positions are not aligned. In patchVVC, we suggest constructing a common patch to reach alignment among a group of patches. This reduces redundancy in both the geometric domain and the color domain. But this method is short of flexibility. In future work, we prefer developing a method that could more flexibly make a trade-off between visual quality and compression ratio.

### 7.3 Quality Metrics

We adopt PSNR.Y and PSNR.GEO as the quality metrics in patchVVC, which are also the MPEG standard [27]. However, these metrics are extended from the ones used in 2D images, and can not completely adapt to 3D objects. For example, it is meaningless to evaluate the quality of the occluded part of the point cloud in practice. Additionally, the size of points when rendering also affects the visual quality of a point cloud but this is not captured by the current metrics. Thus, a quality metric that more accurately characterizes the user's quality of experience is critical in building a satisfactory compression pipeline. With such a metric, it becomes easy to understand if a compression technique could impair the user's quality of experience.

## 8 CONCLUSION

In this paper, we propose patchVVC, a real-time 3D-based compression framework for volumetric videos. patchVVC reaches a high compression ratio by effectively exploiting the inter-frame redundancy in both the geometric and color domains. Moreover, patchVVC is designed based on splitting the point cloud frames into patches and is friendly to both the modern multi-threading architecture and the FoV-adaptive streaming systems. This leads to the real-time decoding speed and low bandwidth demands of deploying patchVVC. The evaluation shows that patchVVC achieves comparable compression ratios to V-PCC and similar decoding speeds to GROOT. Also, patchVVC outperforms MP3DG on both compression ratios and decoding speeds.

## ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive comments. This work has been partially supported by the grants from the National Natural Science Foundation of China (Grant No. 62102229 and No. 62122042), and the Natural Science Foundation of Shandong Province, China (Grant No. ZR202206140010).

## REFERENCES

[1] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. 2019. Pointnetlk: Robust & efficient point cloud registration using pointnet. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 7163–7172.

[2] Paul J Besl and Neil D McKay. 1992. Method for Registration of 3-D Shapes. *Sensor fusion IV: control paradigms and data structures* 1611 (1992), 586–606.

[3] Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J. Sullivan, and Jens-Rainer Ohm. 2021. Overview of the Versatile Video Coding (VVC) Standard and its Applications. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 10 (2021), 3736–3764.

[4] Seonghwa Choi, Anh-Duc Nguyen, Jinwoo Kim, Sewoong Ahn, and Sanghoon Lee. 2019. Point Cloud Deformation for Single Image 3d Reconstruction. In *2019 IEEE International Conference on Image Processing.* 2379–2383.

[5] Cisco. 2020. Cisco Annual Internet Report (2018-2023) White Paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. Online; accessed 31 March 2022.

[6] Yann Collet and Murray Kucherawy. 2018. Zstandard Compression and the Application/Zstd Media Type. *RFC 8478* (2018).

[7] Ricardo L De Queiroz and Philip A Chou. 2016. Compression of 3D point clouds Using a Region-Adaptive Hierarchical Transform. *IEEE Transactions on Image Processing* 25, 8 (2016), 3947–3956.

[8] Ricardo L de Queiroz and Philip A Chou. 2017. Motion-Compensated Compression of Dynamic Voxelized Point Clouds. *IEEE Transactions on Image Processing* 26, 8 (2017), 3886–3895.

[9] Eugene d'Eon, Bob Harrison, Taos Myers, and Philip A Chou. 2017. 8i Voxelized Full Bodies–A Voxelized Point Cloud Dataset. *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006* 7 (2017), 8.

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *The Second International Conference on Knowledge Discovery and Data Mining*, Vol. 96. 226–231.

[11] Pierre-Marie Gandoin and Olivier Devillers. 2002. Progressive Lossless Compression of Arbitrary Simplicial Complexes. *ACM Transactions on Graphics* 21, 3 (2002), 372–379.

[12] Serhan Gül, Dimitri Podborski, Thomas Buchholz, Thomas Schierl, and Cornelius Hellge. 2020. Low-Latency Cloud-based Volumetric Video Streaming using Head Motion Prediction. In *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video.* 27–33.

[13] Serhan Gül, Dimitri Podborski, Jangwoo Son, Gurdeep Singh Bhullar, Thomas Buchholz, Thomas Schierl, and Cornelius Hellge. 2020. Cloud Rendering-based Volumetric Video Streaming System for Mixed Reality Services. In *Proceedings of the 11th ACM Multimedia Systems Conference.* 357–360.

[14] Dorina hanou, Philip A Chou, and Pascal Frossard. 2016. Graph-based Compression of Dynamic 3D Point Cloud Sequences. *IEEE Transactions on Image Processing* 25, 4 (2016), 1765–1778.

[15] Mohammad Hosseini and Christian Timmerer. 2018. Dynamic Adaptive Point Cloud Streaming. In *Proceedings of the 23rd Packet Video Workshop.* 25–30.

[16] Yan Huang, Jingliang Peng, C-C Jay Kuo, and M Gopi. 2006. Octree-based Progressive Geometry Coding of Point Clouds. In *Proceedings of the 3rd Eurographics.* 103–110.

[17] K Krishna and M Narasimha Murty. 1999. Genetic K-means Algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29, 3 (1999), 433–439.

[18] Kyungjin Lee, Juheon Yi, Youngki Lee, Sunghyun Choi, and Young Min Kim. 2020. GROOT: A Real-time Sstreaming System of High-Fidelity Volumetric Videos. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking.* 1–14.

[19] Li Li, Zhu Li, Shan Liu, and Houqiang Li. 2019. Occupancy-Map-Based Rate Distortion Optimization for Video-based Point Cloud Compression. In *2019 IEEE International Conference on Image Processing.* 3167–3171.

[20] Li Li, Zhu Li, Vladyslav Zakharchenko, Jianle Chen, and Houqiang Li. 2019. Advanced 3D Motion Prediction for Video-based Dynamic Point Cloud Compression. *IEEE Transactions on Image Processing* 29 (2019), 289–302.

[21] Rufael Mekuria, Kees Blom, and Pablo Cesar. 2016. Design, Implementation, and Evaluation of a Point Cloud Codec for Tele-Immersive Video. *IEEE Transactions on Circuits and Systems for Video Technology* 27, 4 (2016), 828–842.

[22] Rufael Mekuria and Pablo Cesar. 2016. MP3DG-PCC, Open Source Software Framework for Implementation and Evaluation of Point Cloud Compression. In *Proceedings of the 24th ACM International Conference on Multimedia.* 1222–1226.

[23] Jounsup Park, Philip A Chou, and Jenq-Neng Hwang. 2019. Rate-Utility Optimized Streaming of Volumetric Media for Augmented Reality. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 1 (2019), 149–162.

[24] 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.* 135–140.

[25] ITUR Rec. 1995. BT 601: Studio encoding parameters of digital television for standard 4: 3 and wide-screen 16: 9 aspect ratios. *ITU-R Rec. BT* 656 (1995).

[26] Ruwen Schnabel and Reinhard Klein. 2006. Octree-based Point-Cloud Compression. In *Proceedings of the 3rd Eurographics.* 111–120.

[27] Sebastian Schwarz, Gaëlle Martin-Cocher, David Flynn, and Madhukar Budagavi. 2018. Common Test Conditions for Point Cloud Compression. *Document ISO/IEC JTC1/SC29/WG11 w17766, Ljubljana, Slovenia* (2018).

[28] Sebastian Schwarz, Marius Preda, Vittorio Baroncini, Madhukar Budagavi, Pablo Cesar, Philip A Chou, Robert A Cohen, Maja Krivokuća, Sébastien Lasserre, Zhu Li, et al. 2018. Emerging MPEG standards for Point Cloud Compression. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 1 (2018), 133–148.

[29] Jacopo Serafin and Giorgio Grisetti. 2015. NICP: Dense Normal based Point Cloud Registration. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems.* 742–749.

[30] Yiting Shao, Zhaobin Zhang, Zhu Li, Kui Fan, and Ge Li. 2017. Attribute Compression of 3D Point Clouds Using Laplacian Sparsity Optimized Graph Transform. In *2017 IEEE Visual Communications and Image Processing.* 1–4.

[31] Subramanyam Shishir, Hanjalic Alan, and Cesar Pablo. 2020. User Centered Adaptive Streaming of Dynamic Point Clouds with Low Complexity Tiling. In *Proceedings of the 28th ACM International Conference on Multimedia.* 3669–3677.

[32] Hubert P. H. Shum, Edmond S. L. Ho, Yang Jiang, and Shu Takagi. 2013. Real-Time Posture Reconstruction for Microsoft Kinect. *IEEE Transactions on Cybernetics* 43, 5 (2013), 1357–1369.

[33] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology* 22, 12 (2012), 1649–1668.

[34] Jeroen van der Hooft, Tim Wauters, Filip De Turck, Christian Timmerer, and Hermann Hellwagner. 2019. Towards 6DoF HTTP Adaptive Streaming Through Point Cloud Compression. In *Proceedings of the 27th ACM International Conference on Multimedia.* 2405–2413.

[35] Yue Wang and Justin M Solomon. 2019. Deep Closest Point: Learning Representations for Point Cloud Registration. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 3523–3532.

[36] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. 2003. Overview of the H. 264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology* 13, 7 (2003), 560–576.

[37] Yiqun Xu, Wei Hu, Shanshe Wang, Xinfeng Zhang, Shiqi Wang, Siwei Ma, Zongming Guo, and Wen Gao. 2020. Predictive Generalized Graph Fourier Transform for Attribute Compression of Dynamic Point clouds. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 5 (2020), 1968–1982.

[38] Xiangyu Yue, Bichen Wu, Sanjit A. Seshia, Kurt Keutzer, and Alberto L. Sangiovanni-Vincentelli. 2018. A LiDAR Point Cloud Generator: From a Virtual World to Autonomous Driving. In *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval.* 458–464.

[39] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. 2022. YuZu : Neural-Enhanced Volumetric Video Streaming. In *19th USENIX Symposium on Networked Systems Design and Implementation.* 137–154.

[40] Cha Zhang, Dinei Florencio, and Charles Loop. 2014. Point Cloud Attribute Compression with Graph Transform. In *2014 IEEE International Conference on Image Processing.* 2066–2070.

[41] Wenjie Zhu, Zhan Ma, Yiling Xu, Li Li, and Zhu Li. 2021. View-Dependent Dynamic Point Cloud Compression. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 2 (2021), 765–781.