Attribute-Specific Compression Techniques for 3D Gaussian Point Cloud Sequences

Zhehao Shen, Shouchen Zhou
School of Information Science and Technology
ShanghaiTech University
{shenzhh, zhoushch}@shanghaitech.edu.cn
2021533110, 2021533042

***Abstract*—Compression plays a pivotal role in optimizing the storage and transmission of digital data, particularly in the era of immersive media. This project explores contemporary data compression methods with a focus on volumetric video, an innovative medium that provides an unprecedented immersive experience. We begin by reviewing a variety of compression techniques, including entropy-based methods, dictionary-based algorithms, transform coding, and machine learning-based approaches. Building on these insights, we apply several advanced compression strategies to optimize the HiFi4G [1] framework, which is used for high-fidelity human performance rendering. Specifically, we leverage the Range Asymmetric Numerical System (RANS) [2] for efficient residual encoding, the WebP format for image data compression, and K-Nearest Neighbors (KNN)-based techniques for point cloud compression. These methods collectively reduce storage requirements and enhance data processing efficiency, enabling real-time rendering of dynamic 4D human performances in virtual and augmented reality environments. Our results demonstrate that the integration of these compression techniques significantly improves the scalability and efficiency of high-fidelity rendering systems, making them more practical for deployment in VR/AR applications.**

***Index Terms*—Data compression, image compression, entropy coding, point cloud compression, WebP, RANS, 4D rendering, Gaussian splatting**

I. Introduction

Compression holds a fundamental position in the field of information theory. At its core, compression is governed by Source Coding Theory, with Shannon's Source Coding Theorem laying the theoretical foundation for compression tasks. This theorem tells us that the minimum average code length for representing information is closely tied to its entropy. Entropy measures the uncertainty or the amount of information in a source. Therefore, the goal of source coding is to identify an encoding scheme that minimizes the average length of each message, bringing it as close as possible to the entropy of the source.

Beyond its significant theoretical value in information theory, compression has become even more vital in the era of digital media. The compression of images, audio, and text allows digital content to be transmitted to our mobile devices more efficiently and with minimal loss. Video streaming platforms (such as YouTube and Netflix) and VoIP communication systems (such as Skype) rely heavily on compression technologies. By reducing the bandwidth required for data transmission, compression ensures smoother user experiences.

In this context, our project primarily surveys various compression methods and applies them to volumetric video — an innovative, immersive video format that allows viewers to experience content from any angle during playback. Our goal is to compress this cutting-edge medium so that it can be seamlessly integrated into mobile phones and VR devices, much like images, text, and audio. This will provide users with a novel form of digital media, offering an entirely new level of immersion and interaction.

This project is rooted in a thorough exploration of contemporary data compression methods. We began by reviewing a broad spectrum of compression techniques, including entropy-based methods, dictionary-based algorithms, transform coding, and machine learning-based compression. These methods provide a wide range of strategies for reducing data size while preserving the integrity of the original information.

Based on the insights from this review, we focused on applying several advanced compression strategies to optimize the HiFi4G framework [1], specifically targeting image and point cloud data. Among the methods explored, the Range Asymmetric Numerical System (RANS) [2] was employed for efficient residual encoding, significantly reducing the redundancy in data representation. The WebP image compression format, known for its high compression efficiency and support for lossy and lossless compression, was utilized to compress the image data generated in HiFi4G. Additionally, we incorporated K-Nearest Neighbors (KNN)-based compression techniques to efficiently handle point cloud data, ensuring that the spatial characteristics of dynamic scenes were preserved while minimizing data size.

By integrating these compression methods, we achieved a substantial reduction in storage requirements for HiFi4G's 4D human performance rendering. The combination of RANS for residual compression, WebP for image data, and KNN for point cloud compression not only reduced the data footprint but also facilitated faster processing and real-time rendering without compromising the quality of the rendered human performances. This project demonstrates that through careful selection and implementation of state-of-the-art compression techniques, it is possible to enhance the efficiency and scalability of high-fidelity rendering systems, making them more suitable for practical deployment in VR/AR environments.

Overall, our contributions can be summarized into the following points:

- Developed a novel, attribute-specific compression method tailored specifically for efficient volumetric video representation.
- Integrated compression and decompression methods into a mobile platform, enabling the rendering of long sequence volumetric videos on mobile devices.
- Advanced the field of mobile media by enhancing the accessibility and interactivity of volumetric media in portable formats, pushing the boundaries of mobile media consumption.

II. Related Work

According to Parkinson's First Law [3], the data growth rate consistently outpaces the rate of technological advancement. This phenomenon exacerbates challenges related to managing large datasets, particularly in terms of storage and transmission. In this context, data compression emerges as a highly effective solution to mitigate these issues, offering a means to handle the ever-expanding volume of data efficiently.

A. Review of the Review

We have learned Huffman coding and Lemple-Ziv coding during the classes. To gain a comprehensive understanding of various data compression techniques and their applications, we read the review [4].

Huffman coding minimizes average code length by assigning variable-length codes based on symbol frequencies, while Lempel-Ziv coding reduces redundancy through dynamically constructed dictionaries, both serving as foundational methods for efficient lossless data compression.

Building upon the foundational concepts of Huffman coding and Lempel-Ziv coding that we have explored in class, it becomes clear that while these algorithms provide efficient solutions for basic lossless data compression, their limitations in handling more complex data types and achieving higher compression ratios necessitate the exploration of additional methods. The review highlights the importance of extending beyond these classic techniques, as the explosive growth in data volume and diversity demands more sophisticated approaches. Algorithms like arithmetic coding, Burrows-Wheeler Transform (BWT), and transform-based methods such as Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT) offer enhanced performance for specific applications, such as multimedia compression and large-scale datasets. These advanced techniques are designed to address the nuances of different data structures, achieving better compression ratios and adapting to the varying redundancy patterns found in real-world data.

Some details of the classification of data compression algorithms can be seen in Fig. 1.

III. Preliminaries

The previous discussion highlighted various compression techniques for different types of data, including text, images, audio, video, and other traditional forms of digital media. These compression methods have significantly contributed to enhancing our digital lives. In recent years, however, 3D and 4D content has rapidly emerged, reshaping society's expectations and applications of digital landscapes. Volumetric video, as an innovative video format, represents a transformative leap in visual media, allowing viewers to experience content from any perspective and providing an immersive virtual experience.

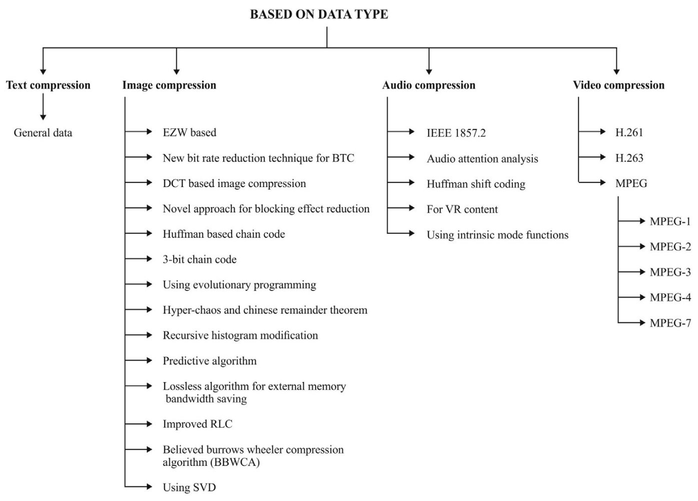
BASED ON DATA TYPE
Text compression
General data
Image compression
EZW based
New bit rate reduction technique for BTC
DCT based image compression
Novel approach for blocking effect reduction
Huffman based chain code
3-bit chain code
Using evolutionary programming
Hyper-chaos and chinese remainder theorem
Recursive histogram modification
Predictive algorithm
Lossless algorithm for external memory bandwidth saving
Improved RLC
Believed burrows wheeler compression algorithm (BBWCA)
Using SVD
Audio compression
IEEE 1857.2
Audio attention analysis
Huffman shift coding
For VR content
Using intrinsic mode functions
Video compression
H.261
H.263
MPEG
MPEG-1
MPEG-2
MPEG-3
MPEG-4
MPEG-7

Fig. 1. Compression algorithms are categorized based on the type of data they process, including text, image, audio, and video data.

A. 3D Gaussian Splatting

3D Gaussian Splatting [5], as a point cloud-based rendering method, begins with a collection of images capturing a static scene, their corresponding camera parameters, and a sparse point cloud generated through Structure-from-Motion (SfM) [6]. It then leverages a series of Gaussian ellipsoids to reconstruct the scene with high quality. The method represents a scene as a set of 3D Gaussian primitives, each centered at (x, y, z) and defined by attributes such as opacity (for alpha-blending), covariance (anisotropic scale (s_x, s_y, s_z) and rotation (r_x, r_y, r_z, r_w)), and spherical harmonics ($SH_0 \sim SH_{45}$) for diffuse directional radiance. Currently, numerous methods, such as those from HiFi4G [7], 3DGStream [8], V^3 [9] and DualGS [10], utilize Gaussians to form volumetric video representations, extending the static scene representation of Gaussians to dynamic scenarios. However, dynamic Gaussian point clouds come with extremely high storage overhead. For instance, representing a static digital human scene typically requires around 120,000 points, where each point stores 56 float attributes, including position (x, y, z), rotation (r_x, r_y, r_z, r_w), opacity, scale (s_x, s_y, s_z), and 45 spherical harmonics ($SH_0 \sim SH_{44}$). A 10-second volumetric video, consisting of 300 frames, would require approximately $300 \times 120,000 \times 56$ floats, amounting to around **7.5 GB**. This substantial storage demand poses a significant challenge to the practical deployment of volumetric video technology.

B. Rendering with 3D Gaussians

3D Gaussian ellipsoids are projected into 2D frame space using camera parameters, making them differentiable and enabling efficient rendering through 2D splatting with α-blending. In radiance field modeling, the directional appearance (color) c is encoded via spherical harmonics (SH). Using tile-based rasterization, the pixel color C is calculated by depth-ordering c_i and aggregating as follows:

$$C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j) \tag{1}$$

Here, α_i is computed by combining the Gaussian weight with the opacity α of the Gaussian sphere. Once rasterized, the color loss can be formulated as:

$$\mathcal{L}_{\text{color}} = \|C - C_{\text{gt}}\|^2 \tag{2}$$

IV. Principles of On-the-Shelf Compression Methods

In this chapter, we will introduce the fundamental principles and mathematical derivations of two on-the-shelf compression algorithms, RANS and WebP, used in our attribute-specific compression techniques.

A. RANS Coding

To leverage the distribution for real-time encoding and decoding, we apply the Range Asymmetric Numerical System (RANS) [2]. RANS enhances compression by taking advantage of the distribution's skewness.

Asymmetric Numeral Systems (ANS) is an entropy encoding method used in data compression developed by Jaroslaw Duda [11]. It is an encoding method that approaches the theoretical value of entropy coding infinitely, and its essential operation is to use a decimal of [0,1) to represent the final encoding result.

Huffman coding maps individual symbols to codewords based on their probabilities and is optimal for independent symbols under the prefix-free constraint. In contrast, RANS coding maps sequences of symbols to a single codeword, leveraging the overall probability distribution of the sequence. As a result, RANS achieves higher compression efficiency and approaches the theoretical entropy limit.

1) Encoding a binary string to a natural number when bits are uniformly distributed: Suppose an integer x with n bits in its binary form $b_1 b_2 \dots b_n$, where $p(b_i = 1) = 0.5$. And suppose that $x_0 = 1$, then the encoder C is that

$$x_{i+1} = C(x_i, b_{i+1}) = 2x_i + b_{i+1}$$

Which means that x_n equals to x.
To recover the message, a decoder D is needed:

$$(x_i, b_{i+1}) = D(x_{i+1}) := \left(\left\lfloor \frac{x_{i+1}}{2} \right\rfloor, x_{i+1} \bmod 2 \right)$$

2) Redefining the Odds and Evens: Suppose that each bit is not uniformly distributed, i.e. $p(b_i = 1) = p_1 = p$. Then suppose that the optimal encoder is C_{opt}.

Then:

$$\begin{aligned} H(x_{i+1}) &= H(C_{\text{opt}}(x_i, b_{i+1})) \\ &= H(x_i) + H(b_{i+1}) \\ &= \log(x_i) - \log(p_{b_{i+1}}) \\ &= \log\left(\frac{x_i}{p_{b_{i+1}}}\right) \\ \Rightarrow C_{\text{opt}}(x_i, b_{i+1}) &\approx \frac{x_i}{p_{b_{i+1}}} \end{aligned}$$

The second equation is set to behold, because we want to minimize the length after encoding, i.e. minimize $H(x_{i+1})$. And since x_i has $\log(x_i)$ bits of information, to achieve the minimum, the case is that b_i has $-\log(p_{b_{i+1}})$ bits of information.

3) Range Asymmetric Numerical System(RANS): Then we expand the binary to s-base; we need to replace all b_i with s_i. Theoretically, we can have arbitrary real numbers for the probability distribution of our alphabet. But a restriction that we'll place is that we'll quantize the probability distribution in 2^n chunks, which is "Ranged". Within this range, the number of times the symbol s appears is f_s. Then the probability could be formed as $p_s = \frac{f_s}{2^n}$. Then the encoding formula C is:

$$C(x_i, s_{i+1}) = \left\lfloor \frac{x_i}{f_i} \right\rfloor \cdot 2^n + \sum_{k=0}^{s-1} f_k + (x_i \bmod f_s)$$

Where x_0 is set to be 2^n. As for the decoder D, then

$$s_{i+1} = \min \ s : x < \sum_{k=0}^{s} f_k$$

$$x_i = D(x_{x+1}) = f_s \cdot \left\lfloor \frac{x_{i+1}}{2^n} \right\rfloor - \sum_{k=0}^{s-1} f_k - (x_{i+1} \bmod 2^n)$$

The meanings for the encoding part are:

- $\left\lfloor \frac{x_i}{f_i} \right\rfloor \cdot 2^n$: finds the right 2^n range (recall that we have a repeating pattern every 2^n natural numbers). If f_s is small, say $f_s = 1$, then it only appears once every 2^n range. If f_s is large, then we would expect to see f_s numbers mapped to every 2^n range.
- $CDF[s] = \sum_{k=0}^{s-1} f_k$ finds the of f_s et within the 2^n range for the current symbol s, i.e. all s symbols will be grouped together within this range starting here.
- $(x_i \bmod f_s)$ finds the precise location within this sub-range (which has precisely f_s spaces allocated for it).

The decoding is just the reverse operation of the encoding.

4) An example of RANS coding: Using the alphabet ['a', 'b', 'c'] with quantization $n = 3$ and distribution $[f_a, f_b, f_c] = [5, 2, 1]$ ($CDF[s] = [0, 5, 7, 8]$), let's encode the string "abc". We need to start with $x_0 = 2^n = 8$:

Encoding process:

$$\begin{aligned} x_1 &= C(x_0, a) = \left\lfloor \frac{x_0}{f_a} \right\rfloor \cdot 2^3 + CDF[a] + (x_0 \bmod f_a) \\ &= \left\lfloor \frac{8}{5} \right\rfloor \cdot 8 + 0 + (8 \bmod 5) = 11 \\ x_2 &= C(x_0, b) = \left\lfloor \frac{x_1}{f_b} \right\rfloor \cdot 2^3 + CDF[b] + (x_1 \bmod f_b) \\ &= \left\lfloor \frac{11}{2} \right\rfloor \cdot 8 + 5 + (11 \bmod 2) = 46 \\ x_3 &= C(x_0, c) = \left\lfloor \frac{x_2}{f_c} \right\rfloor \cdot 2^3 + CDF[c] + (x_2 \bmod f_c) \\ &= \left\lfloor \frac{46}{1} \right\rfloor \cdot 8 + 7 + (46 \bmod 1) = 375 \end{aligned}$$

Decoding process:

$$\begin{aligned} s_2 &= \min \ s : \left((x_3 = 375) \bmod 2^3\right) < \sum_{k=0}^{s} f_k = c \\ x_2 &= D(x_3) = f_c \cdot \left\lfloor \frac{x_3}{8} \right\rfloor - CDF[c] + (x_3 \bmod 8) \\ &= 1 \cdot \left\lfloor \frac{375}{2^3} \right\rfloor - 7 + (375 \bmod 8) = 46 \\ s_1 &= \min \ s : \left((x_2 = 46) \bmod 2^3\right) < \sum_{k=0}^{s} f_k = b \\ x_1 &= D(x_2) = f_b \cdot \left\lfloor \frac{x_2}{8} \right\rfloor - CDF[b] + (x_2 \bmod 8) \\ &= 2 \cdot \left\lfloor \frac{46}{2^3} \right\rfloor - 5 + (46 \bmod 8) = 11 \\ s_0 &= \min \ s : \left((x_1 = 11) \bmod 2^3\right) < \sum_{k=0}^{s} f_k = a \\ x_0 &= D(x_1) = f_a \cdot \left\lfloor \frac{x_1}{8} \right\rfloor - CDF[a] + (x_1 \bmod 8) \\ &= 5 \cdot \left\lfloor \frac{11}{2^3} \right\rfloor - 0 + (11 \bmod 8) = 8 \end{aligned}$$

TABLE I
COMPRESSION EFFECT COMPARISON BETWEEN HUFFMAN CODING AND RANS CODING WITHOUT QUANTIZATION

Methods	No Compression	Huffman Coding	RANS Coding
Memory(KB) ↓	4745	4413	**3864**

To verify that RANS coding works, we extracted features from the point cloud dataset, including the position and rotation of the points. Then we tested the size of the files after Huffman Coding or RANS Coding and tested the size of the files to see that RANS Coding has a better compression effect. The result can be seen in Table I. This is a lossless compression, that is without quantization. The rotation and positions are floating numbers, which typically have high entropy and uniform byte distribution, which makes it difficult for Huffman encoding to find enough repeated or high-frequency symbols, thereby limiting its compression efficiency. To raise a better compression effect, we would add quantization before compression when application. More details can be seen in section IV.

B. *WebP Coding*

1) Introduction to WebP coding: WebP [12] is a compression algorithm developed by Google, which is an image file format that supports both lossy and lossless compression, while supporting transparent channels under lossy conditions. After lossless compression, WebP has about 30% less file size than PNG files, greatly reducing space overhead. But the reason why it is not a universal compression method is that encoding WebP files of the same quality requires more computing resources and only a few websites support WebP decoding functionality. The details of WebP can be seen at https://developers.google.com/speed/webp/docs/compression.

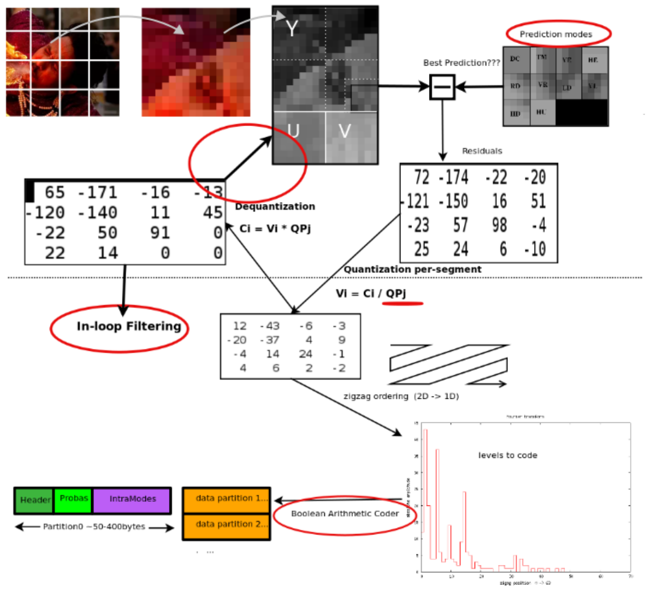

Fig. 2. The steps involved in WebP lossy compression. The differentiating features compared to JPEG are circled in red. The major steps are: MacroBlocking, prediction, DCT, quantization, and arithmetic entropy encoding.

2) Major steps: The major steps for WebP can be seen in the diagram in Fig. 2.

1. MacroBlocking. WebP's lossy compression algorithm leverages the same predictive techniques used by the VP8 [13] video codec('coder-decoder'), which is based on block prediction. Specifically, VP8 divides each frame into smaller units called macroblocks. Within each macroblock, the encoder predicts redundant motion and color information by referencing previously processed blocks, a process known as predictive coding. There are three types of blocks: 4×4 luma, 16×16 luma, and 8×8 chroma.

If the image data before compression is in RGB format, it needs to be converted to YUV format first, where Y represents the luminance component and UV represents the chrominance component. The reason for converting to YUV format is that human vision is much more sensitive to brightness than chromaticity, so reducing the storage of chromaticity data can save data space without causing too much impact on the visual effect.

2. Prediction. In this context, a "key" frame refers to the use of only the pixels already decoded in the immediate spatial vicinity of a given macroblock, with the encoder filling in

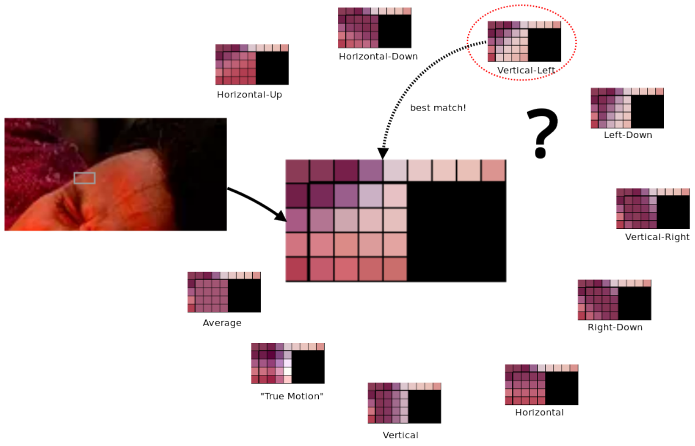

Fig. 3. The different prediction modes used in WebP lossy compression: H_PRED (horizontal prediction), V_PRED (vertical prediction), DC_PRED (DC prediction), and TM_PRED (TrueMotion prediction).

the missing pixel values. This predictive method reduces the amount of redundant data that needs to be stored. There are a total of 4 modes of prediction types, which can details seen in Fig. 3.

3. DCT. Once the prediction is made, the difference between the predicted and actual pixel values—referred to as the residual—is calculated. This residual data is much smaller in size and can be compressed more efficiently. To further optimize compression, the residual is subjected to a mathematically invertible transformation, typically the Discrete Cosine Transform (DCT), which is designed to concentrate the signal's energy into fewer coefficients. As we know in digital image processing, DCT generates a frequency coefficient matrix, with the coefficients in the upper left having the largest amplitude and those in the lower right having the smallest amplitude. The smaller the amplitude value, the higher the frequency. Most of the image information is located in the upper left area.

4. Quantization. As a result, many of the residual coefficients are zero or near zero, allowing for highly efficient compression. The residual values are then quantized, a process where precision is reduced by discarding less important information.

5. Arithmetic entropy encoding. Entropy coding is applied to achieve the final compressed output. Notably, the only step that introduces lossy compression is quantization, where bits are discarded (as indicated by the division by the quantization parameter QP_j), while all preceding steps, including the DCT transform, are lossless.

The main reason that WebP has a better compression rate than JPEG is the 'prediction'. Block adaptive quantization makes a big difference, too. Filtering helps at mid/low bitrates. Boolean arithmetic encoding provides 5%-10% compression gains compared to Huffman encoding.

V. Methods

Our ultimate goal is to draw inspiration from traditional compression techniques for digital media formats such as text, audio, images, and videos, to design an efficient and low-loss compression method for volumetric videos represented as 3D Gaussian point cloud sequences. This approach aims to facilitate the practical deployment of such immersive data formats across multiple platforms, including head-mounted displays and mobile devices, enabling broader accessibility and adoption of this technology. Recent studies focus on the compression of static Gaussian point cloud representations. SOG [14] employs image codec techniques to compress Gaussian attributes mapped onto 2D grids. The techniques in Compact3D [15], C3DGS [16], and Compact-3DGS [17] revolve around vector quantization and entropy encoding. Scaffold-GS [18] leverages anchor points to effectively reduce redundant Gaussians. Similarly, RDO-Gaussian [19] and Reduced3DGS [20] utilize redundant Gaussian culling combined with vector quantization for compression. In the field of Gaussian point cloud sequence compression, a notable work is HiFi4G [1], which achieves a 26x compression ratio by leveraging techniques such as quantization, residual compression, and RANS entropy coding. We requested the checkpoint of the 3D Gaussian point cloud sequences data obtained during training from the authors of HiFi4G [1]. Based on this, we designed an attribute-specific compression technique, as shown in Fig. 4, achieving a compression ratio of **xx** times.

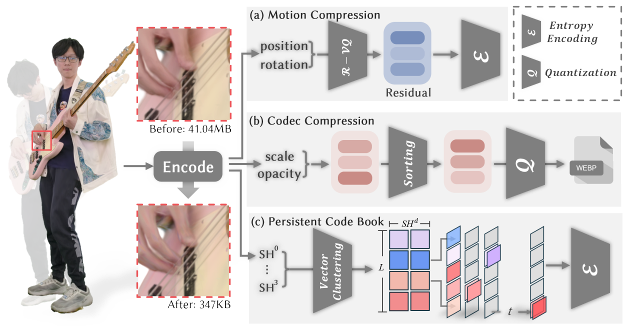

Fig. 4. The pipeline of our attribute-specific compression technique.

a) Residual Compression.: As mentioned in C3DGS [16] and HiFi4G [1], the precision of Gaussian position plays a crucial role in the quality of the scene, where even minor errors can severely impact rendering quality. Therefore, they opt for high-bit quantization. Following the entropy coding strategy of HiFi4G, we implemented a residual-based quantization compression method for position and rotation attributes. We further employ Ranged Asymmetric Numerical System(RANS) encoding for lossless compression.

b) Codec Compression.: While residual compression can be applied to opacity and scaling parameters, the large number of Gaussians significantly increases the storage demand for compressed opacity and scaling data. To strike a balance between data accuracy and storage efficiency, we exploit the spatiotemporal relationships of these two Gaussian attributes. Leveraging a temporal regularizer, we organize the opacity and scaling parameters into separate Look-Up Tables (LUTs) for efficient encoding. Specifically, we construct a 2D LUT where the height represents the number of Gaussian point cloud sequences, and the width corresponds to the segment frame length. To improve 2D consistency, the LUT rows are sorted by their average values. In this way, we effectively

TABLE II
QUANTITATIVE COMPARISON WITH STATIC COMPRESSION METHODS.

Method	PSNR ↑	SSIM ↑	LPIPS ↓	Running Time ↓	Storage(MB / frame) ↓
Compact-3DGS [17]	36.021	0.991	0.0193	14m 21s	6.710 (7.01x)
C3DGS [16]	35.823	0.991	0.0189	16m 09s	2.129 (22.07x)
Ours	35.243	0.989	0.0221	12m 12s	1.159 (40.552x)

transform the data into an image-like format, allowing us to directly apply image compression algorithms such as WebP or JPEG for efficient storage.

c) Persistent CodeBook.: The color attributes account for the majority of the storage cost, occupying 48 out of 59 parameters. Efficient compression of these attributes is essential for significant storage savings. However, conventional residual or codec-based compression methods still require considerable overhead to store these coefficients.

To address this, we propose a novel compression strategy: a persistent codebook that leverages the temporal consistency of Gaussian sequences' SH parameters, achieving up to 360-fold compression. Specifically, we apply K-Means clustering to the d-order ($d = 0, 1, 2, 3$) SH coefficients across all frames in a segment. The codebook $\mathcal{Z}_d$ is initialized with a uniform distribution and iteratively updated by randomly sampling batches of d-order coefficients. After optimization, we obtain four codebooks, each with a length L (set to 8192 in our implementation).

Using these codebooks, the SH attributes are compactly encoded into SH indices as follows:

$$\tau_{i,t}^{d} = \underset{k \in \{1,\ldots,L\}}{\operatorname{argmin}} \left\| \mathcal{Z}_d[k] - \mathcal{C}_{i,t}^{d} \right\|_2^2$$

where $\tau_{i,t}^{d}$ represents the d-order SH index for Gaussian i at frame t. The compressed SH coefficients $\hat{\mathcal{C}}_{i,t}^{d}$ can then be recovered by indexing into the codebook:

$$\hat{\mathcal{C}}_{i,t}^{d} = \mathcal{Z}_d[\tau_{i,t}^{d}].$$

With this encoding, the SH attributes, originally represented as $n \times f \times 48$ float parameters (where n is the number of Gaussians and f is the number of frames), are compressed into $n \times f \times 4$ integer indices, along with four shared codebooks. Furthermore, we observe that the temporally coherent SH coefficients exhibit high consistency even after conversion to indices. On average, only 1% of the SH indices for Gaussians change between adjacent frames.

To further optimize storage, instead of saving spatial-temporal SH indices for each frame, we save only the indices from the first frame and the positions where indices change in subsequent frames. Specifically, if $\tau_{i,t}^{d} \neq \tau_{i,t-1}^{d}$, we update the index to $\tau_{i,t}^{d} = k$ and record this change as an integer quadruple (t, d, i, k). During real-time decoding, these quadruples enable the reconstruction of the spatial-temporal SH indices for each frame. The order of the quadruples does not affect decoding, so we sort them in ascending order by the first two variables (t and d) and further compress them using length encoding.

VI. EXPERIMENTS

A. Compression Comparison.

We evaluate our approach against state-of-the-art (SOTA) static compression techniques, including Compact-3DGS [17] and C3DGS [16]. When adapting these methods to our human performance dataset, the resulting compression ratios differ from those reported in their original studies, which focus on static scenes. Beyond rendering RGB images, we also analyze the residual maps between the predicted results and the ground truth. As shown in Fig. 5, our compression technique demonstrates exceptional storage efficiency while maintaining rendering quality on par with other static Gaussian-based methods.

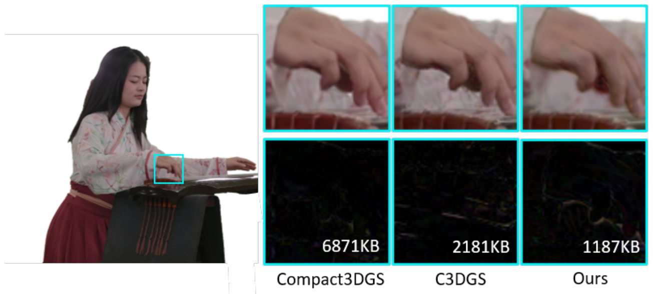

Fig. 5. Qualitative comparison of our method against Compact-3DGS [17], C3DGS [16] on our challenging dataset. we calculate the residual map between the predictions and ground truth. Our method achieves the highest compression ratio while maintaining comparable rendering quality.

A detailed quantitative comparison is presented in Tab. II. Our method capitalizes on spatial-temporal redundancy by employing residual vector quantization for encoding the motion of joint Gaussians, codec-based compression for the opacity and scaling of skin Gaussians, and a persistent codebook to encode spherical harmonics. This approach achieves a remarkable compression ratio of up to 40×, with an average storage requirement of approximately 1187KB per frame.

B. Hybrid Compression Strategy.

As illustrated in Tab. III, we analyze the performance of three distinct compression strategies tailored to various attribute types. For motion-related attributes, including Gaussian positions and rotations, residual compression emerges as the most effective method, delivering high accuracy while maintaining a manageable data size. In contrast, for Gaussian attributes such as opacity and scaling, codec-based compression strikes the best balance, achieving superior accuracy with minimal storage demands. For the SH attribute, which is notably storage-intensive, a persistent codebook is employed to ensure an optimal trade-off between precision and memory efficiency.

C. Codebook Size.

Spatial-temporal SH coefficients are encoded using fixed-size persistent codebooks. As depicted in Tab. IV, increasing the codebook size beyond 8,192 provides negligible improvements in compression performance while incurring substantial growth in storage requirements. Based on this observation, we limit the storage overhead of SH coefficients to a level comparable to that of XYZ coordinates, ensuring a balanced approach to compression and resource utilization.

TABLE III
COMPRESSION STRATEGIES ON DIFFERENT ATTRIBUTES. ERROR IS THE MEAN ABSOLUTE DIFFERENCE FROM THE UNCOMPRESSED DATA.

Method	Total size (KB)			Error		
	Motion	OP+Scale	SH	Motion	OP+Scale	SH
Raw(PLY)	5629	2801	35573	0.0	0.0	0.0
Residual	971.2	362.3	1086.6	0.00122	0.20353	0.04265
Codec	289.1	126	226	0.00982	0.04661	0.00571
Codebook	398.4	658.36	98.76	0.03817	0.05519	0.01127

TABLE IV
CODEBOOK SIZES EVALUATION. GREY ROWS INDICATE OUR CONFIGURATIONS.

Codebook Size	Metrics			
	PSNR ↑	SSIM ↑	LPIPS ↓	SIZE (KB) ↓
1024	35.447(-0.574)	0.99372(-0.000948)	0.02031(+0.00242)	54.8
2048	35.558(-0.463)	0.99386(-0.000807)	0.01999(+0.00210)	63.4
4096	35.653(-0.368)	0.99396(-0.000708)	0.01968(+0.00179)	77.3
8192	35.729(-0.292)	0.99404(-0.000628)	0.01944(+0.00155)	99.4
16384	35.771(-0.250)	0.99413(-0.000538)	0.01914(+0.00125)	152.09

VII. APPLICATION

The attribute-specific compression method we propose demonstrates significant potential in practical applications. Due to limitations in computational power and storage capacity, the playback of high-quality volumetric videos has primarily been confined to desktop systems. However, as illustrated in Fig. 6, we have successfully extended this high-quality volumetric video to mobile devices such as iPads. In processing the compressed data stream, we enhanced an open-source, Vulkan-based static Gaussian renderer [21] to support dynamic rendering, and integrated our compression algorithm for real-time decoding. Specifically, we utilize a multi-threaded approach to efficiently parallelize the processes of data loading, decoding, and rendering. After decoding each frame, the Gaussian kernels are organized into storage buffers that are compatible with compute shaders. These buffers are then rendered directly to the swapchain image using alpha blending. The application is designed to be cross-platform, supporting Windows, Linux, and Android. To further extend compatibility with iOS devices, including Vision Pro, iPhones, and iPads, we leverage the MoltenVK library to map Vulkan API calls to the Metal API. This breakthrough allows long-duration 4D sequences to be seamlessly integrated into standard computer graphics (CG) engines, enabling traditional 3D rendering pipelines to efficiently deliver immersive virtual environments.

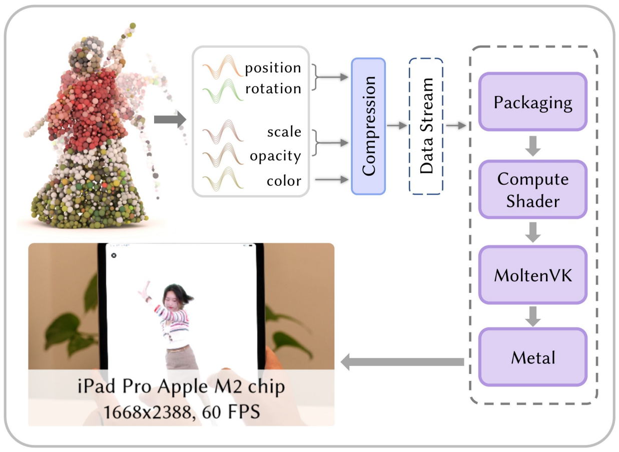

Fig. 6. The pipeline of our volumetric video player in mobile devices.

REFERENCES

[1] Yuheng Jiang, Zhehao Shen, Penghao Wang, Zhuo Su, Yu Hong, Yingliang Zhang, Jingyi Yu, and Lan Xu. Hifi4g: High-fidelity human performance rendering via compact gaussian splatting, 2023.
[2] Jarek Duda. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding, 2014.
[3] C.N. Parkinson. *Parkinson's Law, and Other Studies in Administration*. Houghton Mifflin, 1942.
[4] Uthayakumar Jayasankar, Vengattaraman Thirumal, and Dhavachelvan Ponnurangam. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *Journal of King Saud University - Computer and Information Sciences*, 33(2):119–140, 2021.
[5] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.
[6] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM siggraph 2006 papers*, pages 835–846. 2006.
[7] Yuheng Jiang, Zhehao Shen, Penghao Wang, Zhuo Su, Yu Hong, Yingliang Zhang, Jingyi Yu, and Lan Xu. Hifi4g: High-fidelity human performance rendering via compact gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19734–19745, June 2024.
[8] Jiakai Sun, Han Jiao, Guangyuan Li, Zhanjie Zhang, Lei Zhao, and Wei Xing. 3dgstream: On-the-fly training of 3d gaussians for efficient streaming of photo-realistic free-viewpoint videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20675–20685, June 2024.
[9] Penghao Wang, Zhirui Zhang, Liao Wang, Kaixin Yao, Siyuan Xie, Jingyi Yu, Minye Wu, and Lan Xu. Vˆ 3: Viewing volumetric videos on mobiles via streamable 2d dynamic gaussians. *ACM Transactions on Graphics (TOG)*, 43(6):1–13, 2024.
[10] Yuheng Jiang, Zhehao Shen, Yu Hong, Chengcheng Guo, Yize Wu, Yingliang Zhang, Jingyi Yu, and Lan Xu. Robust dual gaussian splatting for immersive human-centric volumetric videos. *ACM Transactions on Graphics (TOG)*, 43(6):1–15, 2024.
[11] Jarek Duda. Asymmetric numeral systems, 2009.
[12] Li Lian and Wei Shilei. Webp: A new image compression format based on vp8 encoding. *Microcontrollers and Embedded Systems*, 2012.
[13] Abdel-Karim Al-Tamimi. Modeling vp8/webm encoded video traces. In *2012 Sixth UKSim/AMSS European Symposium on Computer Modeling and Simulation*, pages 439–443, 2012.
[14] Wieland Morgenstern, Florian Barthel, Anna Hilsmann, and Peter Eisert. Compact 3d scene representation via self-organizing gaussian grids. In *Computer Vision – ECCV 2024*, pages 18–34, Cham, 2025. Springer Nature Switzerland.
[15] KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. Compgs: Smaller and faster gaussian splatting with vector quantization. *ECCV*, 2024.
[16] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis, 2023.
[17] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3d gaussian representation for radiance field. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21719–21728, 2024.
[18] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20654–20664, 2024.
[19] Henan Wang, Hanxin Zhu, Tianyu He, Runsen Feng, Jiajun Deng, Jiang Bian, and Zhibo Chen. End-to-end rate-distortion optimized 3d gaussian representation. In *European Conference on Computer Vision*, 2024.
[20] Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1):1–17, 2024.
[21] cedric-chedaleux shg8. 3dgs.cpp, 2024. https://github.com/shg8/3DGS.cpp.