BlenDR: Bandwidth-efficient RGB-D Representation and Delivery for Live 3D Video Streaming

Jaehong Kim
Inha University
Incheon, Republic of Korea
jaehong.kim@inha.ac.kr

Joon Ha Kim
The University of Texas at Austin
Austin, Texas, USA
jhkim38@cs.utexas.edu

Yunheon Lee
KAIST
Daejeon, Republic of Korea
yhlee37@kaist.ac.kr

Dongsu Han
KAIST
Daejeon, Republic of Korea
dhan.ee@kaist.ac.kr

Abstract

Live volumetric streaming is experiencing rapid growth due to the availability of depth sensors and 3D cameras. The RGB-D format that utilizes 2D video codecs has emerged as a promising solution for streaming. However, it falls short in delivering high-quality volumetric capture scenes at Internet-friendly bitrates due to inefficient compression, stemming from the unique challenges of adapting the live RGB-D data format to video codecs.

In response, we introduce BlenDR, a novel RGB-D representation and delivery method for live 3D streaming that redefines how RGB and depth channels are multiplexed into 2D video streams. By exploiting the inherent similarities between RGB and depth, BlenDR maximizes bandwidth efficiency for RGB-D streams and delivers superior RGB and depth quality given the same bandwidth. With the proposed new delivery scheme, BlenDR reduces bandwidth usage by up to 63% and achieves 4.6× and 4.3× higher sender and receiver FPS, respectively, than state-of-the-art systems. It also sustains near-30 FPS even for large-scale scenes with depth range up to 6 m while accurately following target bitrates, making live volumetric streaming practical over bandwidth-constrained networks.

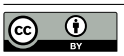
CCS Concepts

• **Information systems** → **Multimedia streaming**; • **Human-centered computing** → **Virtual reality**; **Mixed / augmented reality**; • **Computer systems organization** → **Real-time system architecture**; • **Computing methodologies** → **Volumetric models**.

Keywords

live volumetric video streaming, rgb and depth compression, rate control, multimedia systems

ACM Reference Format:
Jaehong Kim, Joon Ha Kim, Yunheon Lee, and Dongsu Han. 2026. BlenDR: Bandwidth-efficient RGB-D Representation and Delivery for Live 3D Video Streaming. In *The 24th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '26), June 21–25, 2026, Cambridge, United Kingdom.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3745756.3809192



MobiSys '26, Cambridge, United Kingdom

ACM ISBN 979-8-4007-2027-7/26/06
https://doi.org/10.1145/3745756.3809192

1 Introduction

Live volumetric video streaming, which captures and transmits real-time 3D content, is emerging as a transformative application. With the increasing availability of RGB-D cameras, iPhone Pro's LiDAR scanner, and Samsung's time-of-flight (ToF) sensors, volumetric content creation has become more accessible, extending beyond professional studios with costly setups [3, 37]. While *on-demand* volumetric streaming has been widely explored [9, 22, 58], there is growing demand for *live* volumetric streaming on consumer devices.

Earlier on-demand 3D streaming solutions [9, 17, 22, 58] primarily relied on point cloud compression and 3D codecs [6, 42]. However, these methods are ill-suited for live streaming due to inefficient compression and slow encoding/decoding processes [10, 21, 22]. To address these issues, recent research [8, 18, 21] has shifted towards using the RGB-D video format (commonly referred to as "video-plus-depth"), which leverages standard 2D video codecs like H.26x and VPx for both RGB and depth data. This approach offers better real-time performance at Internet-friendly bitrates (7.2–12 Mbps), focusing on low-resolution data (e.g., 640×480 pixels) or specific isolated elements such as faces and human bodies.

In contrast, streaming full-scene volumetric content, including background and spatial context (e.g., indoor environment) demands significantly higher bandwidth (>100 Mbps) [9, 22], which makes live transmission over the Internet particularly challenging due to limited uplink capacity and the need for strict bitrate control under varying network conditions. Existing video-plus-depth approaches fall short in addressing these challenges, due to three inherent limitations.

First, existing live 3D streaming [8, 18, 21, 44, 65] encodes RGB and depth as separate video streams. As a result, they fail to exploit the strong correlation between RGB and depth, leading to inefficient bitrate allocation under bandwidth constraints. Moreover, these methods do not account for differences in compressibility between luminance, chrominance, and depth, making it difficult to enforce strict bitrate control for adaptive streaming.

Second, video-plus-depth streaming relies on "depth packing", where 16-bit single-channel depth information is converted into a 3-channel RGB/YUV format. However, we observe that current depth packings [39, 44, 46] are not resilient to lossy video compression, resulting in significant loss of depth accuracy when transmitted over bandwidth-constrained networks.

Finally, live-captured depth often contains missing regions known as "depth holes" [13, 21, 34, 55, 60], which appear randomly and lack correlation across frames. These holes disrupt both spatial coherence within frames and temporal consistency over time, significantly degrading video codec compression efficiency. As a result, enforcing strict bitrate control for adaptive bitrate streaming becomes challenging.

In this paper, we present BlenDR, a bandwidth-efficient RGB-D representation and delivery method for live volumetric capture. First, BlenDR transforms how RGB and depth channels are multiplexed into video streams with a new *Y-Depth+UV multiplexing* method. By exploiting the correlation between RGB and depth, we combine luminance (Y) and depth into a single video stream while encoding chrominance (UV) at a lower resolution. This approach allows for more efficient bitrate allocation, prioritizing the quality-sensitive Y-Depth video to achieve better bandwidth efficiency.

Second, BlenDR incorporates a *robust depth packing* technique designed to withstand lossy video compression. It transforms depth data into a compression-friendly format, mitigating codec-induced distortions through an artifact concealment mechanism that preserves accuracy. The method is lightweight, supports real-time processing, and adapts to both 2- and 3-channel packing.

Finally, BlenDR employs *sender-side depth completion*. Unlike conventional methods [13, 32, 55, 60] that handle depth completion as an offline post-compression task, BlenDR proactively manages depth holes in real time, using RGB as a guide before the depth data reaches the video codec. This not only improves the compression efficiency of the depth stream but also aligns the depth data more closely with the RGB data, enabling more effective Y-Depth compression.

Our target scenario focuses on indoor environments with fixed multi-camera rigs, covering room-scale spaces (e.g., 6.6m × 8.5m × 3m), such as meeting rooms or studio setups. We implement BlenDR as a multi-view RGB-D live 3D streaming system for this scenario, where a sender at the capture-side transmits multi-view RGB-D streams to a receiver-side edge server for full-scene reconstruction.

Using a real-world room-scale dataset [15], our results demonstrate that BlenDR outperforms video-plus-depth approaches [21, 39, 44] in reconstruction and visual quality under the same network bandwidth budget and video codec settings. Compared to the state-of-the-art live 3D streaming system, DeltaStream [20], BlenDR reduces bandwidth usage by up to 63%, achieves up to 4.6× higher sender FPS and 4.3× higher receiver FPS, while sustaining near-30 FPS even when streaming full-scene depth ranges (up to 6 m). Under constrained networks, BlenDR maintains real-time performance down to 16 Mbps, accurately tracking target bitrates without overshoot. In contrast, prior systems [8, 17, 20] exhibit content-dependent bitrate spikes, TCP retransmission overhead, and significant latency drift. Sample videos are available at https://ina.kaist.ac.kr/projects/blendr.

In summary, we make the following contributions:

- **Practical streamability over the Internet.** BlenDR supports real-time encoding/decoding with accurate bitrate control for adaptive streaming and is compatible with widely-used video codecs (H26x, VPx).
- **Enhanced efficiency in 3D live video streaming.** BlenDR enables bandwidth-efficient compression and delivery of live volumetric capture.
- **New 3D video representation.** BlenDR introduces a new video representation for RGB-D streaming, incorporating robust depth packing, RGB-D multiplexing, and early depth completion.

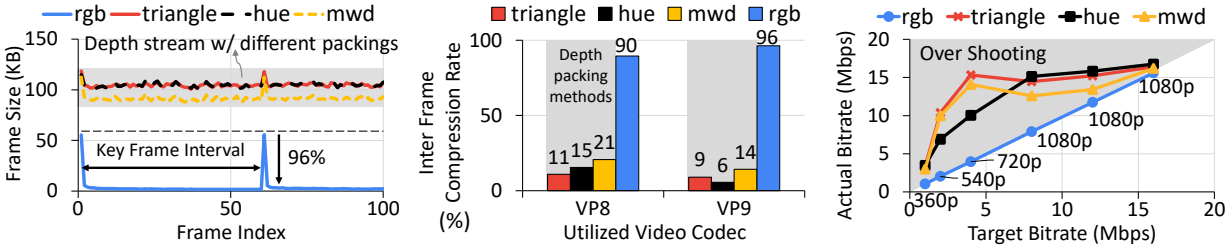
2 Background

RGB-D [13, 32, 55] imagery combines traditional color information (RGB) with depth data (D). This format is utilized by dedicated RGB-D cameras (e.g., Microsoft Azure Kinect, Intel RealSense) and modern smartphones (e.g., iPhone 12+, Samsung Galaxy S series) equipped with LiDAR or Time-of-Flight (ToF) sensors. Using calibration matrices, RGB-D data can be converted into a 3D point cloud—a set of 3D points, each defined by its (x, y, z) coordinates and associated RGB color attributes.

Depth packing encodes depth efficiently by converting a 16-bit depth map into an 8-bit, 3-channel RGB or YUV image. These depth-packed images are then encoded using standard video codecs, producing a depth video stream. In "video-plus-depth" streaming [8, 18, 21, 39, 44], the depth stream is transmitted alongside the RGB video. To recover the original depth after compression, the depth-packed data undergoes "unpacking", an inverse process that reconstructs the 16-bit depth map from the compressed image. For effective streaming, packing and unpacking must be resilient to lossy compression to minimize reconstruction errors.

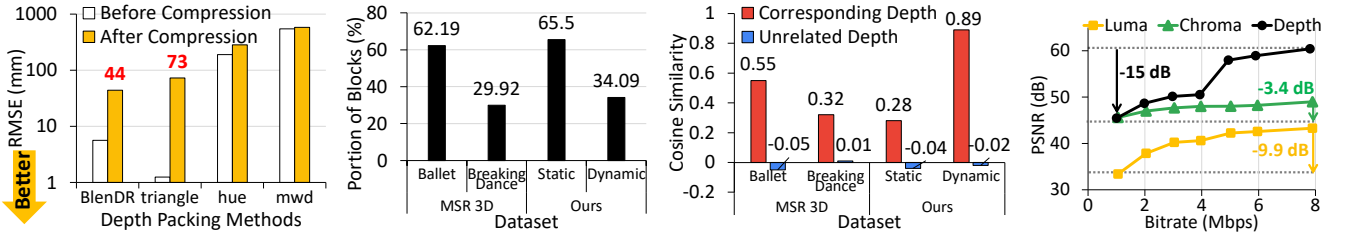
Depth holes. Modern depth sensors typically produce depth data as a single-channel, 16-bit image, where each pixel represents the distance from the sensor in millimeters. However, the data often contains missing pixels, referred to as "depth holes" [13, 21, 34, 54, 55, 60]. Depth holes arise from various factors such as environmental interference, calibration issues, poor lighting, or low-reflective surfaces. They are frequently observed along object edges but can occur unpredictably across the image, posing a challenge to compression efficiency. To the best of our knowledge, this issue has not been explicitly addressed in the context of live video streaming systems.

Color space. Human vision is more sensitive to luminance (brightness) than to chrominance (color) detail. The YUV color space leverages this by separating an image into three 8-bit channels: luminance (Y) and two chrominance channels (U and V). This separation aligns with human visual perception and optimizes encoding by allowing the chrominance data to be reduced without noticeably affecting image quality. YUV encoding formats, such as YUV420p and YUV422p, use subsampling to reduce chrominance data, thereby reducing overall file size while maintaining visual fidelity. YUV444p, on the other hand, preserves all chrominance information without any reduction. This paper exploits flexibility in channel allocation of the YUV format, along with the perceptual importance of luminance over chrominance, to develop a new and efficient encoding framework for depth images.



(a) Frame size after video compression. **(b)** Inter-frame compression rate of (a). **(c)** Unpredictable rate control.

Figure 1: Depth holes negatively affect video compression.



(a) Existing depth packing is not robust. **(b)** Fraction of macroblocks with the same motion vectors. **(c)** Cosine similarity of motion vectors. **(d)** Rate-distortion curve of Luma, Chroma, and Depth.

Figure 2: Various limitations of existing RGB-D dual stream (video-plus-depth) approach.

3 Motivation

This section exposes key limitations of existing approaches to RGB-D live streaming, using raw 1080p RGB-D frames captured from Azure Kinect.

Following prior works [8, 18, 21], we compress the captured RGB and depth streams using VP8 and VP9 (via libvpx) by employing three well-known depth packing methods: multi-wavelength depth (MWD) [44], Hue color encoding [46], and Triangle wave encoding [39]. The same encoding parameters are used for both RGB and depth.

L1. Depth holes disrupt compression, making RGB-D streaming incompatible with adaptive bitrate ladders. Depth sensors, such as LiDAR and Time-of-Flight systems, rely on reflected light to measure distance [21, 61]. As a result, these technologies often struggle to generate dense and clean depth maps, particularly in environments with poor lighting, low-reflective surfaces, or near object edges and discontinuities [55, 60]. Unlike RGB data, depth measurements are not captured for every pixel, resulting in random "depth holes".

Depth holes pose a major challenge in compression. Unlike RGB frames, which contain significant spatial and temporal redundancy that codecs can exploit, depth frames suffer from high-frequency noise due to missing or unreliable depth values. This noise disrupts compression efficiency, leading to significantly larger depth streams compared to their RGB counterparts. For example, as shown in Figure 1(a), the average residual size of the RGB stream is just 1.51 KB, whereas depth streams range from 88 to 101 KB—making them 58× ∼ 68× larger. This results in significantly larger inter-frame sizes for depth, as illustrated in Figure 1(b).

The inefficiency makes it extremely difficult to control the bandwidth of the depth stream for adaptive streaming. To demonstrate this, we compressed both the RGB and depth streams while varying the target bitrate for VP9. As shown in Figure 1(c), while the RGB stream closely follows the target bitrate, the depth stream consistently exceeds it across all depth packing methods. This highlights the difficulty in effectively rate-controlling depth data, further complicating its use in streaming applications.

L2. Existing depth packing methods are not robust to lossy compression. Even after addressing depth holes with our depth completion technique, current packing methods introduce significant errors when restoring depth values after lossy video compression. Figure 2(a) shows the errors of three depth packing methods before and after compression (1080p depth frames compressed at 4 Mbps using VP9). Both the hue and MWD methods exhibit substantial inaccuracies, with deviations of tens of centimeters even before compression. The triangle method, the most sophisticated of the three, employs fast-changing piecewise linear functions to map depth to YUV values, initially achieving higher accuracy due to its complexity. However, its performance deteriorates significantly post-compression. This decline occurs because its triangular wave pattern introduces high-frequency components, which preserve precision before compression but suffer significant quantization loss. These findings highlight the need for more robust depth packing that can withstand lossy encoding, ensuring accurate depth transmission under bandwidth constraints.

L3. Existing dual-stream methods encode RGB and depth independently, despite their strong correlation. This design has two key limitations. First, it results in suboptimal compression by failing to exploit the strong correlation between RGB and depth

frames. To quantify this correlation, we extracted motion vectors from macroblocks in both RGB and depth videos using two samples from our captured data and the MSR3D [64] dataset. Each dataset includes two video samples representing different movement levels: static and dynamic. To ensure accurate assessment of this correlation, we removed noise artifacts from the depth measurements.

Figure 2(b) shows the percentage of blocks sharing the same motion vector in both RGB and depth streams. This fraction remains consistently high across all videos, with static scenes exhibiting an exceptionally high percentage. For blocks with differing motion vectors, we measure their cosine similarity, as shown in Figure 2(c). The similarity is notably high, reaching up to 0.89 in dynamic videos. Even in relatively static videos, the ~35% of blocks with differing vectors still achieve similarity scores between 0.28 and 0.55. In contrast, unrelated RGB and depth videos exhibit cosine similarity ranging from -0.02 to -0.05. These results confirm a strong correlation between motion vectors, revealing significant redundancy across the two streams.

Second, encoding RGB and depth independently limits compression optimization by ignoring differences in compressibility between luminance, chrominance, and depth. Figure 2(d) presents the rate-distortion curves for these components, showing that chrominance can be compressed more efficiently than luminance and depth with minimal quality loss. This suggests that allocating less bandwidth to chrominance while prioritizing depth and luminance would improve bandwidth efficiency. However, in existing video-plus-depth streaming, luminance (Y) and chrominance (UV) are jointly encoded as a single stream—mirroring 2D video encoding—which restricts flexible bandwidth allocation.[1]

4 System Design

BlenDR introduces a design that addresses the limitations of existing approaches, with the following objectives:

- Ensure depth compression and overall RGB-D streaming efficiency to strictly conform to the bandwidth constraints of adaptive streaming bit ladders.
- Design a codec-agnostic approach that can be applied to existing standard 2D video codecs (i.e., H26x, VPx) and support a wide range of applications, from high-quality RGB-D streaming to depth-only streaming [45, 49, 62, 63].
- Enable real-time encoding suitable for client-side embedded devices.

4.1 Y-Depth+UV Multiplexing

BlenDR introduces a novel method for multiplexing luminance (Y), depth (D), and chrominance (UV) into video streams. Building on the insights from §3, we encode luminance and depth in the same video stream while transmitting chrominance as a separate lower-resolution stream.

This format offers significant flexibility for various use cases. First, it prioritizes resources for the quality-critical Y-Depth (Luminance-Depth) stream, improving compression efficiency over a conventional RGB-D split. For example, even with limited uplink bandwidth, our method ensures precise depth video alongside sharp

[1]While YUV subsampling provides some control over bit allocation between luminance and chrominance, it remains a coarse-grained solution.

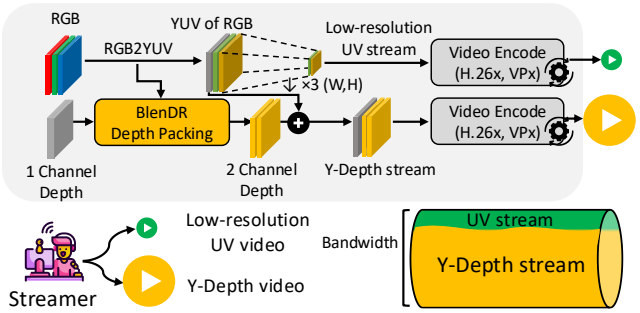


Figure 3: BlenDR Y-Depth+UV Multiplexing Design.

RGB visuals with minimal and unnoticeable color variation. Second, when color fidelity is less critical, all resources can be allocated to Y-Depth, with color reconstructed using deep learning-based video colorization methods [51, 59] and a reference image. Next, we detail how BlenDR multiplexes luminance, chrominance, and depth into video streams.

Multiplexing. To establish a codec-agnostic approach, we pack depth data into the original chroma (UV) space, creating a Y-Depth stream. This method aligns well with the inner workings of video encoding [7, 26, 40, 47, 53], where RGB is converted to YUV, and the motion vectors calculated for the luminance (Y) channel are reused for the chrominance (UV) channels. By embedding depth into the chroma channels, the codec can seamlessly encode both luminance and depth, taking advantage of their similarities. This approach does not require any modifications to the codec, making it compatible with standard codecs.

For the chroma stream, we create a separate low-resolution video. However, since the original luminance channel is no longer necessary for this video, we shift the U channel data to the Y channel, repurposing U as Y. This allows the codec to calculate motion vectors based on the U data for the chroma stream. In the now-empty U channel, we set a median value of 128 for all pixels, which prevents the codec from allocating any bits. We denote this approach as Y-Depth+UV, where '+' indicates an additional video stream and Y-Depth signifies that luminance and depth are packed in a single video stream.

Resolution and bitrate allocation. The Y-Depth stream is encoded in its original high-resolution by allocating most of the available bitrate. Instead, chroma is aggressively encoded using the remaining bitrate at a lower resolution. Empirically, we find dedicating 90% of the bitrate to the high-resolution Y-Depth stream and only 10% to the downscaled (by a factor of 3) chroma stream consistently yield superior performance.

Figure 3 illustrates the design of BlenDR. Our multiplexing technique is particularly advantageous for live streaming in bandwidth- or resource-constrained environments. It requires less bandwidth and computing power than the standard dual-stream (YUV+Depth) approach because 1) chrominance is further downscaled; and 2) the depth is packed to what used to be the UV channel, which has limited space—only a quarter of the Y's in the YUV420p format. As a result, BlenDR is well-suited for Internet-based streaming.

Supporting diverse use cases. While the primary goal of BlenDR is to enable RGB-D streaming with Y-Depth+UV multiplexing, it is designed to accommodate a variety of application needs. Y-Depth+UV streaming is used for applications requiring both high-quality depth and color information. For use cases where depth quality is prioritized over RGB [19, 24, 52], particularly in bandwidth-constrained environments, BlenDR can operate in a mode that streams only "Y-Depth" video while periodically transmitting a reference color image. In this configuration, the system recovers the color information using the luminance data from the Y-channel and the reference image, following a similar principle to NeuriCam [51]. Unlike NeuriCam, which employs a low-resolution grayscale video with a high-resolution color reference, BlenDR maintains a high-resolution luminance video, complemented by an infrequent low-resolution color image. This ensures that the Y-channel remains at high resolution, preserving image quality and structural similarity (SSIM). We explore this design as a case study in Appendix §D. Finally, some applications only require transferring depth data [45, 49, 62, 63], in which case depth utilizes all three channels in the codec.

Realizing this approach requires overcoming two key challenges. First, supporting diverse use cases necessitates both 2- and 3-channel depth packing. However, existing methods are not robust to lossy compression and are constrained to 3-channel configurations, with 2-channel adaptations resulting in significant quality loss. Second, raw depth images with noisy depth holes hinder efficient compression. In the following, we present our design to address these challenges.

4.2 Robust Depth Packing

BlenDR introduces a new design for depth packing. The primary objectives are to devise a packing (encoding) and unpacking (decoding) scheme that is compression-friendly, i.e., robust to lossy compression, and compatible with both 2- and 3-channel encodings.

Key idea. The design is centered around two main principles. First, we craft our packing to produce a depth image with smooth color transitions in the UV space (YUV space for 3-channel packing) that correlate with gradual depth changes in the physical world. This makes depth images more compressible because video codecs use intra-prediction that leverages neighboring blocks to predict and encode each block [1]. Second, we ensure that error due to lossy compression is locally confined. Instead of utilizing the entire space, we carefully limit the encoded UV (or YUV) space such that even after the distortion from video encoding, accurate recovery of the original depth is likely. We start with the 2-channel method, followed by its expansion to 3-channel.

Packing. We first normalize the depth values based on the effective range of our Azure Kinect sensors, which is typically bounded at 5.46 meters in indoor capture scenarios [30]. The normalized depth (D) is then mapped to a point on one of four distinct linear subspaces, (R_1, R_2, R_3, R_4). These subspaces are strategically chosen as they correspond to the edges of the 2D (uv) space, as depicted in Figure 4. Note that the subspaces (R_1, R_2, R_3, R_4) are respectively characterized as $v=0, u=255, v=255, u=0$ in the uv space. To map D on R_i, we create $|R_i|$ (four) segments in the scalar range [0,1] by dividing it using the segmentation thresholds, r_1, r_2, r_3, as shown in the figure. We assign each subspace R_i to the i-th segment in the scalar range [0,1] and linearly map points in R_i onto the segment. For example, the R_1 linearly maps to $[0, r_1)$, R_2 maps to $[r_1, r_2)$ and so on.

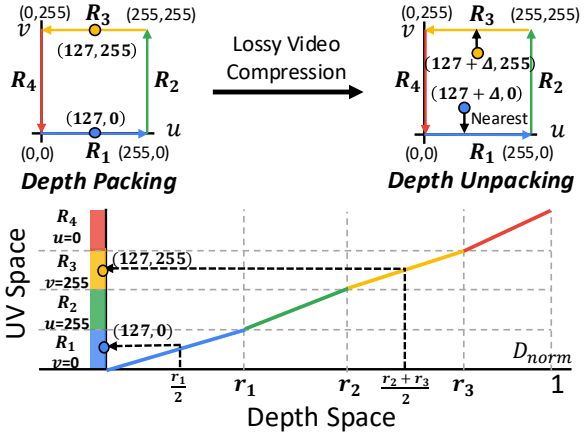


Figure 4: Illustration of BlenDR's Depth packing.

Algorithm 1 Depth to UV Conversion

- D: Array of depth values
- max_depth: Maximum cutoff depth value
- r: Segmentation thresholds (r_1, r_2, r_3)

function DEPTH2UV(D, max_depth, r)
  $(uv_{min}, uv_{max}) \leftarrow (0, 255)$ ▷ for 8-bit channels
  $uv_{range} \leftarrow |uv_{max} - uv_{min}|$
  $depth \leftarrow \text{CLIP}(D, 0, max\_depth)$
  $D \leftarrow D/max\_depth$ ▷ Normalize & convert to float32
  $(r_1, r_2, r_3) \leftarrow r$
  Initialize u and v arrays to zeros with the same size as D
  for i = 1 to $|D|$ **do**

$$(u_i, v_i) \leftarrow \begin{cases} (|\frac{D_i}{r_1}|, 0) & \in R_1 \textbf{ if } 0 \le D_i < r_1 \\ (1, |\frac{D_i - r_1}{r_2 - r_1}|) & \in R_2 \textbf{ if } r_1 \le D_i < r_2 \\ (1 - |\frac{D_i - r_2}{r_3 - r_2}|, 1) & \in R_3 \textbf{ if } r_2 \le D_i < r_3 \\ (0, 1 - |\frac{D_i - r_3}{1 - r_3}|) & \in R_4 \textbf{ if } r_3 \le D_i < 1 \end{cases}$$

  $(u, v) \leftarrow (u, v) \cdot uv_{range} + uv_{min}$
  return u, v

D is then assigned to a point in the linear mapped segment, R_i, such that $D \in [r_{(i-1)}, r_i)$, with its relative position in R_i determined as $\frac{D - r_{(i-1)}}{r_i - r_{(i-1)}}$. Then the scalar value D is transformed to the (u, v) space, preserving the relative position within R_i as shown in Figure 4. For example, if D is a mid-point in R_1, $v=0$ (R_3, $v=255$), its uv-coordinate becomes 127,0 (127, 255). Algorithm 1 describes this in mathematical formulations. The careful progression along the UV periphery ensures a seamless gradation in color that mirrors the depth changes.

Unpacking. After retrieving the decoded depth image from the encoded depth video, depth unpacking is carried out. It restores the original depth value from the decoded pixel, (u', v'), by locating the *nearest* point of the edges in the UV space. As the encoding space is limited to the edge of the UV space, its nearest point on the edge is highly likely to be close to the original location (u, v).

Even if (u', v') was located in a different region from which it was originally encoded, the nearest edge is adjacent to its original region. This minimizes the error, as the adjacent regions in the UV space correspond to neighboring depth values.

In sum, our strategy achieves both efficiency and robustness. First, the depth packing produces a smoothly changing colored image. As the depth progresses, the change only occurs in one of the two dimensions in UV, making compression more efficient. Second, the encoding is confined to the edge of the UV space, which limits the possible outcomes and allows it to mask distortions in the unpacking phase. Setting the max depth to a large enough value (e.g. 6 meters) leaves a buffer at the corners, making it even more robust to errors.

Segmentation thresholds. The segmentation threshold (r_1, r_2, r_3) determines the precision for each segmented space. The precision becomes $\frac{|r_n - r_{n-1}|}{255}$ for each region R_n. In our empirical assessments, equally dividing the space, $(r_1, r_2, r_3) = (0.25, 0.5, 0.75)$, yields the best results in RMSE metric. These thresholds can be fine-tuned according to the depth distribution and the significance of specific depth ranges. For instance, to capture finer details in nearer depth ranges, the interval between the relevant thresholds can be reduced to improve precision in those areas. For a metric that considers relative error against the depth's absolute value, such as δ_i [25], a configuration more oriented towards closer ranges (e.g., $(0.2, 0.3, 0.4)$) works better. The effects of varying these thresholds are further explored in the Appendix §C.

Design comparison. Our method stands out for being more reliable, compression-friendly, flexible in supporting both 2- and 3-channel configurations, and lightweight compared to existing approaches. Existing designs, including various color filters (e.g., Hue) [8, 11, 46] and the triangle wave method [39], face challenges in accurately restoring depth because they lack sufficient mechanisms to mask distortions caused by lossy compression [27, 50]. The triangle wave method achieves high precision by employing fast-changing, piece-wise linear functions [39] to depth to color. However, this produces images containing a wide range of high-frequency information, making it less ideal for video compression and highly sensitive to quantization. Consequently, it requires significantly more bandwidth than our method, and its performance rapidly deteriorates in bandwidth-constrained environments.

Moreover, existing approaches rely on all three channels to operate effectively, meaning the absence of even one channel leads to a substantial loss of depth information. In contrast, our approach is lightweight, using pixel-wise operations that are independent of one another, relying on simple logic and arithmetic. This design allows for high parallelization on both CPUs and GPUs. Compared to the triangle wave method, BlenDR 's 2-channel approach is 2.6× faster for packing and 2.1× faster for unpacking, making it a more efficient solution overall.

3-channel packing. Our methodology easily extends to 3-channel packing by including the Y channel to utilize the 6 edges of the YUV three-tuple space. Specifically, each encoding space corresponds to one of the six edges on the YUV cube, and the normalized depth value is segmented into one of the six linear sub-spaces. For unpacking, the process involves locating the nearest edge within the YUV space. Adding the Y channel augments our depth packing method, providing additional space for improved depth representation and enabling more precise quantization. Finally, BlenDR's depth packing is compatible with YUV subsampling. For example, in YUV420p format, the UV space is a quarter the size of the Y channel. Therefore, after depth packing, we downsample the UV components of the depth image using bilinear interpolation by a factor of 2.

4.3 Sender- and Receiver-side Processing

Sender-side depth completion. As highlighted in §3, depth holes degrade video compression efficiency, making it essential to address them before depth packing and encoding. We choose to employ RGB data as a reference for filling in these holes, based on the following considerations. First, RGB data is rich in texture and edge details, which depth data typically lacks, making it an excellent source for inferring missing depth information. Second, RGB data exhibits strong spatial and temporal coherence, and using it as a guide improves the compression efficiency of depth video. Finally, by aligning depth data more closely with RGB, we increase the similarity between the two, amplifying the effectiveness of our Y-Depth+UV multiplexing approach.

While generating clean and dense depth maps using RGB guidance is a well-established technique [13, 32, 55, 60], our approach specifically targets the overlooked issue of noisy depth holes in real-time video compression. Unlike conventional methods that address these artifacts post-compression at the receiver side [44], our strategy *proactively* resolves noisy holes before encoding at the *sender side* in *real time*. Our objectives are distinct: prior work [13, 32, 55, 60] focuses primarily on enhancing depth map visual quality, while we aim to 1) improve depth compression efficiency, 2) align depth with RGB for better Y-Depth compression, and 3) ensure real-time processing.

Selective filtering. To address depth holes, we employ a selective filter based on the Joint Bilateral Filter (JBF) [35], a widely used image-guided noise-removal technique. While the standard JBF can improve compression efficiency when applied before depth packing, it has limitations. It applies uniform filtering across all pixels, inadvertently modifying valid pixels, and incorporates neighboring points, including invalid pixels (depth holes), which degrades both accuracy and compression efficiency. To mitigate these issues, we implement JBF to selectively update only *invalid* depth pixels using *only* valid pixels within the filter kernel. This preserves the accuracy of captured valid pixels while ensuring a smooth depth map.

Enabling real-time processing. For real-time depth completion, we use a combination of JBF's nearest-neighbor filling and multi-scale image processing [5]. We apply the JBF in a coarse-to-fine manner, starting from downscaled images and progressively refining depth as resolution increases. Unlike the vanilla JBF, which requires large filters, this strategy enables us to use smaller filters while maintaining the completeness of the final depth image.

Specifically, for 1080p RGB-D inputs, we use three scales with spatial Gaussian parameters $\sigma_s \in \{12, 6, 2\}$ and range (RGB intensity) parameters $\sigma_r \in \{0.2, 0.04, 0.02\}$, using a small kernel size $S = 2$. Here, σ_s controls the spatial extent of smoothing (i.e., how

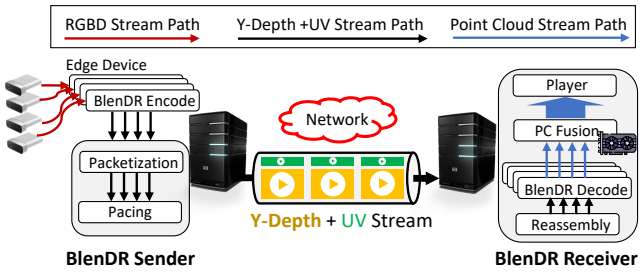


Figure 5: BlenDR end-to-end system overview. Each camera view is processed in parallel through dedicated encoding and decoding modules. BlenDR encode module consists of depth completion, robust depth packing, Y-Depth+UV multiplexing, and video encoding.

far neighboring pixels contribute), while σ_r controls edge preservation by weighting differences in RGB intensity, allowing depth discontinuities to be maintained.

Receiver-side processing. The receiver processes RGB-D data and reconstructs a 3D point cloud from the two video streams. In the YUV+Depth format, this involves straightforward video decoding and depth unpacking. For Y-Depth+UV streams, the receiver first decodes the Y-Depth stream, where the first channel contains luminance (Y). Depth is then recovered by unpacking the data from the two repurposed UV channels. Next, the receiver decodes the second video stream containing chrominance (UV) and merges it with luminance from the first stream to reconstruct the color information. Since the chroma channels are downscaled by a factor of three at the sender side, bilinear upscaling is applied for basic quality restoration, while super-resolution techniques [16, 56] can be used for higher-quality results.

Point cloud construction. The receiver constructs a point cloud using the recovered RGB-D data. We follow the standard method for constructing point cloud from 2D depth map [8, 29, 46, 64], which involves applying a post-processing filter to remove flying pixels [46, 64] and transformation of RGB-D to 3D space [29]. BlenDR utilizes the RGB's intrinsic and extrinsic matrices for the transformation, as depth is aligned to RGB space at the sender-side for depth completion. More sophisticated 3D volumetric fusion [34], such as Truncated Signed Distance Function (TSDF), can be applied if BlenDR's receiver is a cloud server equipped with GPU resources.

Multi-view Fusion. BlenDR supports multi-view fusion, merging streams from multiple RGB-D cameras on the receiver side to enable six degrees of freedom (6DoF), as illustrated in Figure 5. During the offline phase, BlenDR calibrates multiple cameras on the sender side, aligning them within a single coordinate system using per-camera intrinsic calibration matrices and a calibration board [17, 36]. This process generates an extrinsic matrix that transforms point clouds from secondary views into the target view. Sender-side calibration occurs only once before streaming begins, which is sufficient for fixed camera setups. The sender precomputes and shares all calibration matrices (intrinsic, extrinsic, ICP transforms) before streaming, and the receiver uses them to fuse Y-Depth+UV frames in real time.

5 Implementation

BlenDR sender and receiver are implemented in C++ (≈5 KLOC) using OpenCV for image processing and Open3D (v0.19.0) for GPU-accelerated point-cloud fusion and rendering. Both RGB and depth streams are encoded and decoded via FFmpeg's libavcodec (v58.76.100) using the VP9 (libvpx) codec. The encoding parameters included a Group of Picture (GOP) duration of two seconds and a frame rate of 30 fps. Comprehensive codec parameters can be found in Appendix §B.

We use UDP for low-latency transmission and implement a custom application-layer protocol for synchronized multi-view RGB-D delivery. The protocol encapsulates encoded RGB and depth with per-view metadata to support joint transport while remaining compatible with standard video codecs. To ensure stable delivery over UDP, we further incorporate a packetization and pacing mechanism that segments frames into MTU-sized chunks and schedules transmissions smoothly, avoiding IP fragmentation and reducing burstiness. On the receiver, packets are reassembled, decoded, and fused in real time for rendering. Figure 5 illustrates the end-to-end system of BlenDR.

6 Evaluation

We evaluate BlenDR by answering the following questions:

- How does BlenDR perform compared to existing live volumetric video streaming systems? (§ 6.2)
- Does BlenDR effectively improve the bandwidth efficiency of RGB-D streams? (§ 6.3)
- How does each design component of BlenDR contribute to the overall performance? (§ 6.4)

6.1 Experimental Setup

Dataset. Our target scenario requires high-resolution, multi-view RGB-D video capturing full-scene context in indoor environments. To this end, we use the publicly available SceneHub4D dataset [15]. SceneHub4D provides full depth-range RGB-D captures of realistic, room-scale indoor environments, including background geometry and object interactions under natural activities (e.g., group conversations). Captured using modern Azure Kinect sensors, the dataset provides high-resolution RGB and depth (1920×1080), enabling the reconstruction of dense point clouds. For our evaluation, we select four representative scenes (Lab, Couch, Kitchen, Hallway) and use the first sequence (id=1) from each scene.

Baselines. To evaluate BlenDR, we compare it against representative live volumetric video streaming systems, as summarized in Table 1. Our primary baseline is *DeltaStream* [20], a state-of-the-art live 3D streaming system that reduces point cloud bandwidth by reusing the motion vectors of RGB video to guide the encoding of 3D geometry. By performing motion estimation on the RGB stream, DeltaStream achieves lower bitrate and faster encoding/decoding performance of 3D codec, Draco [6]. We use the authors' open-source implementation and adapt it to Azure Kinect inputs, as the original system assumes Intel RealSense cameras operating at 640×480 resolution.

System	Data Type	Codec	Target Rate Control	Supported Resolution	Capture Scope	Dataset
LiveScan3D [17]	Point Cloud	✗	✗	–	Full-scene	Custom
MetaStream [8]†	Point Cloud	Draco	✗	1024×768	Full-body	Custom
DeltaStream [20]	Point Cloud	Draco	✗	640×480	Full-body	Custom
FarFetchFusion [21]	RGBD	H.264	–	640×480	Facial-region	Custom
Starline [18]	RGBD	H.265	–	1280×1024	Upper-body	Custom
BlenDR (Ours)	RGBD	H.265/VP9	✓	1920×1080	Full-scene	SceneHub4D [15]

Table 1: Comparison of recent live 3D video streaming systems. BlenDR supports online target bitrate control, full-scene capture, and high-resolution processing with modern RGB-D sensors. †Metastream uses point cloud in the server-to-client transmission stage. '–' indicates that the feature is not discussed in the paper.

For additional baselines, we use the same implementations provided in DeltaStream's evaluation suite, which include widely referenced systems such as LiveScan3D [17] and MetaStream [8]. We follow the assumptions and configurations used in DeltaStream's evaluation, and in the case of MetaStream, we likewise focus our evaluation on the server-to-client transmission stage, where reconstructed point cloud is transmitted to the receiver.

Finally, we include FarFetchFusion [21] as a lightweight Video-plus-Depth baseline. FarFetchFusion uses a video codec (H.264) to transmit RGB-D streams; however, because the full system is not open-source, we implement only the depth-packing component described in their work (Y–channel packing). Note that all these baseline systems support only a limited capture scope and low-resolution RGB-D inputs, as summarized in Table 1.

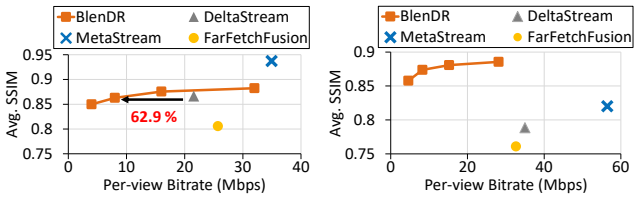
Metrics. We evaluate BlenDR using standard metrics for real-time volumetric video streaming: (1) frames per second (FPS) for throughput, (2) end-to-end latency, (3) bandwidth usage, and (4) visual quality, measured using SSIM.

Methodology & setup. We follow the DeltaStream evaluation methodology to ensure a fair comparison and run all systems on a consistent hardware setup. In our deployment model, the RGB-D camera is attached to an edge device (Jetson Xavier), which is responsible for sender-side processing and forwarding encoded frames to an edge server acting as the sender. The sender performs packetization and transmission, while the receiver, equipped with an RTX 5090 GPU and an Intel Ultra 9 285K processor, decodes and renders the reconstructed 3D content, as illustrated in Figure 5.

Emulated network environment. To emulate constrained network conditions, we use the Mahimahi network emulator [33]. Network traces are generated using a scripted bandwidth generator based on packet–time models derived from Mahimahi's public interface, which produces synthetic traces at target bandwidth levels (e.g., 10–100 Mbps). These traces are replayed during experiments to model realistic wireless conditions. We configure the emulator with a one-way latency of 40 ms and vary the available link bandwidth between 16 and 128 Mbps.

6.2 System Performance

Bandwidth usage and visual quality. Figure 6 compares visual quality versus per-view bitrate under two depth-range settings: (a) a near-range configuration (<3 m), following DeltaStream's default setup, and (b) a full-scene configuration that preserves the entire room depth (up to 6 m). DeltaStream and MetaStream operate



(a) Depth range (<3m) **(b)** Depth range (6m)

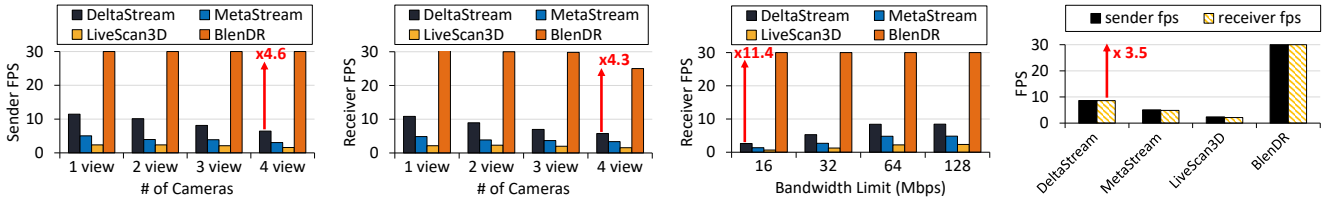
Figure 6: Bandwidth usage and visual quality.

primarily in the near range because they segment the foreground human subject to reduce point cloud density; as a result, they cannot represent background interactions (e.g., moving furniture or interacting with objects farther in the room).

Under the <3 m setting (Fig. 6(a)), BlenDR provides multiple operating points along the bitrate ladder, whereas other systems operate at a single point. Because BlenDR uses a modern 2D video codec, it benefits from highly efficient spatial and temporal compression, unlike DeltaStream, which still relies on Draco-based 3D compression with frequent keyframes (interval of 5). By contrast, BlenDR uses a longer keyframe interval (60), leading to a 62.9% bitrate reduction at similar SSIM quality. FarFetchFusion also uses a 2D codec but relies on simple 8-bit Y–channel depth packing, which severely quantizes depth at distances beyond short-range capture, resulting in lower quality even at higher bitrates.

The benefit of BlenDR becomes more pronounced when evaluating the full-scene depth range (Fig. 6(b)). As the depth limit increases, point counts grow substantially for point-cloud–based systems, causing their bitrate to rise sharply and reducing their compression efficiency. In contrast, BlenDR treats depth maps as 2D images, allowing the video codec to perform spatial prediction and block-based compression across the full scene without requiring background segmentation. As a result, BlenDR consistently achieves a better rate–distortion curve than all baselines when encoding full-scene RGB-D data.

Frames per second (FPS). Figure 7 evaluates how well each system scales with the number of camera views. To isolate computation-only bottlenecks, we do not apply network constraints in this experiment. Because BlenDR relies on a highly optimized 2D video codec, it maintains stable throughput close to 30 fps across 1–4 views (both sender Fig. 7(a) and receiver Fig. 7(b)), whereas point-cloud–based baselines fall significantly below this range due to



(a) Sender FPS vs. # of cameras. **(b)** Receiver FPS vs. # of cameras. **(c)** FPS under bandwidth limit. **(d)** Increased depth range (6m).

Figure 7: End-to-end FPS.

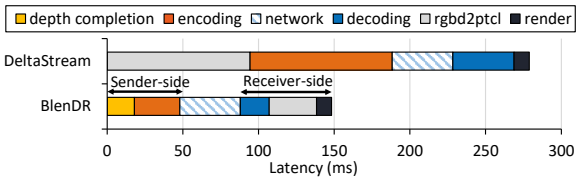


Figure 8: End-to-end Latency. BlenDR transmits compression-efficient RGB-D streams over the network and performs reconstruction at the receiver-side media server. No bandwidth limit is applied in this measurement.

Sender-side Process	GPU (ms)	CPU (ms)
Depth Completion	21.0 ± 3.5	28.2 ± 4.7
Robust Depth Packing	11.8 ± 1.6	9.8 ± 1.4
Video Encoding	4.7 ± 1.4	4.6 ± 1.5
Throughput (fps)	43.9 ± 1.9	33.0 ± 1.5

Table 2: Latency breakdown on the camera-side edge device (Jetson Xavier).

their dependence on slower 3D compression pipelines. We verified that DeltaStream [20] can reach 30 fps at both sender and receiver when operating at its native 640×480 resolution, but its performance drops sharply at 1080p and does not scale beyond two views. As a result, BlenDR achieves up to 4.6× higher sender FPS and 4.3× higher receiver FPS compared to DeltaStream under the four-view configuration, as shown in Figure 7.

FPS under bandwidth constraint. Figure 7(c) evaluates how well the system performs under varying network conditions. When bandwidth limits are applied, the bottleneck shifts from computation to the network. All point-cloud–based systems experience a drop in receiver FPS compared to the no-limit case because they operate at a single high-bitrate point (typically 50–100 Mbps). When the available bandwidth falls below this operating point, these systems cannot maintain their target frame rate, causing FPS to degrade sharply.

In contrast, BlenDR supports multiple operating points through online target–bitrate control, enabling it to adapt to the available bandwidth. As a result, BlenDR sustains near-30 fps performance down to 16 Mbps (allocating 4 Mbps per view with four cameras), whereas all point-cloud baselines collapse in this range.

Figure 7(d) further evaluates performance under full-scene depth range (6 m). Increasing the depth range increases the number of points for point-cloud baselines, reducing their throughput even under unconstrained bandwidth. In contrast, BlenDR maintains near real-time performance at 1080p because its compression efficiency is largely insensitive to point density: depth is encoded as 2D images by a video codec, rather than being expanded into increasingly large point sets.

End-to-end latency. We compare the end-to-end latency of BlenDR against DeltaStream. Figure 8 presents a breakdown of the latency across all processing stages, including sender-side encoding, network transmission delay, receiver-side decoding, RGB-D–to–point-cloud conversion, and rendering (capture-to-render pipeline). Network delay varies depending on the emulated one-way latency and is shown as a patterned region in the figure. No bandwidth limits are applied in this experiment.

BlenDR achieves significantly lower end-to-end latency than DeltaStream due to the much faster encoding and decoding speed of the 2D video codec, compared with the Draco-based 3D point cloud compression by DeltaStream. A key difference is that BlenDR transmits an RGB-D representation and performs RGB-D–to–point-cloud reconstruction on the receiver side using GPU acceleration, enabling efficient delivery of a 2.5D representation over the network that is highly compressible with a 2D codec. In contrast, DeltaStream converts RGB-D frames into point clouds on the sender side and then compresses and streams the point cloud itself using Draco [6], incurring significantly higher processing and transmission latency. Finally, the reported latency (total of 148.2 ms) corresponds to the first-frame latency. After the initial frame, this cost is amortized by pipeline parallelism, allowing BlenDR to sustain real-time throughput at 30 fps.

Since BlenDR uses one-way UDP streaming, network latency mainly affects startup delay (time to first frame), while steady-state performance remains stable once playback begins. We further validate this over a wide-area Internet path (≈90 ms one-way latency), where BlenDR maintains a stable playback frame rate throughout streaming, with only a slight increase in time to first frame.

Latency breakdown on edge devices. Table 2 summarizes the sender-side processing latency on the camera-side edge device, where BlenDR performs depth completion, packing, and video encoding before transmitting the stream to the central server. The measurements include both GPU and CPU execution on the Jetson Xavier. The pipeline is fully parallelized, enabling the system to

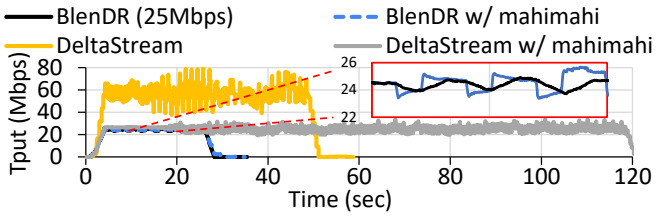


Figure 9: Rate control under a bandwidth-limited environment (25 Mbps). BlenDR maintains throughput near the target bitrate and achieves real-time playback.

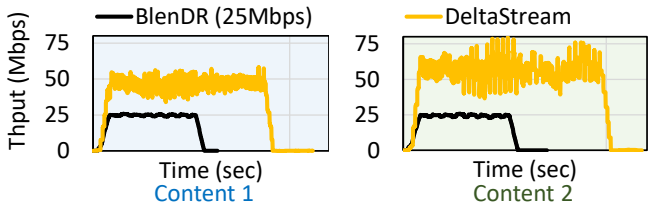
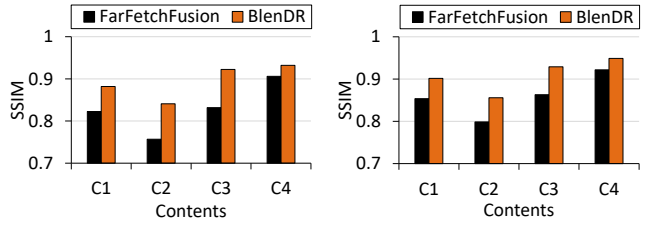
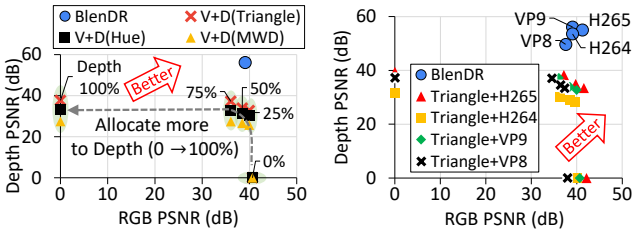


Figure 10: Rate control across different contents. Target bitrate is set to 25 Mbps. BlenDR achieves consistent and predictable bitrate control across diverse contents.



(a) Results with VP9 **(b)** Results with H.265

Figure 11: Comparison of RGB-D streaming methods under consistent codec settings across different content.



(a) BlenDR outperforms V+D. **(b)** BlenDR is codec-agnostic.

Figure 12: BlenDR vs. V(v%)+D(100-v%) with 8Mbps.

sustain real-time throughput (> 30 fps) even on a resource-limited edge platform.

Practical streamability via rate control. We demonstrate BlenDR's ability to accurately regulate bitrate to a target value of 25 Mbps over a 25-second video sequence. We first evaluate the system without any bandwidth constraint. As shown in Figure 9, BlenDR maintains throughput tightly around the target bitrate and completes playback in real time. When a bandwidth limit is applied using our Mahimahi-based emulation, the completion time remains identical, indicating that BlenDR does not overshoot the available bandwidth and therefore avoids buffering stalls.

The periodic spikes in the blue curve correspond to keyframes. Although we rely on UDP, we implement a packetization and pacing layer that divides frames into 1500-byte chunks and schedules them smoothly. This prevents IP fragmentation, mitigates bursty transmissions, and enables large keyframes to be delivered gracefully.

DeltaStream, in contrast, exhibits highly irregular throughput with large spikes caused by sending point-cloud keyframes every five seconds (yellow curve in Fig. 9). Even without bandwidth limits, the sender- and receiver-side processing pipelines cannot sustain the 30 fps, leading to a drift in completion time relative to wall-clock. Under a 25 Mbps limit, the mismatch becomes more severe: DeltaStream frequently overshoots the available bandwidth, which, combined with its TCP-based transport, results in retransmissions and timeouts that further extend end-to-end latency (gray curve). We provide sample video comparing BlenDR with DeltaStream at https://ina.kaist.ac.kr/projects/blendr.

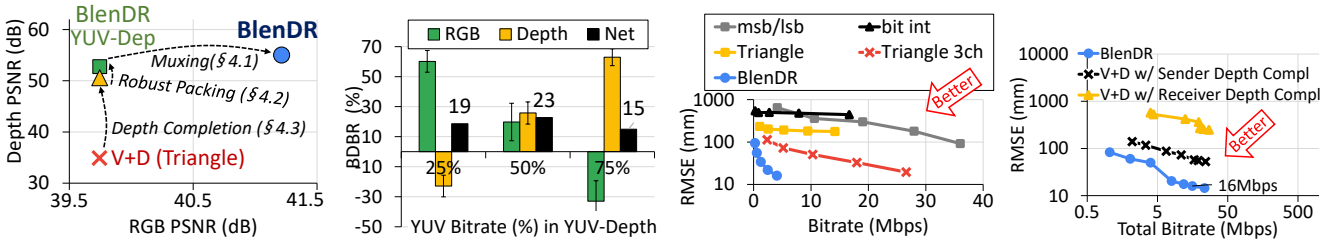
Predictable rate control across contents. BlenDR achieves consistent and predictable bitrate control across diverse contents, closely tracking the 25 Mbps target in Figure 10. This is enabled by the stable rate-control mechanisms of the underlying video codec. In contrast, DeltaStream relies on a fixed compression configuration, making its bitrate highly content-dependent and leading to drastically different peak rates across content types, which complicates operation under heterogeneous network conditions.

6.3 BlenDR vs. RGB-D Approaches

In this section, we compare BlenDR against RGB-D streaming methods (i.e., Video-plus-Depth (V+D) approaches) in terms of compression efficiency.

Evaluation with consistent codec. To examine performance under consistent codec settings, we evaluate BlenDR and FarFetchFusion under identical codec configurations and bandwidth budgets. We adapt FarFetchFusion to operate with modern codecs (H.265 and VP9). We use YUV 4:4:4 10-bit pixel format for H.265, similar to Starline [18], and YUV 4:4:4 8-bit for VP9. At the same bandwidth usage (32 Mbps), the results are as shown in Figure 11. BlenDR achieves higher visual quality across both codecs due to its advanced depth packing, whereas FarFetchFusion relies on simple Y-channel packing, leading to depth quantization beyond short-range capture.

Comparison with more V+D. We evaluate BlenDR against Video-plus-Depth (V+D) approaches that employ more advanced depth packing methods, including triangle, hue, and MWD introduced in § 3. Specifically, we compare BlenDR and V+D across a spectrum of depth bandwidth allocations (0 to 100%) while fixing the total bandwidth. Figure 12(a) displays the quality of RGB and depth in PSNR, comparing BlenDR and V+D at each point of allocation. Even when V+D allocates a greater bitrate to depth, it cannot reach the depth quality of BlenDR due to our efficient depth packing.



(a) Ablation study. **(b)** YD+UV vs. YUV+D **(c)** 2 Channel packings. **(d)** Depth completion.

Figure 13: Component-wise Analysis: Ablation study and effect of each design component.

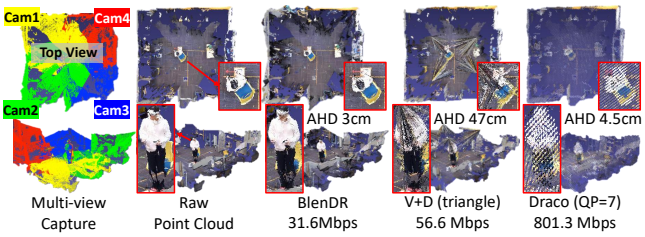


Figure 14: Qualitative result: BlenDR vs. Baselines

Figure 12(b) compares the performance of BlenDR against V+D with the triangle method for various mainstream video codecs—H265, H264, VP9, and VP8—across the full RGB to depth bitrate allocation range, highlighting the compatibility of BlenDR.

Qualitative result. Figure 14 presents the end-to-end reconstruction after multi-view fusion of RGB-D videos from four cameras at the receiver side. We also include Draco (v1.5.7) [6] as a representative point-cloud codec for comparison. Each RGB-D stream is encoded at 8 Mbps, totaling 32 Mbps. We use the average Hausdorff distance (AHD) [23, 41, 43, 48] to evaluate point cloud quality. BlenDR effectively controls the bitrate, achieving 31.6 Mbps when targeting 32 Mbps, while V+D struggles, reaching 56.6 Mbps. Draco lacks bitrate control, requiring QP=7 to match BlenDR 's AHD quality, leading to 801.3 Mbps (35.4 Mbps without color). In terms of point cloud quality, BlenDR achieves an AHD of 3 cm, outperforming V+D (47 cm) and Draco (4.5 cm). Moreover, BlenDR requires 25× less bandwidth than Draco for similar AHD quality while reconstructing a 108× denser (i.e., high-resolution) point cloud.

6.4 Component-wise In-depth Analysis

In this section, we analyze each design component in detail.

Ablation study. Figure 13(a) offers an ablation study, detailing the progressive impact of BlenDR's design components on performance enhancement. The figure shows the case for 8 Mbps, but we observe a similar trend at other bitrates. The sender-side depth completion (§4.3), indicated by the yellow triangle, raises Depth PSNR by 15.63 dB, surpassing the V+D (triangle) baseline. Moving forward, the triangle method is replaced with our robust depth packing (§4.2), shown as the green square. This switch yields an additional increase of 2.18 dB in Depth PSNR. The Y-Depth+UV multiplexing (§4.1), illustrated by the blue circle, delivers a 2.25 dB and 1.46 dB improvement in depth and RGB PSNR, respectively. Each design element is essential, contributing to the heightened compression efficiency of BlenDR.

Benefit of Y-Depth+UV streaming. Figure 13(b) presents the BDBR performance (i.e., bitrate savings) of our Y-Depth+UV multiplexing compared to a YUV+Depth variant of BlenDR. In this context, YUV+Depth refers to a version of BlenDR that does not use Y-Depth+UV multiplexing but instead relies on sender-side depth completion and 3-channel depth packing, encoding RGB and depth as separate streams. For this comparison, we vary the RGB (YUV) bandwidth allocation in the YUV+Depth scheme from 25% to 75%. The green bars and yellow bars in Figure 13(b) represent the BDBR for the RGB stream and depth stream, respectively, while the black bars indicate the combined average net BDBR. Compared to YUV+Depth at 50% RGB bitrate allocation, BlenDR achieves a net bitrate savings of 23%, calculated as the average of 20% savings in RGB BDBR and 26% savings in depth BDBR.

As the bitrate for RGB increases within the YUV+Depth scheme, the quality of the RGB stream improves. However, this allocation strategy negatively impacts the depth stream quality, reflected by the rising BDBR for the depth component. Despite this trade-off, BlenDR achieves net gains in compression efficiency, resulting in an overall more efficient BDBR than standard separate encoding, with 15-23% net BDBR savings (i.e., bitrate savings).

Accuracy of BlenDR's 2-channel packing. Figure 13(c) compares the RMSE performance of various 2-channel packing methods across different bitrates. Conventional depth packing methods struggle with 2-channel configurations due to significant depth information loss when even a single channel is absent. For example, MSB/LSB [39], which splits 16-bit depth data into two 8-bit halves, and bit interleaving [39], which separates odd and even bits, result in large errors. Additionally, we adapted the triangle method for two-channel packing by using its Y and U packing and unpacking functions and omitting V. The absence of a channel led to significant depth detail loss.

In contrast, BlenDR maintains depth integrity and outperforms the 3-channel packing method (triangle 3ch), demonstrating superior compression efficiency.

Benefit of sender-side depth completion. To isolate the benefit of sender-side depth completion, we compare BlenDR with advanced V+D versions which incorporate depth completion on either the sender- and receiver-side. Figure 13(d) shows the compression efficiency results. Both V+D and BlenDR benefit from performing depth completion at the sender-side, highlighting the

importance of addressing depth holes at the source for improved video compression and more accurate bitrate control. Additionally, BlenDR outperforms V+D in both depth RMSE and RGB PSNR, demonstrating the enhancement provided by robust depth packing and the Y-Depth+UV method, which leverages the correlation between depth and RGB.

7 Related Work & Discussions

Existing live 3D streaming systems. The introduction of depth cameras has spurred the development of live 3D streaming systems [8, 10, 12, 17, 18, 20, 21, 37]. Holoportation [37] was one of the first to offer immersive telepresence through real-time 3D reconstructions, though it required substantial bandwidth (over 1 Gbps). LiveScan3D [17] utilized multiple Kinect v2 sensors for live 3D streaming. However, it faces bandwidth constraints due to its reliance on point clouds for data transmission. MeshReduce [12] proposes a distributed mesh reconstruction system for live 3D scene capture, utilizing textured mesh representation and Draco [6], which relies on frame-by-frame compression. The system demonstrates a proof-of-concept rate control mechanism based on offline profiling per-scene. In contrast, BlenDR leverages well-established 2D video codecs that achieve predictable and adaptive rate control at runtime. MeshReduce's approach can also be combined with BlenDR, streaming mesh for static background while BlenDR handles dynamic objects.

Multi-view fusion systems. Starline [18], MetaStream [8], and FarFetchFusion [21] focus on volumetric fusion from multiple cameras, using the video-plus-depth method for 3D data transmission. However, these systems are specialized for low-resolution facial captures and do not support full-scene fidelity. To leverage video codecs, FarFetchFusion employs a simplistic depth packing method that only utilizes the Y channel. The approach is sufficient for their intended use cases, which focus on segmented objects like facial features rather than full-scene reconstruction. In contrast, BlenDR supports high-resolution, full-scene live RGB-D capture while maintaining compatibility with existing 3D telepresence and multi-camera fusion systems. Its multiplexing and depth packing methods are orthogonal to these approaches, allowing broader applicability across different volumetric streaming applications.

Deployment scope and limitations. BlenDR is designed for static, indoor multi-camera capture setups, such as studio or meeting room environments similar to prior live 3D streaming systems [12, 17, 18, 21, 37]. This design choice reflects practical constraints of the hardware, specifically, the use of Azure Kinect sensors, which offer reliable depth data within a limited range (0.5–6 m) [28] and under controlled lighting. These factors naturally restrict the system to indoor deployments.

Alternative depth-free approaches, such as photogrammetry, Gaussian Splatting [14] or Neural Radiance Fields (NeRF) [31], can capture 3D scenes without dedicated depth sensors, and are being explored for real-time streaming applications. However, they require more complex preprocessing pipelines and remain challenging to deploy in real-time scenarios. In contrast, BlenDR is designed for practical, low-latency deployment using depth sensors and 2D codecs.

Generalization. BlenDR operates on RGB-D image inputs. We believe any RGB frame and corresponding depth map (e.g., LiDAR or ToF-based depth outputs) can serve as input to our pipeline, provided that camera in/extrinsics are available. This includes mobile devices that provide 16-bit depth maps as 2D images (e.g., iPhone 13 or higher).

Extending to mobile and VR devices. BlenDR uploads the full 3D scene to the edge media server located at the receiver-side, where reconstruction is performed. This architecture naturally supports scaling to mobile or VR devices by streaming synthesized views or viewports on demand, similar to VR streaming systems. Our current work focuses on real-time full-scene delivery from capture to the receiver. Extending BlenDR toward mobile/VR deployment is left for future work.

Making depth completion more reliable. BlenDR applies conservative RGB-guided depth completion only for small, locally inferable holes and avoids hallucinating large missing regions. Still, this may introduce inaccuracies compared to actual measurements of these missing values. The accuracy of depth completion could be improved by using deep learning inference methods [2, 38]; however, this paper focuses on building a real-time system that runs on edge computing devices.

Neural-enhanced streaming is a promising technique for improving the streaming video quality. It has been applied across multiple domains: NAS [56] for on-demand 2D video, PARSEC [4] for 360°video streaming, LiveNAS [16] and NeuroScaler [57] for live streaming, NeuriCam [51] for IoT Cameras, and YuZu [58] for on-demand volumetric streaming. Extending the benefit of neural enhancement to live 3D streaming presents a promising path for future research.

8 Conclusion

Existing live 3D streaming systems fail to maintain stable bitrate constraints, making them unsuitable for adaptive streaming. We introduce BlenDR, a new bandwidth-efficient RGB-D representation that enables practical streamability. BlenDR achieves this through Y-Depth+UV multiplexing for efficient bitrate allocation, robust depth packing for resilience to lossy compression, and sender-side depth completion to improve encoding efficiency.

Our evaluation shows that BlenDR strictly adheres to adaptive streaming bit ladders, cuts bandwidth usage by up to 63%, and delivers higher sender/receiver FPS than existing systems. It operates at near-30 FPS under constrained bandwidth and full-scene depth, where point-cloud pipelines degrade sharply. Overall, BlenDR makes live volumetric streaming practical over bandwidth-constrained networks.

Acknowledgments

We thank anonymous reviewers for providing helpful feedback and suggestions to improve our work. This research was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) of the Korea government (MSIT) (No.RS-2024-00398157), and by INHA UNIVERSITY Research Grant. We also thank WiseLab at Carnegie Mellon University for their assistance with the SceneHub4D dataset.

References

[1] Jim Bankoski, Paul Wilkins, and Yaowu Xu. 2011. Technical overview of VP8, an open source video codec for the web. In *2011 IEEE International Conference on Multimedia and Expo*. IEEE, 1–6.

[2] Zhao Chen, Vijay Badrinarayanan, Gilad Drozdov, and Andrew Rabinovich. 2018. Estimating Depth from RGB and Sparse Sensing. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[3] Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. 2015. High-quality streamable free-viewpoint video. *ACM Transactions on Graphics (ToG)* 34, 4 (2015), 1–13.

[4] Mallesham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjal Sahu, Aruna Balasubramanian, and Samir R Das. 2020. Streaming 360-degree videos using super-resolution. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1977–1986.

[5] Raanan Fattal, Maneesh Agrawala, and Szymon Rusinkiewicz. 2007. Multiscale shape and detail enhancement from multi-light image collections. *ACM Trans. Graph.* 26, 3 (2007), 51.

[6] Google. 2024. Draco 3D data compression library. https://google.github.io/draco/.

[7] Adrian Grange, Peter De Rivaz, and Jonathan Hunt. 2016. VP9 bitstream & decoding process specification. *WebM Project* (2016).

[8] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. 2023. MetaStream: Live Volumetric Content Capture, Creation, Delivery, and Rendering in Real Time. In *Proceedings of the 29th annual international conference on mobile computing and networking*. https://doi.org/10.1145/3570361.3592530

[9] Bo Han, Yu Liu, and Feng Qian. 2020. ViVo: Visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th annual international conference on mobile computing and networking*. 1–13.

[10] Jinhan Hu, Aashiq Shaikh, Alireza Bahremand, and Robert LiKamWa. 2021. Characterizing real-time dense point cloud capture and streaming on mobile devices. In *Proceedings of the 3rd ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges*. 1–6.

[11] IntelRealSense. 2022. Intel RealSense SDK 2.0 for Intel RealSense depth cameras. https://github.com/IntelRealSense/librealsense/blob/master/src/proc/colorizer.cpp.

[12] Tao Jin, Mallesham Dasa, Connor Smith, Kittipat Apicharttrisorn, Srinivasan Seshan, and Anthony Rowe. 2024. MeshReduce: Scalable and Bandwidth Efficient 3D Scene Capture. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*. 20–30. doi:10.1109/VR58804.2024.00026

[13] Adrien Kaiser, Jose Alonso Ybanez Zepeda, and Tamy Boubekeur. 2018. Proxy clouds for live RGB-D stream processing and consolidation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 252–268.

[14] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[15] Jaehong Kim, Tao Jin, Mallesham Dasari, Srinivasan Seshan, and Anthony Rowe. 2026. SceneHub4D: A Dataset and Evaluation Framework for 6-DoF 4D VR Scenes . *IEEE Transactions on Visualization & Computer Graphics* (March 2026), 1–11. doi:10.1109/TVCG.2026.3679140

[16] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. 2020. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 107–125.

[17] Marek Kowalski, Jacek Naruniec, and Michal Daniluk. 2015. Livescan3D: A Fast and Inexpensive 3D Data Acquisition System for Multiple Kinect v2 Sensors. In *2015 International Conference on 3D Vision*. 318–325. doi:10.1109/3DV.2015.43

[18] Jason Lawrence, Danb Goldman, Supreeth Achar, Gregory Major Blascovich, Joseph G. Desloge, Tommy Fortes, Eric M. Gomez, Sascha Häberling, Hugues Hoppe, Andy Huibers, Claude Knaus, Brian Kuschak, Ricardo Martin-Brualla, Harris Nover, Andrew Ian Russell, Steven M. Seitz, and Kevin Tong. 2021. Project starline: a high-fidelity telepresence system. *ACM Trans. Graph.* 40, 6, Article 242 (Dec. 2021), 16 pages. doi:10.1145/3478513.3480490

[19] Thi-Lan Le, Minh-Quoc Nguyen, and Thi-Thanh-Mai Nguyen. 2013. Human posture recognition using human skeleton provided by Kinect. In *2013 International Conference on Computing, Management and Telecommunications (ComManTel)*. 340–345. doi:10.1109/ComManTel.2013.6482417

[20] Hojeong Lee, Yu Hong Kim, Sangwoo Ryu, James Won-Ki Hong, Sangtae Ha, and Seyeon Kim. 2025. *DeltaStream: 2D-Inferred Delta Encoding for Live Volumetric Video Streaming*. Association for Computing Machinery, New York, NY, USA, 361–373. https://doi.org/10.1145/3711875.3729131

[21] Kyungjin Lee, Juheon Yi, and Youngki Lee. 2023. FarfetchFusion: Towards Fully Mobile Live 3D Telepresence Platform. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 1–15.

[22] Kyungjin Lee, Juheon Yi, Youngki Lee, Sunghyun Choi, and Young Min Kim. 2020. GROOT: a real-time streaming system of high-fidelity volumetric videos. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.

[23] Yonghyeon Lee, Seungyeon Kim, Jinwon Choi, and Frank Park. 2022. A statistical manifold framework for point cloud data. In *International Conference on Machine Learning*. PMLR, 12378–12402.

[24] Billy Y.L. Li, Ajmal S. Mian, Wanquan Liu, and Aneesh Krishna. 2013. Using Kinect for face recognition under varying poses, expressions, illumination and disguise. In *2013 IEEE Workshop on Applications of Computer Vision (WACV)*. 186–192. doi:10.1109/WACV.2013.6475017

[25] Fangchang Ma and Sertac Karaman. 2018. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 4796–4803.

[26] Detlev Marpe, Thomas Wiegand, and Gary J Sullivan. 2006. The H. 264/MPEG4 advanced video coding standard and its applications. *IEEE communications magazine* 44, 8 (2006), 134–143.

[27] MartyG. 2022. Depth Image Colorization Formula. https://support.intelrealsense.com/hc/en-us/community/posts/4419463222931-Depth-Image-Colorization-Formula.

[28] Microsoft. 2021. Azure Kinect DK hardware specifications. https://learn.microsoft.com/en-us/previous-versions/azure/kinect-dk/hardware-specification.

[29] Microsoft. 2022. Use Azure Kinect Sensor SDK image transformations. https://learn.microsoft.com/en-us/azure/kinect-dk/use-image-transformation.

[30] Microsoft. 2024. Azure Kinect DK. https://azure.microsoft.com/ko-kr/products/kinect-dk.

[31] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.

[32] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. 2012. Indoor Segmentation and Support Inference from RGBD Images. In *ECCV*.

[33] Ravi Netravali, Anirudh Sivaraman, Keith Winstein, Somak Das, Ameesh Goyal, and Hari Balakrishnan. 2014. Mahimahi: a lightweight toolkit for reproducible web measurement. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 129–130. doi:10.1145/2740070.2631455

[34] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. 2011. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE international symposium on mixed and augmented reality*. Ieee, 127–136.

[35] OpenCV. 2024. Filters Extended Image Processing. https://docs.opencv.org/4.x/da/d17/group__ximgproc__filters.html.

[36] OpenCV. 2024. OpenCV Camera Calibration. https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html.

[37] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L Davidson, Sameh Khamis, Mingsong Dou, et al. 2016. Holoportation: Virtual 3d teleportation in real-time. In *Proceedings of the 29th annual symposium on user interface software and technology*. 741–754.

[38] Jinsun Park, Kyungdon Joo, Zhe Hu, Chi-Kuei Liu, and In So Kweon. 2020. Non-Local Spatial Propagation Network for Depth Completion. In *Proc. of European Conference on Computer Vision (ECCV)*.

[39] Fabrizio Pece, Jan Kautz, and Tim Weyrich. 2011. Adapting standard video codecs for depth streaming.. In *EGVE/EuroVR*. ., 59–66.

[40] WebM Project. 2024. libvpx: VP8/VP9 Codec SDK. https://github.com/webmproject/libvpx/.

[41] Javier Ribera, David Guera, Yuhao Chen, and Edward J Delp. 2019. Locating objects without bounding boxes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6479–6489.

[42] Radu Bogdan Rusu and Steve Cousins. 2011. 3D is here: Point Cloud Library (PCL). In *2011 IEEE International Conference on Robotics and Automation*. 1–4. doi:10.1109/ICRA.2011.5980567

[43] Oliver Schutze, Xavier Esquivel, Adriana Lara, and Carlos A Coello Coello. 2012. Using the averaged Hausdorff distance as a performance measure in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation* 16, 4 (2012), 504–522.

[44] Stephen Siemonsma and Tyler Bell. 2022. HoloKinect: Holographic 3D Video Conferencing. *Sensors* 22, 21 (2022). doi:10.3390/s22218118

[45] Shuran Song, Fisher Yu, Andy Zeng, Angel X. Chang, Manolis Savva, and Thomas Funkhouser. 2017. Semantic Scene Completion From a Single Depth Image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[46] Tetsuri Sonoda and Anders Grunnet-Jepsen. 2021. Depth image compression by colorization for Intel RealSense depth cameras. *Intel, Rev 1.0* (2021).

[47] 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.

[48] Danhang Tang, Saurabh Singh, Philip A Chou, Christian Hane, Mingsong Dou, Sean Fanello, Jonathan Taylor, Philip Davidson, Onur G Guleryuz, Yinda Zhang, et al. 2020. Deep implicit volume compression. In *Proceedings of the IEEE/CVF*

conference on computer vision and pattern recognition. 1293–1303.
[49] Terabee. 2023. Depth sensors: Precision and personal privacy. https://www.terabee.com/depth-sensors-precision-personal-privacy/.
[50] TetsuriSonoda. 2022. Github repository of Hue colorization. https://github.com/TetsuriSonoda/rs-colorize/issues/1.
[51] Bandhav Veluri, Ali Saffari, Collin Pernu, Joshua Smith, Michael Taylor, and Shyamnath Gollakota. 2022. NeuriCam: Video Super-Resolution and Colorization Using Key Frames. *arXiv preprint arXiv:2207.12496* (2022).
[52] Jingpeng Wang, Kechen Song, Defu Zhang, Menghui Niu, and Yunhui Yan. 2022. Collaborative Learning Attention Network Based on RGB Image and Depth Image for Surface Defect Inspection of No-Service Rail. *IEEE/ASME Transactions on Mechatronics* 27, 6 (2022), 4874–4884. doi:10.1109/TMECH.2022.3167412
[53] Paul Wilkins, Yaowu Xu, Lou Quillio, James Bankoski, Janne Salonen, and John Koleszar. 2011. VP8 Data Format and Decoding Guide. RFC 6386. doi:10.17487/RFC6386
[54] Andrew D Wilson. 2017. Fast lossless depth image compression. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces*. 100–105.
[55] Jianxiong Xiao, Andrew Owens, and Antonio Torralba. 2013. Sun3d: A database of big spaces reconstructed using sfm and object labels. In *Proceedings of the IEEE international conference on computer vision*. 1625–1632.
[56] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. 2018. Neural adaptive content-aware internet video delivery. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 645–661.
[57] Hyunho Yeo, Hwijoon Lim, Jaehong Kim, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2022. NeuroScaler: neural video enhancement at scale. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 795–811.
[58] 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 (NSDI 22)*. 137–154.
[59] Bo Zhang, Mingming He, Jing Liao, Pedro V Sander, Lu Yuan, Amine Bermak, and Dong Chen. 2019. Deep exemplar-based video colorization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 8052–8061.
[60] Yinda Zhang and Thomas Funkhouser. 2018. Deep depth completion of a single rgb-d image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 175–185.
[61] Yunfan Zhang, Tim Scargill, Ashutosh Vaishnav, Gopika Premsankar, Mario Di Francesco, and Maria Gorlatova. 2022. InDepth: Real-time depth inpainting for mobile augmented reality. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 1 (2022), 1–25.
[62] Shibo Zhao and Zheng Fang. 2018. Direct Depth SLAM: Sparse Geometric Feature Enhanced Direct Depth SLAM System for Low-Texture Environments. *Sensors* 18, 10 (2018). doi:10.3390/s18103339
[63] Zerong Zheng, Tao Yu, Hao Li, Kaiwen Guo, Qionghai Dai, Lu Fang, and Yebin Liu. 2018. HybridFusion: Real-Time Performance Capture Using a Single Depth Sensor and Sparse IMUs. In *Proceedings of the European Conference on Computer Vision (ECCV)*.
[64] C Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. 2004. High-quality video view interpolation using a layered representation. *ACM transactions on graphics (TOG)* 23, 3 (2004), 600–608.
[65] Marek Šimoník. 2022. Record3D. https://github.com/marek-simonik/record3d.

Appendix A Detailed Algorithms

Algorithm 2 UV to Depth Conversion

- u,v : Array of u and values (uint8)
- max_depth: Maximum depth value
- r: Quartile mapping values (r_1, r_2, r_3)
- Returns array of decoded depth (uint16)

function UV2Depth(u, v, max_depth, r)
  $(uv_{min}, uv_{max}) \leftarrow (0, 255)$ ▷ for 8-bit channels
  $uv_{range} \leftarrow |uv_{max} - uv_{min}|$
  $(u, v) \leftarrow (u, v)/uv_{range}$ ▷ Normalize & Convert to float32
  $(r_1, r_2, r_3) \leftarrow r$
  Initialize depth array D with the same size as u and v
  for $i = 1$ to $|u|$ **do**
    Calculate distances and find the minimum:
    $dist_1 \leftarrow |v_i|$
    $dist_2 \leftarrow |1 - u_i|$
    $dist_3 \leftarrow |1 - v_i|$
    $dist_4 \leftarrow |u_i|$
    min_idx $\leftarrow \text{argmin}\{dist_1, dist_2, dist_3, dist_4\}$
    $$D_i \leftarrow \begin{cases} r_1 \cdot u_i, & \text{if min\_idx} = 0 \\ r_1 + |r_2 - r_1| \cdot v_i, & \text{if min\_idx} = 1 \\ r_2 + |r_3 - r_2| \cdot |1 - u_i|, & \text{if min\_idx} = 2 \\ r_3 + |1 - r_3| \cdot |1 - v_i|, & \text{if min\_idx} = 3 \end{cases}$$
  $D \leftarrow \lfloor D \cdot max\_depth \rfloor$ ▷ Scale & convert to uint16
  return D

Appendix B Experimental Settings

Encoding parameters. We implement video encoding using the `libavcodec` library in a C++ application. The following configurations are applied to `libvpx`, `libx264`, and `libx265`:

- **libvpx (VP9):**
```
-pix_fmt {pixel_format} -s {width}x{height} -framerate {fps}
-keyint_min 60 -g 60 -quality realtime -speed 8 -threads 16
-row-mt 1 -tile-columns 4 -frame-parallel 1 -static-thresh 0
-max-intra-rate 300 -lag-in-frames 0 -qmin 4 -qmax 48
-minrate {bitrate}k -maxrate {bitrate}k -b:v {bitrate}k
-error-resilient 1
```
- **libx264 (H.264):**
```
-pix_fmt {pixel_format} -s {width}x{height} -framerate {fps}
-keyint_min 60 -g 60 -preset fast
-b:v {bitrate}k
```
- **libx265 (H.265):**
```
-pix_fmt {pixel_format} -s {width}x{height} -framerate {fps}
-keyint_min 60 -g 60 -preset veryfast
-b:v {bitrate}k -maxrate {bitrate}k -bufsize {bitrate*2}k
```

Appendix C Additional Results

Effect of segmentation thresholds on depth packing. Figure 15 compares the performance of two segmentation threshold schemes, $(0.2, 0.3, 0.4)$ and $(0.25, 0.5, 0.75)$, denoted as "Close"(i.e., the configuration oriented towards closer depth ranges) and "Uniform", respectively. The "Close" is optimal for the $\delta_{1.01}$ [25]—where it measures the percentage of pixels with less than 1% error. This method excels at capturing finer details without exceeding a minimal error threshold, hence showing a higher percentage in the $\delta_{1.01}$ metric.

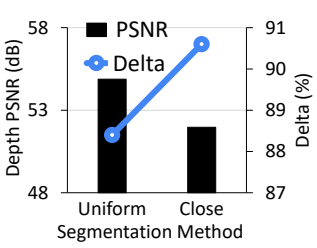


Figure 15: Effect of segmentation threshold.

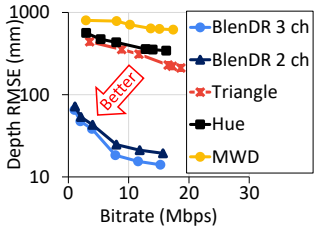


Figure 16: BlenDR for depth-only streaming with 3 channel depth packing.

Data Type (# of stream)	Depth Packing	Model	Bitrate (Mbps)	Depth RMSE	RGB PSNR	RGB SSIM
Depth (1 stream)	Triangle	-	14	252mm	-	-
Y-Depth (1 stream)	Ours	NeuriCam	8	23 mm	35.8 (dB)	0.96

Table 3: BlenDR with video colorization models.

On the other hand, the "Uniform", while not as effective for $\delta_{1.01}$, shows better performance in RMSE, which translates to higher PSNR, indicating its suitability for general depth accuracy over a broader range of depth values. These findings illustrate how different threshold configurations can be tailored to optimize for specific depth quality metrics.

Appendix D BlenDR with Diverse Use cases

We demonstrate BlenDR's support for diverse use cases.

BlenDR for depth-only streaming. BlenDR supports depth-focused applications with its 3-channel packing, enhancing depth accuracy by utilizing all available channels. Figure 16 demonstrates this advantage. BlenDR's accuracy in RMSE is superior at tested bitrates between 1-16 Mbps compared to other depth packing methods. Compared to the triangle packing method, it achieves 8.9 times lower average RMSE, and offers a 1.2 times reduction in average RMSE compared to the BlenDR 2-channel packing method.

BlenDR with video colorization. Table 3 presents a comparison between BlenDR with video colorization and the conventional single-stream depth video using triangle depth packing. The baseline depth-only stream, while providing depth information, does not offer color information. BlenDR provides 10 times more accurate depth in RMSE, alongside outstanding RGB quality—SSIM of 0.96 and a PSNR of 35.8 dB, even though the conventional single-stream depth video uses 75% more bandwidth. The high RGB quality comes from the Y channel, which contains most of the visual information.