Aalto University

OPEN ACCESS

This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Siekkinen, Matti; Kämäräinen, Teemu

Neural Network Assisted Depth Map Packing for Compression Using Standard Hardware Video Codecs

Published in:
ACM Transactions on Multimedia Computing, Communications and Applications

DOI:
10.1145/3588440

Published: 07/06/2023

Document Version
Publisher's PDF, also known as Version of record

Please cite the original version:
Siekkinen, M., & Kämäräinen, T. (2023). Neural Network Assisted Depth Map Packing for Compression Using Standard Hardware Video Codecs. *ACM Transactions on Multimedia Computing, Communications and Applications*, *19*(5s), 1-20. Article 174. https://doi.org/10.1145/3588440

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

Neural Network Assisted Depth Map Packing for Compression Using Standard Hardware Video Codecs

MATTI SIEKKINEN, Aalto University, Department of Computer Science and University of Helsinki, Department of Computer Science
TEEMU KÄMÄRÄINEN, University of Helsinki, Department of Computer Science

Depth maps are needed by various graphics rendering and processing operations. Depth map streaming is often necessary when such operations are performed in a distributed system and it requires in most cases fast performing compression, which is why video codecs are often used. Hardware implementations of standard video codecs enable relatively high resolution and frame rate combinations, even on resource constrained devices, but unfortunately those implementations do not currently support RGB+depth extensions. However, they can be used for depth compression by first packing the depth maps into RGB or YUV frames. We investigate depth map compression using a combination of depth map packing followed by encoding with a standard video codec. We show that the precision at which depth maps are packed has a large and nontrivial impact on the resulting error caused by the combination of the packing scheme and lossy compression when the bitrate is constrained. Consequently, we propose a variable precision packing scheme assisted by a neural network model that predicts the optimal precision for each depth map given a bitrate constraint. We demonstrate that the model yields near optimal predictions and that it can be integrated into a game engine with very low overhead using modern hardware.

CCS Concepts: • **Computing methodologies** → **Image compression**; **Neural networks;**

Additional Key Words and Phrases: Depth map, video encoding, neural network, game engine

ACM Reference format:
Matti Siekkinen and Teemu Kämäräinen. 2023. Neural Network Assisted Depth Map Packing for Compression Using Standard Hardware Video Codecs. *ACM Trans. Multimedia Comput. Commun. Appl.* 19, 5s, Article 174 (May 2023), 20 pages.
https://doi.org/10.1145/3588440

1 INTRODUCTION

Depth is an important metric in graphics rendering and processing. A depth map is a representation of the distance of pixels to the camera that recorded the image. Depth maps are necessary, for example, in rendering to handle occlusions properly and in computer vision to perform **three-dimensional** (**3D**) object detection.

With the introduction of various distributed architectures for rendering and graphics processing, it has become necessary to transmit depth information across a network efficiently. **Depth**

Authors' addresses: M. Siekkinen, Aalto University, Department of Computer Science, P.O. Box 15400, FI-00076 AALTO, Finland; email: matti.siekkinen@aalto.fi; T. Kämäräinen, University of Helsinki, Department of Computer Science, P.O. Box 4, 00014 University of Helsinki, Finland; email: teemu.kamarainen@helsinki.fi.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2023 Copyright held by the owner/author(s).
1551-6857/2023/05-ART174 $15.00
https://doi.org/10.1145/3588440

image based rendering (**DIBR**) uses a color image supplemented with depth information in order to reproject that image to a new viewpoint [13]. DIBR enables immersive video viewing with **6 degrees of freedom** (**6DoF**) in which multiple video streams from different viewpoints can be reprojected to create a completely new view from a perspective not directly provided by any of the individual cameras [3]. In remote **rendered augmented** (**AR**) and **virtual reality** (**VR**) where graphics are rendered by a remote server and streamed as video to a thin client device, latency between the server and the client causes frames to be rendered from a slightly incorrect viewpoint when the client device moves. In this case, DIBR can be used to compensate for latency by streaming also depth and reprojecting incoming video frames to the current viewpoint of the client device [23]. In the split rendering system presented in [24], global illumination is computed remotely and streamed as compressed light probe data which includes depth. In this way, the system enables dynamic high-quality lighting on a resource constrained device, such as a standalone **virtual reality** (**VR**) headset.

Similar to color videos, transmitting depth information requires fast performing compression, which is why video codecs are often used. Both extensions for standard codecs as well as custom solutions for depth compression have been developed. However, currently none of these solutions are enabled with hardware accelerated encoding and decoding (e.g., Nvidia NVENC [17], AMD AMF [1], and Intel Media SDK [8]). We focus solely on solutions that are feasible for real-time encoding of rendered graphics using currently available (cloud) server hardware and for real-time decoding using currently available mobile devices, such as phones and standalone VR/AR devices. For this reason, using hardware implementations of standard codecs is the only feasible way currently to stream depth with high enough resolutions and frame rates. This in turn requires packing depth maps into YUV or RGB video frames prior to video compression because the RGB-D extensions for standard codecs are not supported by current hardware implementations. The dedicated piece of hardware for video coding also makes sure that the CPU and GPU of the device are not burdened with depth map encoding or decoding, as in a typical scenario they are required for other operations, such as graphics rendering at server side and warping the incoming depth images at client side.

In this article, we investigate different ways of packing depth maps extracted from computer generated game scenes into color textures so that the resulting bitrate-accuracy tradeoff is as good as possible when the depth maps are compressed and decompressed using a standard video codec. We have used H.264 [34] in this work but the methods can be applied to other video codecs with available hardware encoder and decoder implementations, such as H.265 [25], as well. Specifically, we examine the combined impact of depth packing precision, i.e., how many bits are used to represent depth, and lossy video encoding on the resulting depth error when bitrate is constrained. Our results show that there is an optimal precision to use in depth map packing that depends on the target bitrate. In other words, while naive 8-bit depth packing (e.g., grayscale image encoding) provides too low precision considering that game engines can typically provide up to 24-bit depth when rendering to a texture, it is also generally not useful to pack depth using maximal number of bits (24) because the benefit of inter-frame compression diminishes with increasing precision.

The results also reveal that the optimal number of bits to use in depth map packing for video encoding is not constant but depends on the game scene and even varies between frames from the same scene. To cope with this, we adapt a neural network model to predict the optimal depth precision to use given a depth map and bitrate constraint. Evaluation results reveal that the model outperforms manually extracted baseline and produces near optimal predictions. The model generalizes for different trajectories of the same scene and the results suggest that it can also generalize across different scenes if provided with sufficiently diverse training data, alleviating the need to fine tune the model for each different scene/game. We benchmark the performance of the model in

different scenarios and demonstrate that a prediction can be inferred in under a millisecond with a reasonable resolution using modern hardware and optimized inference engine. We also show that integrating the model into a game engine introduces only 0.5–2 ms of overhead depending on the hardware.

2 RELATED WORK

Many different solutions have been proposed to compress depth maps over the years. Concerning standards, a 3D extension has been defined for the HEVC standard introducing several techniques to improve coding of depth maps [26]. A number of improvements to HEVC depth coding tools have also been proposed in recent years (e.g., [11, 20, 22]). However, there is no support yet for hardware-accelerated coding using this standard, even though work on designing such efficient hardware has been conducted (e.g., [2, 21]). Therefore, these solutions are unfeasible currently when high resolution and frame rate depth map streaming is required, which is the case with immersive applications, for instance. This is especially the case with mobile phones and standalone VR/AR headsets that have limited power and computational resources.

The **MPEG Immersive Video** (**MIV**) specification is codec agnostic but the common test conditions of the working group used 10-bit depth with the HEVC Main 10 profile (i.e., grayscale images) [3]. In this way, hardware encoding and decoding are enabled with most modern graphics cards and embedded devices but the depth precision is currently limited to 10 bits unless a specific depth packing method is applied. An alternative approach to the depth map encoding and streaming within MIV is to estimate depth from the received multiple different streamed views and even a compromise between the two is possible as shown in [5]. However, these solutions require multiple incoming video streams, and the depth estimation process is computationally much too demanding to be performed in real-time on resource constrained mobile devices.

The light probe streaming solution presented in [24] packs depth information directly into YUV channels according to a predefined bit arrangement which is then streamed using lossless HEVC encoding, hence providing high-performance also on mobile devices. While lossless encoding maintains depth maps intact, it is very inconvenient for all scenarios where bandwidth is limited because the resulting bitrate cannot be controlled. In this article, we specifically examine the behavior with lossy compression where the encoder rate control is given a specific bitrate target. This behavior is typical for real-time streaming protocols, such as WebRTC, that need to react to varying available bandwidth in a frame-by-frame manner.

Another lossless depth encoding scheme is presented in [35]. It is shown to perform fast compared to regular image compression alternatives on a PC and is a good candidate for such deployments. However, in addition to rate control being impossible, as a custom method it, unfortunately, does not enjoy the benefit of dedicated hardware acceleration as standard codec-based methods do, and it is, therefore, unlikely to be a feasible solution for high resolution and frame rate depth streaming to/from mobile, standalone devices.

In the telepresence systems presented in [4, 6], depth images are encoded as 8-bit grayscale images while in [12] the precision is set to 12 bits. All three solutions use the H.264 standard codec. The three proposed solutions use lossy, lossless, and hybrid lossless-lossy compression, respectively. The hybrid scheme compresses the most significant bits of depth images in lossless and the rest in lossy manner. In contrast to these works, we do not preset the depth precision but instead search for the optimal precision when applying lossy compression and given a bitrate target.

A custom depth image and video codec are proposed in [9]. The solution targets fast decoding by running the decoder entirely on GPU. However, the fixed 4:1 compression ratio provided by the solution is modest and does not allow dynamic rate control. In addition, it is only benchmarked

with many parallel depth streams on a PC with relatively powerful GPU. Hence, it is unclear how it would perform on a resource constrained device when decoding a single depth stream.

Most closely related work to ours is presented in [19] which introduces a scheme to pack 16-bit depth into YUV textures for compression with standard codec. The method aims at reducing quantization and chroma subsampling effects as much as possible but provides a knob that can be turned to adjust the bitrate-accuracy tradeoff. Unfortunately, the article does not explore the impact of adjusting that parameter at all. In this work, we use the depth packing scheme presented in the article in some of our experiments in Section 3 and examine the bitrate-accuracy tradeoff when adjusting the packing scheme parameter. The neural network assisted parameter optimization approach we propose can be applied to their packing scheme too.

3 DEPTH MAP PACKING FOR STANDARD VIDEO ENCODING

In this section, we present two different packing schemes and investigate what kind of bitrate–depth accuracy tradeoff they provide when used in conjunction with a standard H.264 video codec.

3.1 Packing Schemes

A depth map consists of a value per pixel and the values can describe a linear or nonlinear distance of the pixel from the camera. They can also describe disparity in the case of multiple cameras (e.g., stereo view). In order to pack a depth map into a color texture, the values need to be normalized. Game engines limit the depth range that cameras see by introducing so called near and far planes. Any object closer than the near plane or farther than the far plane will not be visible. The values stored in a depth buffer of a game engine renderer are typically nonlinear and normalized to $[0, 1]$ range so that zero and one correspond to the depth of the near and far planes, respectively. Depth value normalized in this way is in practice stored into a 32-bit single-precision floating-point number (i.e., FP32) or an unsigned integer so that it can hold the depth at sufficient precision. The Unity game engine [30], for instance, provides up to 24-bit precision depth when rendering to a texture.

The required depth precision and its impact on rendered image quality depends largely on the application. Even with a specific application, such as DIBR, it is likely to be very much specific to the scene and viewing setup. For instance, users may perceive warping artefacts with DIBR differently on a phone compared to a head-mounted display. Depth precision has been studied in computer graphics (e.g., [14, 33]) and current game engines use up to 32-bit depth in rendering. The higher the precision, the more accurate will be the outcome of the process for which the depth map is used. Hence, if there is spare bandwidth available, it makes sense to use it to reduce the resulting error in the streamed depth maps. Naturally, this requires that the original depth map is of high quality, which may not be the case when the graphics are not computer generated. With natural imagery, for instance, the obtained depth information may not be sufficiently accurate to merit high precision encoding, depending of course on the equipment and methods to measure or estimate depth.

When packing a depth map for video compression using a standard codec, depth of each pixel is typically packed directly into the YUV channels so that the resulting texture is directly submitted to the video encoder without RGB to YUV color space conversion. We consider both YUV444 format, i.e., each channel is represented by 8 bits, and YUV420 format, i.e., the chroma channels UV are subsampled using 2×2 pixel blocks, which effectively reduces the number of bits per pixel to 12. Packing depth directly to YUV makes it easier to ensure that the most significant bits of the depth value are not affected by possible chroma subsampling by placing them to the Y channel. As chroma subsampling (YUV420) basically averages the chroma channel values over the neighboring four pixels, it, therefore, introduces error to the less significant parts of the depth values packed to those channels. As it is a compression technique designed for visual images perceived by humans taking

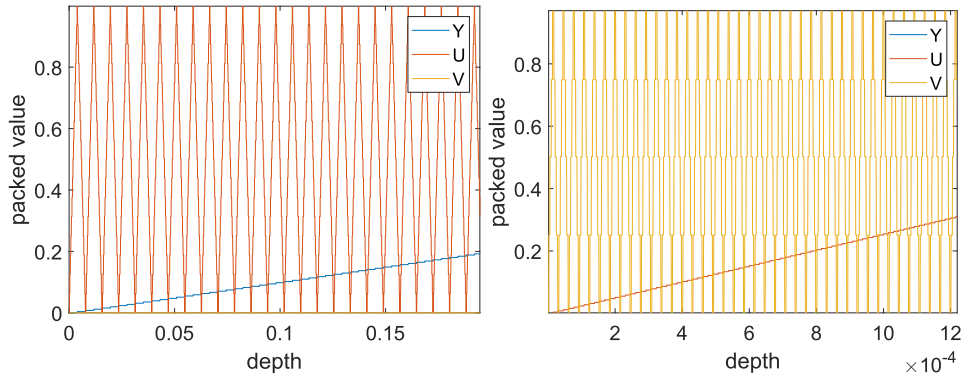
ALGORITHM 1: Variable bit packing (VBP)

```
inline float3 VariableBitPacking(float depth, uint bits) {
    if (depth == 1.0f) {
        return float3(1.0f, 1.0f, 1.0f);
    }
    float4 scale = float4(1.0f, 255.0f, 65025.0f, 16581375.0f);
    float3 ogb = float3(65025.0f, 255.0f, 1.0f) / 16581375.0f;
    float4 unit = float4(depth, depth, depth, depth);
    unit.gba -= floor(float3(unit.gba / ogb)) * ogb;
    float4 color = unit * scale;
    color = color - floor(color);
    color.rgb -= color.gba / 255.0f;

    //invert values to make triangle waves
    uint chromacbint = uint(round(float(color.g * 255.0f)));
    uint lumaint = uint(round(float(color.r * 255.0f)));
    if (chromacbint % 2 > 0) {
        color.b = 1 - color.b;
    }
    if (lumaint % 2 > 0) {
        color.g = 1 - color.g;
    }

    //reduce precision to specified number of bits
    float bits_to_use = min(8.0f, bits - 8.0f);
    if (bits_to_use > 0) {
        color.g = color.g * pow(2.0f, (bits_to_use - 8));
    }
    bits_to_use = min(8.0f, bits - 16.0f);
    if (bits_to_use > 0) {
        color.b = color.b * pow(2.0f, (bits_to_use - 8));
    }
    if (bits == 8) {
        color.g = 0;
    }
    if (bits <= 16) {
        color.b = 0;
    }
    return color.rgb;
}
```

advantage of the fact that our visual system is less sensitive to color than luminosity changes, and not for depth maps, it is somewhat harmful when applied to depth maps as we will show in Section 3.4. Video encoders typically also allow specifying a quantization parameter offset for the rate controller (chroma QP offset) in order for it to quantize the chroma channels more than the Y channel. The effect with depth packing would likely be similar to using a lower depth precision and we do not experiment with this encoder feature in this work.

There are two sources of error introduced when compressing packed depth maps into video. First, the packing scheme itself may introduce error if it does not employ as high depth precision per pixel as the original depth map obtained from the renderer's depth buffer. Second, the video compression is typically lossy which introduces error in the quantization phase of compression. We define the *mean depth map error* as the absolute difference of pixel depth in the original depth map and the depth map that has been packed, compressed, decompressed, and unpacked averaged over all the pixels. We use this metric to quantify the quality of the packed and compressed depth maps in the rest of the article.

3.1.1 Variable Bit Packing (VBP). VBP is the first of the two packing schemes that we present here and the one that we have designed. We implement the packing method as a shader in order to run it efficiently on GPU (see Section 3.2). The shader function code for the VBP packing is shown in Algorithm 1. The code is written in HLSL and can be directly used in a shader with Unity, for



(a) 12-bit precision (V component is zero) (b) 18-bit precision (Y component is close to zero)

Fig. 1. Illustration of the resulting YUV color component values when applying VBP packing scheme with different precisions. Note the different x-axis (depth) scales. Y component overlaps with x-axis in (b) because its value is so close to zero.

example. VBP enables packing depth with different precision, i.e., different number of bits. When the algorithm is adviced to pack a given depth map using full 24-bit precision (the `bits` argument of the shader function specifies the precision), the VBP scheme straightforwardly packs the bits into the YUV channels so that the most, the second most, and the least significant 8 bits are stored to Y, U, and V channels, respectively. In the shader code, this corresponds to lines 5–11. In other cases, the depth float is quantized according to the specified precision (lines 24–37 in Algorithm 1). For example, when packing using 12-bit precision, the first 8 bits are packed to the Y channel and the remaining 4 bits as the most significant bits of the U channel, setting the rest of the bits of that channel as well as the whole V channel to zero. In addition, the values packed to the U and V channels are inversed modulo two of the Y and U channels (lines 14–21 in Algorithm 1), respectively, effectively producing triangle waves with linearly increasing depth. The reason for this is to avoid discontinuities with continuously increasing and decreasing depth, which would be subject to a more pronounced impact of quantization by the video encoder. The unpacking function (not shown here) simply performs the inverse operations. Figure 1 shows some samples of the resulting YUV values when packing depth using the VBP scheme.

3.1.2 Robust Packing (RP). RP was originally presented in [19] and it is the second packing scheme we use in our experiments. We have implemented it as a shader similarly to the VBP scheme. It has been designed to be robust against compression artefacts due to quantization and chroma subsampling. The scheme packs 16-bit depth maps so that an 8-bit coarse depth is stored into Y channel and the chroma channels carry partially redundant information encoded as triangle waves with same period but different phase.

More specifically, the scheme maps an integer depth value d to the RGB color channels $L(d)$, $H_a(d)$, and $H_b(d)$. $L(d)$ is a linear mapping of d into [0, 1] and it is interpreted as a low-depth-resolution representation of d. The other two components H_a and H_b are fast-changing, piece-wise linear functions (triangle waves) with slopes high enough to be expressed in the low-precision output representation. The period of the triangle waves can be adjusted using a parameter n_p which is the integer period of H_a and H_b in the input depth domain. This parameter value needs to be at most twice the number of output quantization levels ($n_p \leq 512$ for 8-bit output) in order for the packed depth map to be resistant to quantization according to the authors. The three components are visualized in Figure 2 with different values of n_p. Increasing the value of n_p beyond the specified

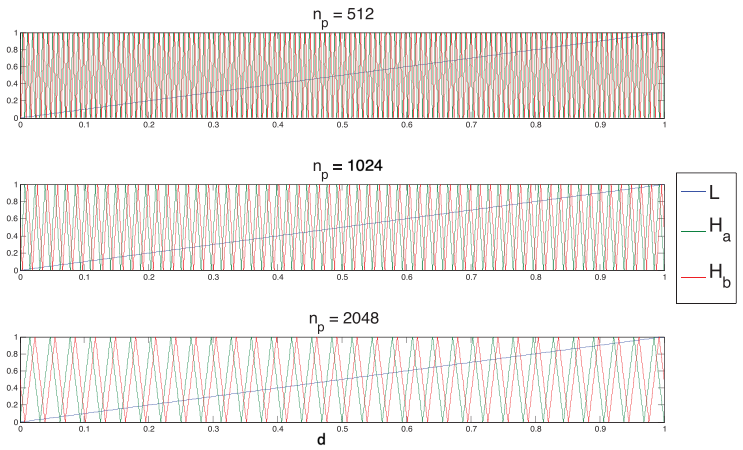


Fig. 2. $L(d)$, $H_a(d)$, and $H_b(d)$ for increasing values of n_p. X-axis is d [19].

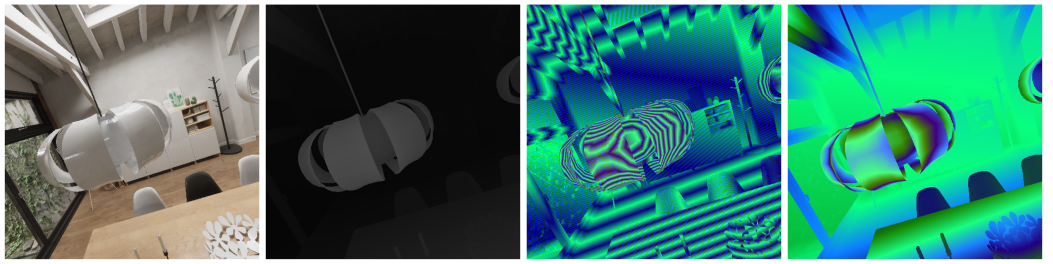
maximum target value (i.e., 512 in this case) provides a way to allow the packed depth map to gradually become more and more subject to quantization. Hence, the error due to compression increases but, as we will show in Section 3.4, the resulting bitrate of the compressed depth map video decreases. As mentioned by the authors, their scheme bears some resemblance to phase-shift encoding.

3.2 System Setup

In all the experiments, we used the Unity game engine supplemented with the WebRTC plugin from the Render Streaming package for real-time video streaming of the rendered graphics [31]. We modified the plugin to submit textures in the chosen YUV format to the video encoder, to override the WebRTC congestion control feedback to the encoder rate control so that it obeyed our bitrate target, and to dump the encoded video to a file. Depth maps were created by injecting a custom pass to the HD Render Pipeline. The custom pass uses a shader, such as the one in Algorithm 1, to directly pack the camera depth buffer into a render texture that is subsequently passed to the video encoder plugin. The depth buffer contained inversed non-linear depth normalized to $[0, 1]$ range according to the near and far plane, which were set to 0.1 and 100, respectively. This way of packing and encoding depth maps can be easily done in real-time at our target frame rates because the data does not leave GPU memory until it is read as a compressed video frame.

In addition, two other forms of depth maps were gathered. Non-packed high precision depth maps were extracted as the ground truth data by using a compute buffer to store the depth buffer values, reading that buffer to CPU memory, and storing it as bit array to a file. For a subset of frames, packed but uncompressed depth maps were extracted as a reference by saving the render textures as PNG image files before they are submitted to the video encoder so that we could also separately evaluate the impact of packing with given depth precision without video compression on depth map error. Afterward, to evaluate the quality of the resulting compressed depth maps, video dumps were decoded and/or unpacked, and compared to the reference and ground truth data using FFmpeg, OpenCL, and Python scripts.

For the video encoding, we used hardware accelerated H.264 codec provided by the Nvidia Video Codec SDK 11.0 run by the Titan V graphics card. The encoder was configured to use P1 preset (NVENC provides seven presets from P1 (highest performance) to P7 (lowest performance) to control performance/quality tradeoff), CBR rate control, ultra-low latency tuning



(a) RGB capture (b) 8-bit depth map (c) VBP w/ 24-bit precision (d) RP with $n_p = 4096$

Fig. 3. Sample frame from the Scandinavian House demo scene and corresponding depth map packed in three ways.

info, and other settings according to the recommendation by the Nvidia programming guide for "low-latency use cases like game-streaming, video conferencing and so on." [17]. The only exception to the recommendations was that we used one second long GOP (GOP length equal to the `frameRateNum` parameter in NVEncoder API with `frameRateDen=1`).

3.3 Scenes and Datasets

To generate all the results presented in Section 3.4, we used a photorealistic architectural visualization scene from Oneiros [18] called *The Scandinavian House demo (AVP Vol.6)* available in the Unity Asset Store [28]. The scene is a fully navigable interior of a Scandinavian house that includes more than 200 objects and 4K textures. The scene was adapted for VR and the target frame rate set to 90 fps. We recorded the camera trajectory (pose including rotation and position) while a user explored the different rooms in the virtual house for a total of 6 minutes using the Oculus Quest VR headset. The user was able to teleport to fixed positions located inside the house in different rooms and look around with 6DoF. Figure 3 displays a sample frame from this scene and the corresponding depth map packed in different ways.

The recorded trajectory was then repeatedly replayed so that each time either the ground truth depth data, packed reference depth maps, or compressed depth maps were grabbed with different depth packing and video encoding configurations. Temporal anti-aliasing was turned off to avoid depth jittering and all animations were disabled to have identical depth maps each time for packing and encoding. The replay resulted in compressed and ground truth depth maps corresponding roughly to 26k rendered frames of which roughly 3,200 were also extracted as packed but uncompressed reference frames. We name this dataset *HOUSE-VR*.

We also generated another dataset using the same scene with a prerecorded trajectory created with the Camera Path Creator asset for Unity. The resulting camera dynamics are very different from the VR usage and are closer to **first person shooter** (**FPS**) gaming as there is no head motion involved. The resulting dataset comprises depth maps corresponding to about 11k rendered frames. We name this dataset *HOUSE-FPS*.

In addition, we created datasets from two other scenes, namely the Fontainebleau demo scene [10] and the Unity HD Render Pipeline template scene [27]. Both of them are very high quality photorealistic scenes with snapshots shown in Figure 4. Both were used with an FPS camera controller and played for a few minutes by moving and looking around the scene resulting to depth maps corresponding to roughly 10k rendered frames. We name these two datasets *FBLEAU* and *HDRP*.

In all the experiments, depth maps were generated at a rate of 90 fps and using a spatial resolution of 512×512. Also, lower resolution 256×256 depth maps were generated for specific

(a) Fontainebleau demo scene (b) Unity HD Render Pipeline template scene

Fig. 4. Snapshots from the two other scenes used in the experiments.

experiments (Section 4.3). The 512 × 512 resolution was chosen because it still enables real-time performance (Section 5.1) and is still sufficient for good quality DIBR when applied together with 2k resolution color textures.

3.4 Impact of Bitrate on Depth Map Error

We now study what is the resulting mean depth map error of a stream of depth maps that has been packed using one of the two schemes with given parametrization and encoded to video using specific target bitrate.

Figure 5 shows the resulting mean depth map error averaged over all the frames for the different packing configurations and target bitrates for the HOUSE-VR dataset. We show separately results with subsampled (YUV420) and full rate sampled (YUV444) chroma channels. 8-bit packing is included as well, and it corresponds to grayscale packing in which the depth value is replicated to all the three color channels. To contrast these results, the bitrate of the ground truth depth maps (uncompressed 512 × 512 video) would be roughly 570 Mbps or 283 Mbps depending on whether YUV444 or YUV420 is used, respectively.

The most important observation is this: Increasing precision, i.e., the number of bits to use for packing with VBP, does not automatically result in reduced error after video compression and decompression. While increasing the precision reduces the error caused by the packing scheme itself, the error caused by quantization by the video encoder starts to increase after some point when the target bitrate remains constant. The reason for this is that it becomes increasingly difficult for the encoder to leverage inter-frame compression because small depth differences lead to relatively large differences in packed depth when using high precision. This means that there is a precision that provides the smallest error and it depends on the target bitrate. The RP scheme behaves similarly when we adjust the n_p parameter, which means that there is an optimal value for that parameter, and it similarly depends on the target bitrate. Chroma subsampling is harmful with both packing schemes and full rate chroma sampling improves the results, especially with higher target bitrates.

Figure 6 further illustrates what the resulting mean depth map error is composed of in different scenarios. We only show results with full rate chroma sampling (YUV444) case as results with chroma subsampling (YUV420) are similar. When packing precision is increased, the proportion of accuracy lost by the packing scheme decreases and error by video encoding increases, which can be observed in Figure 6(a) for each group of bars corresponding to a specific target bitrate where the bars represent packing precision that increases from left to right. The same applies to the RP scheme when the n_p parameter is decreased, again visible with each group of bars in Figure 6(b) but this time from right to left. An ideal parameterization of

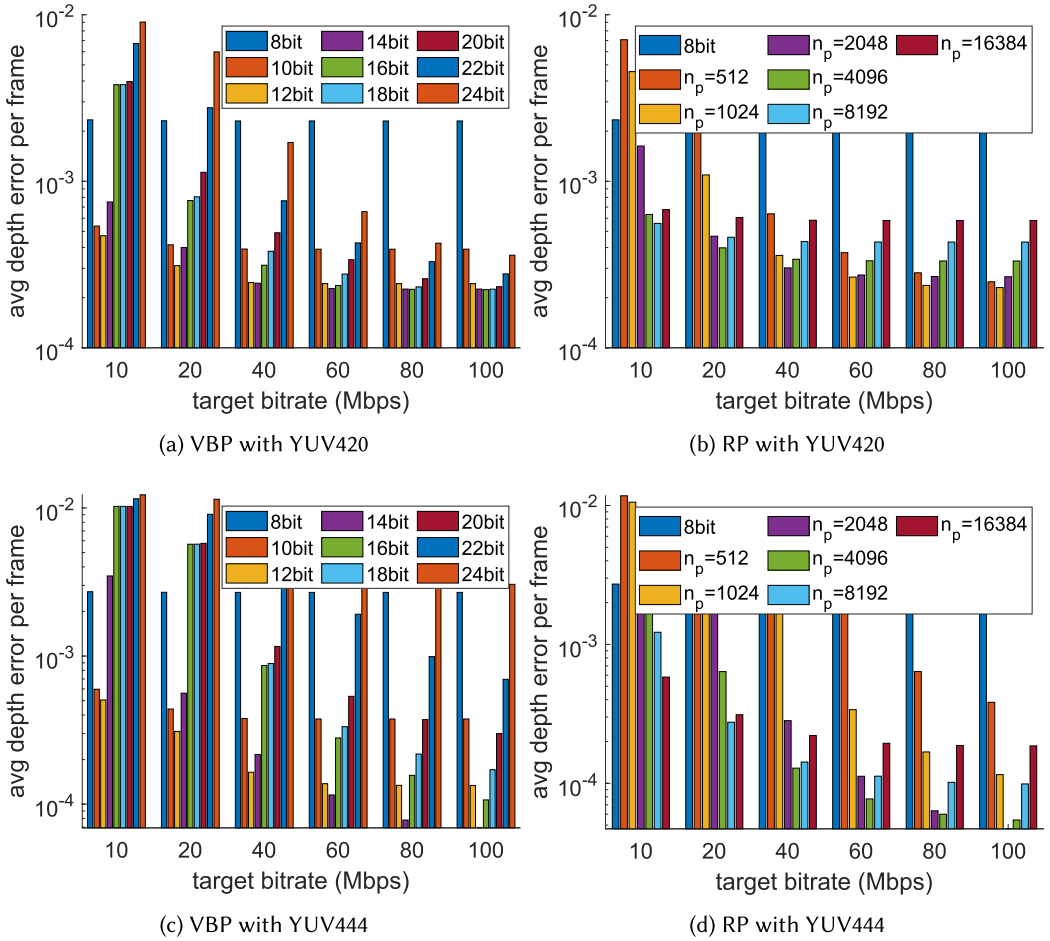


Fig. 5. Mean depth map error with packing schemes and target bitrates (HOUSE-VR dataset).

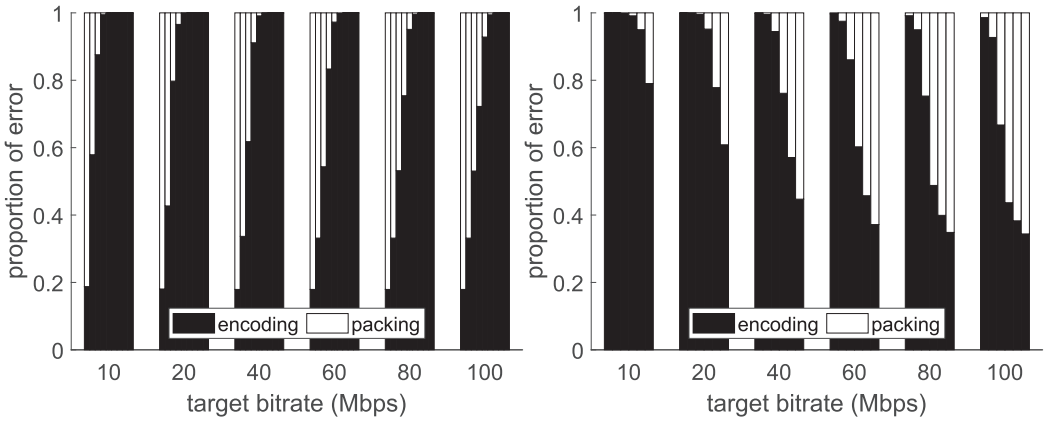
either of the two packing schemes would be such that optimally balances these two sources of error.

We examine the tradeoff between mean depth map error and bitrate in Figure 7, which plots the error against the resulting average bitrate (rate control may sometimes end up delivering bitrates that somewhat differ from the target). The lowest errors are achieved with the VBP when chroma channels are subsampled whereas RP works slightly better without chroma subsampling, but there is no clear difference between the two schemes.

3.5 Depth Map Specific Error

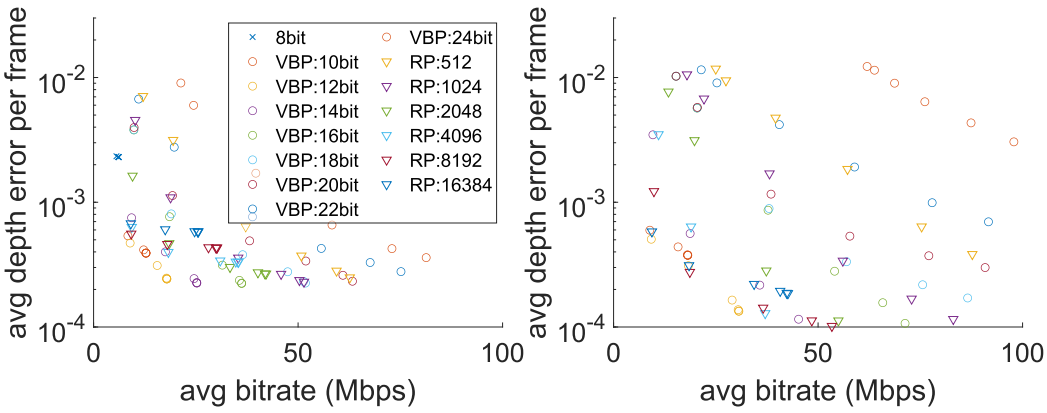
So far, we have established that there is a target bitrate dependent configuration that minimizes the overall mean depth map error for a specific depth packing scheme when we compute the error over all the depth maps of the stream. We now look at how the error varies between individual depth maps.

The scheme enables changing the packing configuration on a frame-by-frame basis, i.e., we can choose a different parameter configuration for each depth map. We computed the depth packing configuration (both VBP and RP considered) that yields the smallest mean depth map error for



(a) VBP (b) RP

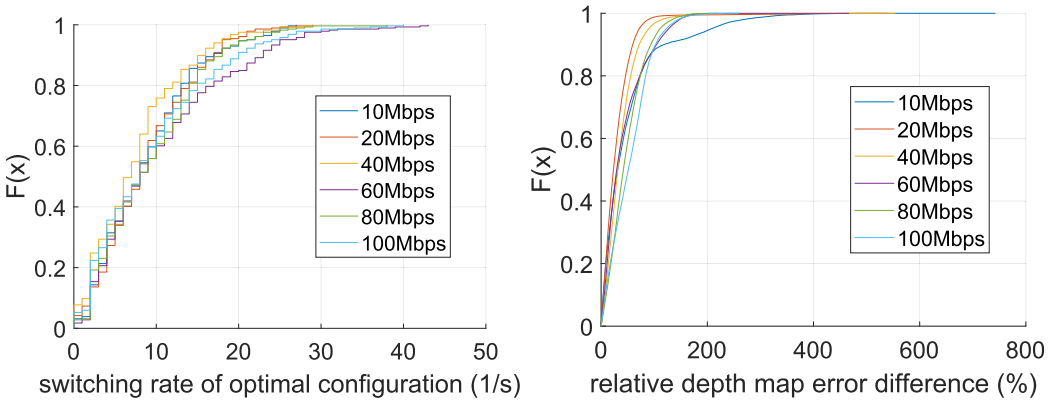
Fig. 6. Mean depth map error composition. For each group of bars specific to target bitrate, the bars correspond to results for depth bit values ranging from 8 (leftmost bar) to 24 (rightmost bar) for VBP and for n_p values ranging from 512 (leftmost) to 16,384 (rightmost) for RP, similarly to the bar groups in Figure 5.



(a) YUV420 (b) YUV444

Fig. 7. Mean depth map error vs. average bitrate.

each depth map with a given target bitrate. We then examined how frequently this optimal configuration changes within a stream of depth maps. In other words, if we were to always choose the optimal packing configuration for each depth map, how often would we need to switch from one configuration to another. Figure 8(a) plots a CDF of this frequency, which suggests that the optimal configuration is not the same for each depth map and in fact it changes quite frequently. Hence, choosing a constant packing configuration for a depth map stream is not ideal even if target bitrate stays constant during the entire streaming session. We also compare the mean depth map error between the optimal and second-best configurations by computing a relative depth map error difference so that we subtract the error using the optimal configuration from the error using the second-best configuration and divide the result with the former. Figure 8(b) plots a CDF of this value and shows that it can be significant. Therefore, even if we chose a constant depth packing scheme configuration that is almost optimal, the resulting depth map error may be substantially



(a) CDF plot of the frequency with which the optimal packing configuration changes.

(b) Comparison of depth map error with optimal and second best packing configurations

Fig. 8. CDF plots illustrating the stability of the optimal depth packing configuration for a given bitrate and the resulting depth map error compared to the second best configuration during a stream of depth maps.

larger than if we could always choose the optimal configuration. In summary, these result suggest that a depth map specific packing scheme would be useful.

4 NEURAL NETWORK ASSISTED PACKING

Inspired by the findings that there is an optimal packing configuration for a given target bitrate and, furthermore, that this configuration is depth map specific, we design a model to predict the best configuration prior to the actual packing. We apply transfer learning and adapt an existing neural network model to this task and tune it to predict the mean depth map error with different configurations. We consider here only the VBP scheme and YUV444 encoding but the model could as well be trained to predict the optimal n_p parameter for the RP scheme and YUV420 encoding.

4.1 Model

We repurposed the **residual neural networks** (**ResNet**) architecture for mean depth map error prediction. ResNet architecture was introduced by He et al. [7] and popularized the use of deeper architectures compared to previous work. This was made possible by using skip connections which reduce the information loss in deep networks.

We experimented with different variants of the ResNet architecture, namely ResNet-18, ResNet-34, ResNet-50, and ResNet-101. Out of these, we obtained better results (i.e., lower validation loss) with ResNet-50 than with ResNet-18 and ResNet34, while ResNet-101 did not further improve the results. Hence, we chose the ResNet-50 for our task. The final model is illustrated in Figure 9.

The ResNet model was modified to perform multi-output regression instead of classification by using L1 loss function (i.e., mean depth map error) and outputting predictions for each depth precision. We also added an expansion to the beginning of the architecture to expand a one channel depth image into three channels so that we could utilize pre-trained weights of networks trained with RGB input. The output of the model is the predicted mean depth map error for 8 different depth precision alternatives.

Main reason for choosing L1 is that we wanted to avoid an application (e.g., DIBR) specific metric. We did also consider alternative application-specific loss functions, such as **Structural Similarity** (**SSIM**) of the warped image using the packed and encoded depth map in DIBR scenarios. However, such loss would be highly dependent on the scene, camera (HMD in VR) motion,

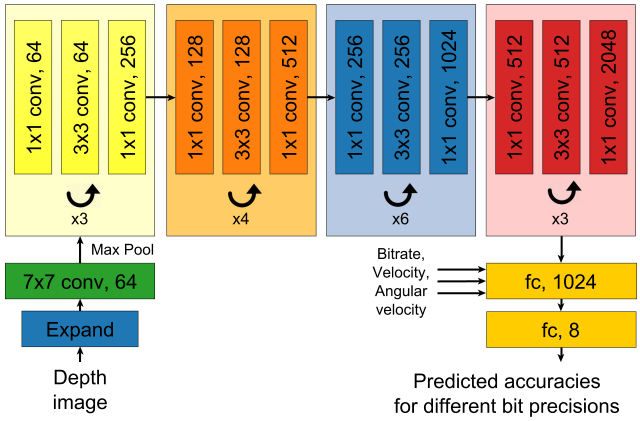


Fig. 9. Modified ResNet-50 model for depth precision prediction. Inputs are depth image, bitrate, (camera) velocity and angular velocity. The model outputs the predicted accuracies for different bit precisions.

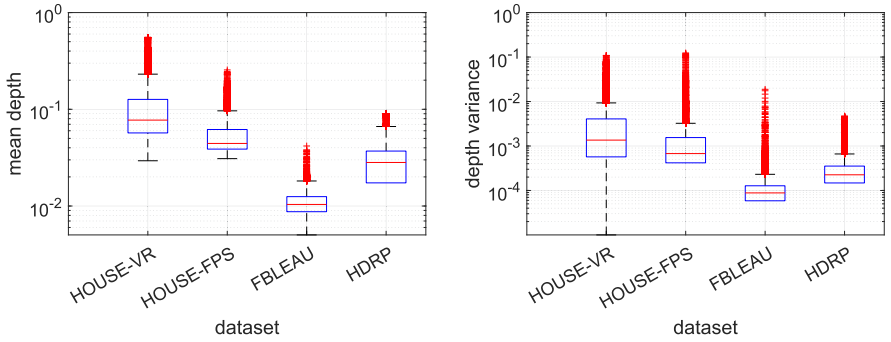


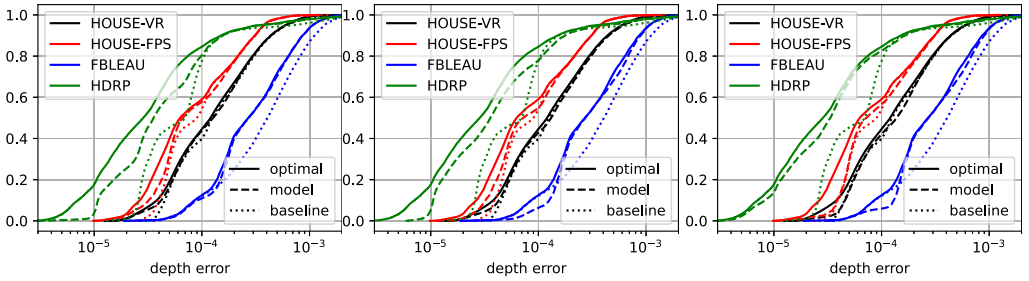
Fig. 10. Dataset statistics.

and latency if DIBR is used for latency compensation. These factors together will determine how much 3d warping will alter the original video frame(s) and, consequently, how much the depth map error will affect the warped outcome. Therefore, it raises questions on the generalizability of the resulting models: If one wants the loss to optimize the model for specific scenarios where L1 would not be ideal, the model might not perform well in other scenarios, whereas minimizing L1 does minimize also warping artefacts due to erroneous depth maps in general.

In order for the model to be able to predict the optimal precision with different target bitrates, we added a fully connected layer to the end of the network which also takes target bitrate of the video encoder as input. In addition, camera velocity and angular velocity (calculated using the previous and current depth map) are additional inputs to the layer. The reason for including velocities is to capture camera motion that affects the inter-frame compression of the video encoder. By training a model without the velocities as well, we confirmed that they provide useful information to the neural network and somewhat improve the prediction accuracy.

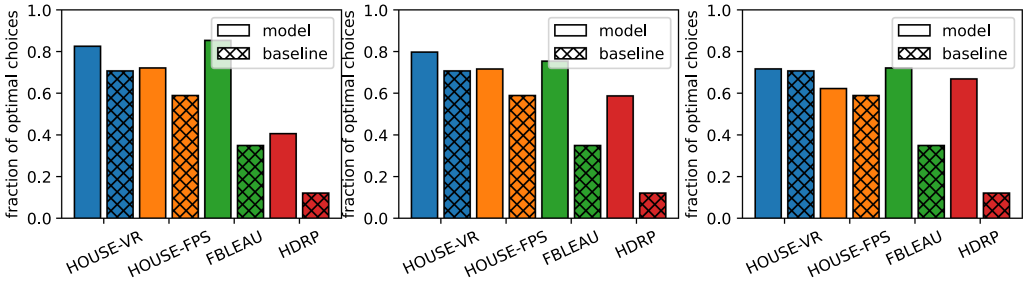
4.2 Training

We used the datasets described in Section 3.3 to train the model. Figure 10 shows distribution of the mean and variance of depth per depth map in the four datasets. There are substantial differences, particularly the FBLEAU dataset stands out with low mean depth values which refer to distant objects as the depth buffer values are inverted. Indeed, it is an outdoor scene, whereas the others



(a) Two-game model (2GM) (b) Three-game model (3GM) (c) 3GM trained on 256x256 input

Fig. 11. Distribution of mean depth map error.



(a) Two-game model (b) Three-game model (c) 3GM trained on 256x256 input

Fig. 12. Number of optimal choices.

are indoor. Datasets were split into non-overlapping train (80%) and test (20%) data. When multiple datasets were used to train a model, we included an equal number of samples from each one.

We observed that when training our models from scratch (randomly initialized) using our depth map datasets, they overfit. Using a less deep ResNet architecture does not behave in this way, which suggests that our datasets are simply too small or not diverse enough for training ResNet-50 from scratch. Therefore, we instead use a ResNet-50 pretrained for image classification on the ImageNet dataset (i.e., RGB images) and tune it with our depth map datasets to generate all the models presented in the next section.

We used the RAdam optimizer with default values (lr = 0.001, $betas$ = (0.9, 0.999), eps = 1e-8). Learning rate was decayed by a factor of 0.1 every 7 epochs, batch size set to 10, and each model trained for 30 epochs.

4.3 Results

In presenting the results, we mainly focus on the resulting mean depth map error and deliberately avoid using an application specific metric, such as an objective image quality metric applied to images warped using the depth map. Figure 11(a) shows the mean depth map error results with a model fine-tuned on a combination of HOUSE-VR and FBLEAU datasets. We call this model the *two-game model* as it contains data from two different games. They were calculated by choosing for each depth map the precision that yields the lowest mean depth map error according to the model. The baseline error was calculated by using the best performing precalculated configuration for each bitrate with the HOUSE-VR dataset, i.e., 12-bit precision for bitrates less than 50 Mbps and 14-bit precision otherwise (Figure 5(c)). Figure 12(a) shows the fraction of optimal choices by the model and baseline on the different datasets.

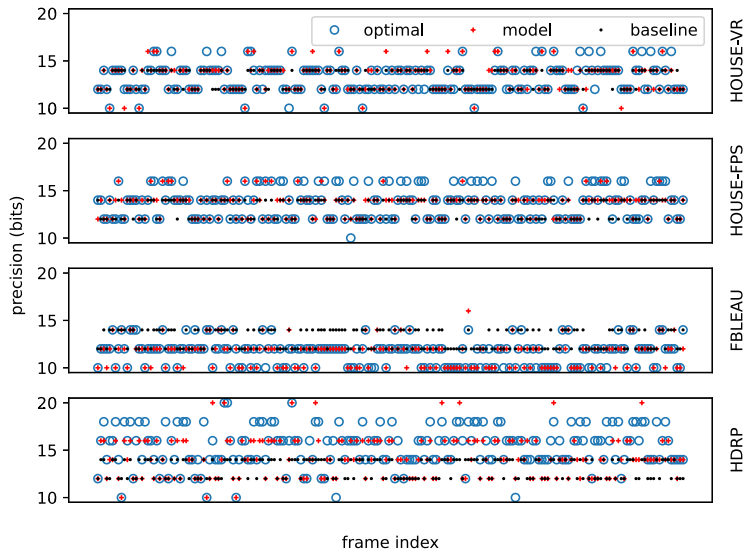


Fig. 13. Chosen precision using the two-game model for the first 200 depth maps of the HOUSE-VR test data. The test data includes all the target bitrates.

The model yields nearly optimal decisions for the datasets that were used in training. The baseline performs almost as well with the HOUSE-VR that it was extracted from. However, we notice clearer differences with the other datasets. In particular, the model beats the baseline on the HOUSE-FPS dataset which suggests that it generalizes across different trajectories of the same dataset, as HOUSE-FPS was not included in training. Moreover, compared to the baseline, the model shows much better, although not quite optimal, accuracy on the HDRP dataset, representing a game scene not included in the training. For the unseen HDRP dataset, the model still clearly beats the baseline (Figure 11(a)). We do believe that with a large and diverse enough dataset, the model will generalize better to unseen game scenes. In addition, there is always the option to fine tune the model with new game scenes.

We also trained a three-game model using a combination of all the three different game scenes, i.e., HOUSE-VR, FBLEAU, and HDRP datasets. The accuracy of the model clearly increases on the HDRP test data compared to the two-game model, while there is a minor penalty with the test data from the other datasets (Figures 11(b) and 12(b)).

Figure 13 visualizes the precision chosen by the model and baseline compared to the optimal for the first 200 depth maps. There are clear differences between the datasets in terms of the optimal precision and the way it varies. The model is able to adapt reasonably well, whereas a fixed baseline policy obviously is not. We would like to point out that choosing a sub-optimal depth precision only increases the depth map error according to Figure 11 and does not affect the depthmap stream bitrate in any way.

We also trained a model using lower resolution depth maps as input. In this way, the model learns to predict higher resolution (512×512) mean depth map error using smaller resolution (256×256) input, which is advantageous from inference speed perspective (Section 5.1). Figures 11(c) and 12(c) show that the results are similar to those with 512×512 input. The overall error is slightly smaller on the HDRP dataset and slightly larger with the other datasets. It would be interesting to investigate how high resolution depth map error could be predicted from a low resolutions (e.g., 256×256) input but we leave that for future work.

In case different encoded depth map resolutions are required, the model could possibly be extended for multi-resolution inference by adding resolution as an additional input to the fully connected layer at the end of the network and training it on multiple resolution datasets. Alternatively, one can train separate models for each resolution needed. Otherwise, the evaluation results suggest that the model can cope with different game scenes and camera trajectories, provided that the training data is sufficiently diverse. This is important for deployment considerations as it reduces the need to fine tune the model for each different game and/or scene.

5 PERFORMANCE AND DEPLOYMENT

Depth maps need to be encoded in real-time in many application use cases. In remote rendering applications, the frame rate requirements can range from 30–60 frames per second for cloud gaming, up to 90–120 frames per second for remote rendering of VR graphics. For this reason, the neural network which makes the decision for optimal depth map encoding settings needs to be relatively light-weight so that it does not significantly delay the complete system pipeline. In this section, we benchmark the bit depth model with different runtimes and by using reduced precision. Finally, we integrate the model into an existing game engine.

To put the results that follow into perspective, the naive alternative would be to compute the resulting depth map error separately for each depth map in order to determine the optimal precision. This means repeatedly encoding and decoding a 512×512 resolution depth frame 9 times with the different precisions of the VBP scheme and then selecting the precision that yields the lowest depth map error, which would require roughly 10–20 ms to only perform the video encoding part using the hardware video encoder of a modern Nvidia graphics card. In addition, one would need to account for the decoding time. This latency is infeasible for 90–120 fps real-time operation. Things would naturally become even worse with higher depth map resolution and in the case that 1-bit precision granularity was desired in determining the optimal precision.

5.1 Inference Runtime Benchmarks

Neural network throughput is highly dependent on the level of optimization for hardware acceleration on the used platform. Overall throughput is also dependent on the used batch size. To evaluate the inference time and throughput of our neural network model in a real-time usage scenario, we benchmarked the model with batch size 1 with a subset of the HOUSE-VR dataset (12,281 depth maps), measuring the median GPU inference times for single depth map inference. Batch size of 1 mimics an application that generates depth maps on-the-fly for example as a part of a graphics pipeline.

The model was developed with the PyTorch framework. This framework, run on a modern desktop PC with single-precision floating-point format (FP32) is the baseline for our evaluation. We used depth map sizes of 512×512 and 256×256 as our input. The first bar group (left) in Figure 14 shows the baseline inference latency of our model for both a currently high-end GPU (Nvidia RTX 3080 Ti) and a middle-range GPU (Nvidia RTX 2060) for the two input sizes.

The standard PyTorch runtime with full FP32 precision can run the neural network for bit depth decision in 5.4 to 6.0 ms for the RTX 3080 Ti and in 6.6 to 15.8 ms for the RTX 2060 depending on the used depth map size. This performance of the standard PyTorch runtime could be enough even for some real-time scenarios. However, as the bit depth decision model should ideally be only a small addition to the pipeline of the rest of the system, more optimized runtimes should be used possibly with reduced precision operations to accelerate the inference.

Runtimes optimized for the underlying hardware can dramatically accelerate the inference speed. As we use Nvidia GPUs, we converted our model to a TensorRT engine. TensorRT is a high-performance deep learning inference optimizer and runtime for Nvidia graphics cards [15].

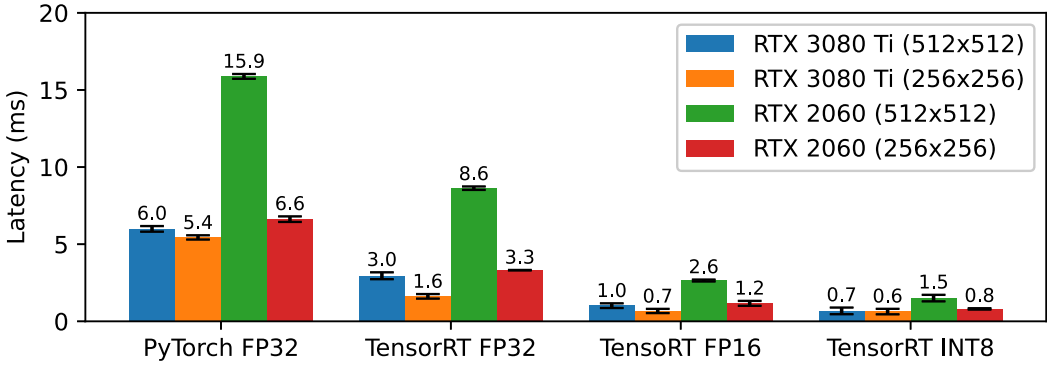


Fig. 14. Median inference latencies for different runtimes, precisions and GPUs (batch size 1, depth map size 512×512 or 256×256).

Figure 14 shows that the use of the TensorRT runtime roughly halves the inference latency and thus doubles the throughput with the same FP32 precision.

5.2 Reduced Precision

TensorRT runtime supports quantization, where lower precision weights and activations together with lower precision math are used to speed up inference. We configured the engine to use reduced precision and re-run the benchmarks for both GPUs using half-precision floating-point format (FP16) and unified 8-bit (INT8) precision to measure the latency gains. For INT8, we also enabled FP16 in the TensoRT engine creation configuration. This allows the framework to automatically choose the best (fastest) mixed precision operation for available hardware.

The inference latency drops to 0.7/1.0 ms for FP16 using the Nvidia RTX 3080 Ti and further to 0.6/0.7 ms with INT8 precision with depth map sizes 256×256 and 512×512 respectively. Similar relative gains can be achieved using the less powerful RTX 2060 graphics card. With reduced precision both GPUs achieve inference latencies around 1 ms which is promising for applying the bit depth decision model for use in real-time applications as a part of the graphics pipeline. Reduced precision can reduce the accuracy of the model, especially with depth maps that have higher precision per channel compared to regular RGB images, but the models can be calibrated to compensate for this. In our experiments FP16 quantization did not affect the model accuracy and no calibration was needed, whereas calibration was necessary with INT8 inference. After using the built-in calibration of TensorRT, the calibrated model with INT8 inference experiences only a small loss in accuracy (1–4% loss in the fraction of correct choices) compared to the non-quantized model. The calibration phase computes a scale value for each tensor in the network based on representative sample input values. We used 1,000 input samples with batch size 60 with the *MinMax Calibrator* of the TensorRT SDK [16].

5.3 Game Engine Integration

Depth maps are often produced one at a time for example by a graphics rendering pipeline of a real-time application. Previously, we showed that our neural network model can be run in 0.6–1.5 ms on the GPU depending on the hardware and precision used. Graphics pipeline integration is however more difficult as the input needs to be fetched after the depth pass and the results of the network need to be available for the encoder as soon as possible when the frame rendering is finished. The inference needs to happen without stalling the rendering pipeline as this could lead to low frame rates.

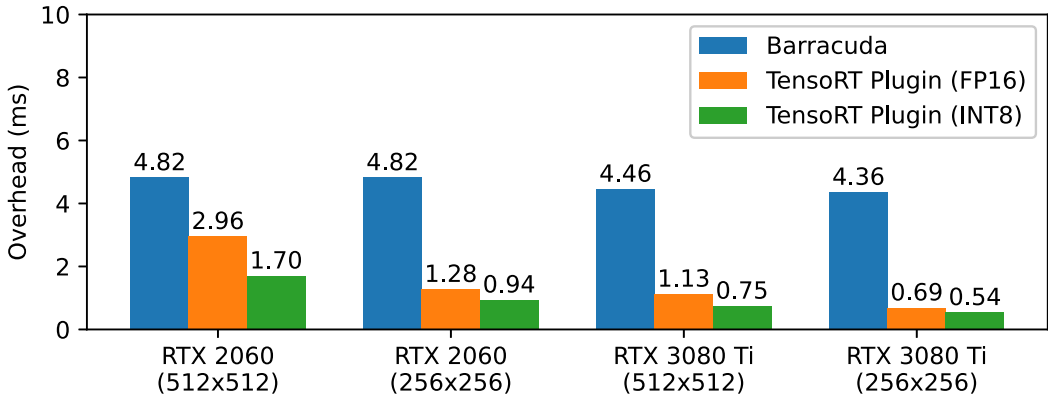


Fig. 15. Overhead in frame time (lower is better) comparison between Barracuda and TensorRT plugin for neural network integration into the Unity game engine using the Nature Starter Kit 2 demo scene with different GPUs (depth map size 512×512 or 256×256).

We integrated our neural network model for bit depth decision into the popular game engine Unity [32]. Unity has a cross-platform Neural Networks inference library named Barracuda. It utilizes OpenGL based kernels for neural inference. We exported our model in **Open Neural Network Exchange** (**ONNX**) file format from PyTorch and imported it into Unity Barracuda. We executed the bit depth model on each frame in a sample scene from a free Unity Asset *Nature Starter Kit 2* [29] using $1{,}920 \times 1{,}080$ resolution for frame rendering and with both 512×512 and 256×256 depth frame input sizes for the neural network. In addition, we developed a native (C++) Unity plugin which uses TensorRT for inference. Figure 15 shows the overhead per rendered frame in terms of extra delay caused by the inference with Unity's Barracuda implementation and with our TensorRT plugin. The Barracuda implementation has an overhead of 4.4 to 4.8 ms depending on the GPU. Our TensorRT plugin is capable of using reduced precision and has a lower overhead with only 0.5–3 ms added to the overall frame time of the application depending on the used GPU and precision.

One possibility to further facilitate deployment with new games and game scenes would be to apply continual learning. In other words, the model would be continuously fine tuned with data that is generated while a game is being played, e.g., in a cloud gaming system that leverages depth maps for DIBR to compensate for latency. This could be done by, e.g., sampling every n frames and adjusting n so that it would not burden the GPU and video codec too much since it requires packing, encoding, and decoding the sampled depth maps multiple times.

6 CONCLUSION

This article describes our study on computer generated depth map packing for compression using a standard video codec. The use of a standard video codec was motivated by the support for hardware acceleration also on mobile devices such as phones and standalone VR/AR devices. We showed that the precision used for depth map packing has a significant impact on the resulting depth map error given a bitrate constraint, which is caused by a combination of the packing scheme and lossy compression. Through measurements, we found that a depth map specific optimal packing configuration exists for a given bitrate. We adapt a neural network to predict the optimal depth precision for each depth map and outperform a manually extracted baseline by producing near optimal predictions. Finally, we show that the used neural network model can be run in real-time

on modern hardware with optimized runtimes and can be integrated into an existing game engine with very low overhead.

REFERENCES

[1] AMD. 2022. Advanced Media Framework (AMF) SDK. Retrieved November 25, 2022 from https://github.com/GPUOpen-LibrariesAndSDKs/AMF.

[2] Farouk Amish and El-Bay Bourennane. 2019. An efficient hardware solution for 3D-HEVC intra-prediction. *Journal of Real-Time Image Processing* 16, 5 (2019), 1559–1571.

[3] Jill M. Boyce, Renaud Doré, Adrian Dziembowski, Julien Fleureau, Joel Jung, Bart Kroon, Basel Salahieh, Vinod Kumar Malamal Vadakital, and Lu Yu. 2021. MPEG immersive video coding standard. *Proceedings of the IEEE* 109, 9 (2021), 1521–1536.

[4] Sam Ekong, Christoph W. Borst, Jason Woodworth, and Terrence L. Chambers. 2016. Teacher-student VR telepresence with networked depth camera mesh and heterogeneous displays. In *Proceedings of the International Symposium on Visual Computing*. Springer, 246–258.

[5] Patrick Garus, Felix Henry, Joël Jung, Thomas Maugey, and Christine Guillemot. 2021. Immersive video coding: Should geometry information be transmitted as depth maps? *IEEE Transactions on Circuits and Systems for Video Technology* 32, 5 (2021), 3250–3264.

[6] Simon N. B. Gunkel, Rick Hindriks, Karim M. El Assal, Hans M. Stokking, Sylvie Dijkstra-Soudarissanane, Frank ter Haar, and Omar Niamut. 2021. VRComm: An end-to-end web system for real-time photorealistic social vr communication. In *Proceedings of the 12th ACM Multimedia Systems Conference*. Association for Computing Machinery, New York, 65–79.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[8] Intel. 2022. Intel Media SDK. Retrieved November 25, 2022 from https://github.com/Intel-Media-SDK/MediaSDK.

[9] Babis Koniaris, Maggie Kosek, David Sinclair, and Kenny Mitchell. 2018. GPU-accelerated depth codec for real-time, high-quality light field reconstruction. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 1 (2018), 15 pages.

[10] Sebastien Lagarde. 2018. Photogrammetry in Unity: Making Real-World Objects into Digital Assets. Retrieved April 5, 2022 from https://blog.unity.com/technology/photogrammetry-in-unity-making-real-world-objects-into-digital-assets.

[11] Jie-Ru Lin, Mei-Juan Chen, Chia-Hung Yeh, Yong-Ci Chen, Lih-Jen Kau, Chuan-Yu Chang, and Min-Hui Lin. 2021. Visual perception based algorithm for fast depth intra coding of 3D-HEVC. *IEEE Transactions on Multimedia* 24 (2021), 1707–1720.

[12] Yunpeng Liu, Stephan Beck, Renfang Wang, Jin Li, Huixia Xu, Shijie Yao, Xiaopeng Tong, and Bernd Froehlich. 2015. Hybrid lossless-lossy compression for real-time depth-sensor streams in 3D telepresence applications. In *Proceedings of the Pacific Rim Conference on Multimedia*. Springer, 442–452.

[13] William R. Mark, Leonard McMillan, and Gary Bishop. 1997. Post-rendering 3D warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*. Association for Computing Machinery, New York, 7–ff.

[14] Nvidia. 2015. Depth Precision Visualized. Retrieved November 25, 2022 from https://developer.nvidia.com/content/depth-precision-visualized.

[15] Nvidia. 2022. NVIDIA TensorRT. Retrieved November 11, 2022 from https://developer.nvidia.com/tensorrt.

[16] Nvidia. 2022. NVIDIA TensorRT Dcoumentation: Post-Training Quantization Using Calibration. Retrieved November 21, 2022 from https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/#enable_int8_c.

[17] Nvidia. 2022. NVIDIA VIDEO CODEC SDK - ENCODER Programming Guide. Retrieved November 25, 2022 from https://docs.nvidia.com/video-technologies/video-codec-sdk/pdf/NVENC_VideoEncoder_API_ProgGuide.pdf.

[18] OneirosVR. 2019. ArchVizPro. Retrieved November 25, 2022 from https://oneirosvr.com/portfolio/archvizpro/.

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

[20] Mário Saldanha, Gustavo Sanchez, César Marcon, and Luciano Agostini. 2019. Fast 3D-HEVC depth map encoding using machine learning. *IEEE Transactions on Circuits and Systems for Video Technology* 30, 3 (2019), 850–861.

[21] Gustavo Sanchez, Mário Saldanha, Ramon Fernandes, Rodrigo Cataldo, Luciano Agostini, and César Marcon. 2020. 3D-HEVC bipartition modes encoder and decoder design targeting high-resolution videos. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 2 (2020), 415–427. DOI: https://doi.org/10.1109/TCSI.2019.2929977

[22] Liquan Shen, Kai Li, Guorui Feng, Ping An, and Zhi Liu. 2018. Efficient intra mode selection for depth-map coding utilizing spatiotemporal, inter-component and inter-view correlations in 3D-HEVC. *IEEE Transactions on Image Processing* 27, 9 (2018), 4195–4206.

[23] Shu Shi and Cheng-Hsin Hsu. 2015. A survey of interactive remote rendering systems. *ACM Computing Surveys* 47, 4 (2015), 29 pages. DOI: https://doi.org/10.1145/2719921

[24] Michael Stengel, Zander Majercik, Benjamin Boudaoud, and Morgan McGuire. 2021. A distributed, decoupled system for losslessly streaming dynamic light probes to thin clients. In *Proceedings of the 12th ACM Multimedia Systems Conference*. Association for Computing Machinery, New York, 159–172.

[25] 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.

[26] Gerhard Tech, Ying Chen, Karsten Müller, Jens-Rainer Ohm, Anthony Vetro, and Ye-Kui Wang. 2016. Overview of the multiview and 3D extensions of high efficiency video coding. *IEEE Transactions on Circuits and Systems for Video Technology* 26, 1 (2016), 35–49.

[27] Unity Technologies. 2021. New HDRP Scene template: Explore. Learn. Create. Retrieved April 5, 2022 from https://youtu.be/7gU7V-EENl4.

[28] Unity. 2021. Unity Asset Store - ArchVizPRO Interior Vol.6. Retrieved November 25, 2022 from https://assetstore.unity.com/packages/3d/environments/urban/archvizpro-interior-vol-6-120489.

[29] Unity. 2022. Unity Asset Store. Nature Starter Kit 2. Retrieved November 11, 2022 from https://assetstore.unity.com/packages/3d/environments/nature-starter-kit-2-52977.

[30] Unity. 2022. Unity Real-Time Development Platform. Retrieved November 11, 2022 from https://unity.com/.

[31] Unity. 2022. Unity Render Streaming. Retrieved November 11, 2022 from https://github.com/Unity-Technologies/UnityRenderStreaming.

[32] Unity Technologies. 2020. *Unity Core Platform*. Unity Technologies, San Francisco. Retrieved November 25, 2022 from www.https://unity.com.

[33] Paul Upchurch and Mathieu Desbrun. 2012. Tightening the precision of perspective rendering. *Journal of Graphics Tools* 16, 1 (2012), 40–56.

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

[35] Andy Wilson. 2017. Fast lossless depth image compression. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces*.

Received 30 June 2022; revised 10 March 2023; accepted 15 March 2023