ISSN 1213-6972 Journal of WSCG http://www.wscg.eu Vol.28, No.1-2, 2020

Improved Lossless Depth Image Compression

Roland Fischer Philipp Dittmann Christoph Schröder Gabriel Zachmann

University of Bremen, Germany
{rfischer, dittmann, schroeder.c, zach}@cs.uni-bremen.de

ABSTRACT

Since RGB-D sensors became massively popular and are used in a wide range of applications, depth data compression became an important research topic. Live-streaming of depth data requires quick compression and decompression. Accurate preservation of information is crucial in order to prevent geometric distortions. Custom algorithms are needed considering the unique characteristics of depth images. We propose a real-time, lossless algorithm which can achieve significantly higher compression ratios than RVL. The core elements are an adaptive span-wise intra-image prediction, and parallelization. Additionally, we extend the algorithm by inter-frame difference computation and evaluate the performance regarding different conditions. Lastly, the compression ratio can be further increased by a second encoder, circumventing the lower limit of four-bit per valid pixel of the original RVL algorithm.

Keywords

Lossless Compression, Depth Image Compression, RGB-D Streaming, Azure Kinect

1 INTRODUCTION

Over the past years, RGB-D sensors became an important topic in many research areas, including computer graphics and computer vision. Their popularity increased enormously when Microsoft released its first Microsoft Kinect, an inexpensive, small, and easy-to-use RGB-D camera, which captures rectangular color images and corresponding depth images. Since then, even smaller RGB-D sensors were developed, which are now integrated into many devices. Prominent examples are the Intel RealSense RGB-D cameras, the subsequently released Kinect V2, and Azure Kinect cameras from Microsoft, whose sensors can also be found in the augmented reality headsets HoloLens 1 and 2, and Lidar scanners. Depth images can also be calculated by using two RGB cameras and semi-global matching (SGM).

Common applications are, for example, telepresence [BKKF13, CK12, SST+18], Virtual/Augmented reality (VR/AR) applications with remote sensors capturing real word scenes [JSM+14], gesture and object recognition and tracking [CJK15], 3D reconstruction [NIH+11, WMS16] and SLAM (simultaneous localization and mapping) [MT17, WKJ+15].

In many cases, the data needs to be streamed over a network beforehand, which runs into the problem of limited network bandwidth, i.e., 100 Mbit/s or 1 Gbit/s for Ethernet, or 300-450 Mbit/s for 802.11n WiFi. The sensors accumulate a large amount of data, and the bandwidth becomes a limiting factor quickly, especially if multiple sensors are combined for better coverage. For instance, a sensor with Full HD color resolution and the depth with a resolution of 512×424 (Kinect V2) produces more than 6.6 MB of raw data per frame. At a typical frame rate of 30 Hz, the network would have to support at least 1.60 Gbit/s. More recent depth sensors often support even higher resolutions. For example, the new Azure Kinect is able to capture color, depending on the mode, up to a resolution of 4K, and has a one-megapixel depth sensor.

Data compression is essential in order to reduce the required bandwidth. This enables lower-bandwidth scenarios, makes room for other payloads, and increases the number of possible cameras. Individual compression of color and depth images is, in general, computationally less expensive than compressing reconstructed 3D data like point clouds or surfaces. Therefore, this approach is often preferred for real-time applications [LBW+15]. The color component of RGB-D sensors can easily be compressed with standard image and video compression algorithms like JPEG [Wal92] or H.264 [WSBL03] as they are optimized precisely for this task.

However, applying the same encoders to the depth data would often result in sub-optimal compression performance or, more crucial, severe artifacts, and geometric distortions [Wil17]. The reason for this is that depth

data usually is represented in a different format, 13 or 16-bit single channel, and has inherently different characteristics than natural color images. In general, depth images consist of more homogeneous regions with abrupt discontinuities at object borders. In addition, individual not-captured pixels and regions of invalid pixels are scattered throughout the image, if not filtered beforehand.

In recent years, the research and development of compression algorithms specially designed for depth images became an important topic. Creating efficient and, at the same time, geometry-preserving (lossless) ones is still a very active area of research.

Our work focuses on analyzing and improving the novel RVL algorithm [Wil17], especially regarding achieving higher compression ratios. More detailed, our main contributions are:

- An improved adaptive intra-image prediction step for the RVL algorithm, enhancing its compression ratio.
- The addition of a final entropy-coder stage, further enhancing the compression ratio.
- A multi-threaded implementation of the RVL algorithm and variants speeding it up further.
- An additional inter-frame delta step as well as an examination of its effectiveness regarding the compression ratio over different application scenarios.
- Extensive experiments comparing the effectiveness of various lossless depth compression algorithms for different range cameras.

2 RELATED WORK

Data compression is an important and widely used topic in computer graphics, image processing, and many other research areas. For example, [KÖP13] and [BÖK11] present methods to achieve compact representations of surface-light interactions like subsurface scattering and BRDFs. The output of RGB-D sensors is often visualized in 3D as mesh or point cloud; therefore, a lot of focus was put on compressing these geometric representations. In 2007 MPEG issued a call for proposals on point cloud compression to develop an ISO standard [SPB+19]. Point cloud compression algorithms are mostly based on spatial data structures like an octree. For example, [MBC17] is based on octrees and is also able to exploit temporal redundancies. [TCF16] introduced a time-varying approach that can predict graph-encoded octree structures between consecutive frames. Real-time capable mesh-based compression and streaming techniques like proposed by [MSA+13] and [BGTB12] are in most cases lossy algorithms. As both of these representations, point clouds, and meshes, are three dimensional, it is more challenging to find and encode redundancy in the data efficiently. Specialized data structures are needed and raise the complexity and computation time for high compression ratios.

A different approach is to encode the raw depth images. Many standard lossless image and video codecs like PNG [RK99], JPEG-LS [WSS96, WSS00] or H.264 [WSBL03] can be applied, but the results are rather poor, founded in the inherent differences between natural images and depth images [SMAP16, Wil17, HKR18].
Similarly, general-purpose compression algorithms can be applied on depth images, but intrinsically are not optimized for this kind of data. Therefore they are not ideal solutions.

In recent years some effort was made to adapt common video codes like H.264 and its successor HEVC [SOHW12] to suit depth data better. [PKW11] proposed an encoding scheme to convert the single-channel depth data to the RGB format used by H.264 and VP8, reducing the occurring artifacts. This technique still produces noise at the borders. [LBW+15] developed a hybrid lossless-lossy algorithm, where the first bits are encoded lossless, and the remaining ten bits are compressed using H.264. The HEVC standard [SOHW12] features an extension called 3D-HEVC designed for 3D video coding, which uses the multiview texture videos plus depth maps (MVD) format. This extension addresses the compression of depth images through multiple techniques. The complexity is rather high. Aimed at lowering it and enhancing the compression speed, [ZCH+15] proposed multiple modifications exploiting depth map and texture video correlation. Further speedups can be gained by limiting costly intra-image compression steps to regions, where they are effective according to a decision model based on tensor feature extraction, as proposed by [HE19].

Recently, [HKR18] proposed another technique in which even noisy depth images and videos are encoded through planar segmentation. Each frame gets segmented into a number of planes by using the Graph Cut algorithm over the image defined as a Markov Random Field. While the proposed method achieves a high Rate-Distortion performance, it is rather time-consuming and only effectively applicable to scenes that can be approximated by planes.

Most work of depth image compression is about lossy compression. Lossless solutions are quite rare. [MZC+11] combines run-length encoding and variable bit length coding for lossless encoding of depth images. [Wil17] took a similar but even simpler approach and achieves with the proposed RVL

algorithm higher speeds at comparable compression ratios.

3 OUR APPROACH

We created a pipeline of four steps (Fig. 1) to compress depth images losslessly from real sensors with a high compression ratio. If we have a sequence of images, we optionally calculate deltas between the frames. Afterward, we define spans and calculate individual predictors for each of them, before it will be compressed by RVL's [Wil17] alternating run-length and variable-length coding. To decrease the lower bound of RVL, we decided to use Zstandard as a final, optional step. The focus lies on captured 16-bit depth values, which can be handled as a single channel grayscale image. The corresponding color images are not considered in this work.

RVL consists of alternating run-length coding of zero-values, which represents invalid pixels, and variable-length coding for the rest. Like many compression algorithms, RVL compresses not the raw pixel values but the residuals, which remain after calculating the delta to a prediction of the current value. This leads to a decorrelation of the data, which in turn improves the effectiveness of subsequent compression steps. Smaller residuals have a low entropy, and fewer bits are needed to encode them. In the case of RVL, the prediction of the current pixel value is simply the last valid pixel, which is, in most cases, the pixel to the left.

3.1 Adaptive Span-Based Prediction

One crucial aspect was to improve the simple inter-image delta calculation of RVL to generate smaller residuals. As the employed decorrelation heuristic is not ideal, we propose to replace it with an adaptive selector of different predictor functions. In lossless compression, the transformations and functions applied in the compression stage must be reversible in the corresponding decompression stage; hence, the used prediction method must be encoded in the form of bitflags, too, because all the transformations must be reversible. Pixel-wise switching of the predictors leads to too many bitflags. Therefore the image is dynamically partitioned into spans of valid pixels in our approach. A span can be described as a one-dimensional block of a fixed number of consecutive pixel values. Invalid zero-pixels are skipped. Figure 2 shows an example partitioning of pixels into spans of length four. Using spans instead of 2D blocks reduces the computational complexity and leads to faster computation.

Our adaptive prediction then works as follows: For each valid pixel p in the span S, all possible predictor functions $\mathrm{Pred}_i(p)$ are evaluated and the one, which in total leads to the smallest absolute residuals, gets chosen for all pixels in the span to calculate the final residuals r_p:

$$r_p = \mathrm{Pred}_k(p) \tag{1}$$

where

$$k = \underset{i\in[0,3]}{\operatorname{argmin}} \left\{ \sum_{p\in \mathrm{valid}(S)} |\mathrm{Pred}_i(p)| \right\} \tag{2}$$

A high-level overview of our prediction can be seen in Algorithm 1.

Algorithm 1 Span-Based Prediction

```
Require: inputImage
  initialize data structures
  for each non-zero pixel do
      calculate possible predictor values
      increment corresp. accumulated span errors
      if span is full then
          choose best predictor in span
          save predicorID and corresp. pixel deltas
          reset span errors
      end if
  end for
```

We decided to use four different predictor functions, which then can be represented by exactly two bits per span. In principle, the actual predictors, as well as their quantity, are easily exchangeable, focusing either on computationally simpler and, therefore, faster ones or more complex and effective ones. To predict a pixel p, we opted to use RVL's standard predictor, given in Equation 3, as default case, as it is similar to the common "left pixel" approach, but handles the occurrences of intermixed zero-pixels very well by skipping them. In Fig. 3, this process is illustrated.

Additionally, we use predictors based on the delta to the upper pixel (Eq. 4), the average of the left and upper pixel (Eq. 5), and lastly, the result of a combination of the left, upper and upper left pixels (Eq. 6).

$$\mathrm{Pred}_0(p) = p_X - p_A \tag{3}$$

$$\mathrm{Pred}_1(p) = p_X - p_B \tag{4}$$

$$\mathrm{Pred}_2(p) = p_X - \frac{p_A + p_B}{2} \tag{5}$$

$$\mathrm{Pred}_3(p) = p_X - (p_A + p_B - p_C) \tag{6}$$

Equation 6 is used as the default case in the Paeth [RK99, pp. 159–162] as well as the MED predictors [WSS96]. We use only this part of them as our fourth predictor because it is computationally less expensive than using the complete Paeth or MED predictor. Tests

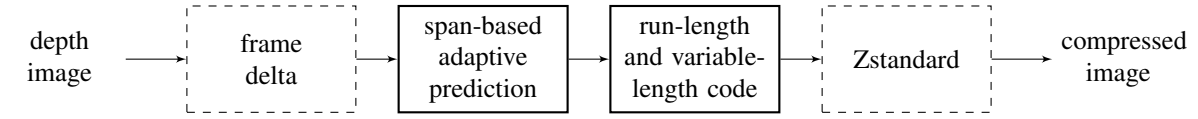

Figure 1: The pipeline of our approach. Stages in dashed lines are optional. They allow a better balance between compression ratio and compression speed. First stage: frame delta, second: our span-based adaptive intra-image prediction, third: RVL's run-length and variable-length coding, fourth: an additional Zstandard [Col16] pass.

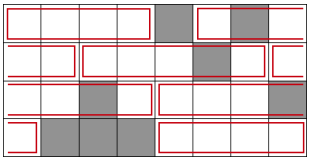

Figure 2: The pixel grid segmented in spans of four valid pixels (red boxes). Invalid pixels (grey) are skipped.

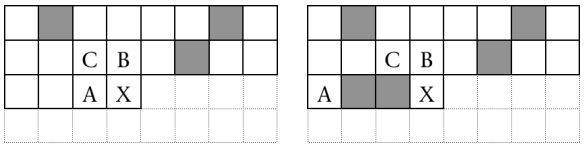

Figure 3: The prediction for pixel p depends on the pixel itself (X) and its neighbors last (A), above (B), and above left (C). Contrary to B and C, A is the last valid pixel and, therefore, not fixed, as the right image depicts.

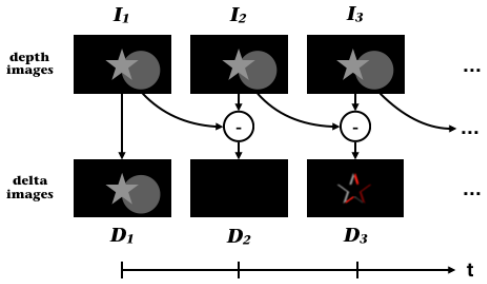

Figure 4: Frame delta computation: Sequence of images I_i and the corresponding differential images D_i between every two consecutive ones.

we conducted showed that using the full MED predictor is much more time consuming, but does not improve the compression ratio significantly.

In this way, the number of additional bits needed for the predictor representation can be significantly reduced, while retaining the ability to adapt to the local image characteristics dynamically. The exact number of predictor bits for the image can be computed as

$$x = \left\lceil \frac{(n-z)}{s} \right\rceil \cdot \lceil \log_2(f) \rceil \tag{7}$$

where n denotes the number of pixels in the image, z the number of zero-pixels, s the span size and f the number of predictor functions.

3.2 Inter-Frame Delta Computation

Another aspect we concentrated on is adding a frame delta component as a first step in the algorithm's pipeline. This is a commonly used technique in video compression, where differentials between subsequent images are encoded instead of individual images one by one. Figure 4 illustrates the process. The effectiveness depends on the application scenario, the content, and how dynamic it is. At least in cases where the change between the images is small, or only some dynamic elements occur, this technique should be beneficial for the compression ratio and speed. In the original RVL paper [Wil17], it is mentioned that such a frame delta calculation was experimented with but the compression was even worse. According to the paper, the test scene was a dynamic scene in which the camera constantly moved, which would be the worst-case scenario for this kind of technique. It is not clear if other scenarios were tested. For our pipeline, we designed the frame delta computation as an optional first stage so that it can be skipped in scenarios where it is not effective. Another aspect to consider is the inherent noise of the sensor. Some pixels switch between being valid and invalid, and in addition, the depth readings continuously vary, even for physically static objects. This results in a decreased effectiveness of frame delta computation but can be compensated for by temporal filtering as a preprocessing step (at least to some degree). Depending on how intelligent and vigorous the employed filter is, other artifacts may be introduced, which could be tolerated depending on the application.

3.3 Further bit reduction

The coding of the residuals in RVL is done by variable bit length, where each valid pixel is at least one nibble (four bits) long. For each nibble, the first bit functions as a continuation bit and the other three bits are used to represent (a part of) the actual value of the residual. Therefore, in a single nibble, a residual $r \in [0,7]$ can be represented. In cases where image regions are very homogeneous, and the computed residuals are frequently below this maximum value of seven, the algorithm can lead to comparatively poor compression results. We propose to couple RVL with a second encoder without such a limitation to mitigate this drawback. In our approach, the compressed output of RVL is, there-

fore, further processed by Zstandard [Col16]. As a combination of a dynamic dictionary-based component with a sliding search window and an ANS-based entropy component, Zstandard can potentially compress symbols smaller than four bits. According to empirical tests, Zstandard performed best as a backend.

3.4 Parallel Execution

To mitigate the inevitably increasing computation time by the more complex prediction, we implemented multi-threaded versions. The principle is the same for all variants: first, for each thread, all necessary data and an output buffer are initialized, then the image, represented as a one-dimensional buffer, is segmented into blocks by the number of threads. Lastly, the compressed parts are stitched together to one continuous block. Especially our improved prediction step in the compression stage should benefit from parallel execution as it is computationally the most expensive part and there are only locally very confined dependencies between the pixel computations.

4 RESULTS

All of our tests were conducted on an Intel Core i7-7800X, 16 GB of system memory, and under Windows 10 x64 Enterprise using the Visual Studio 2017 compiler. Each test was performed multiple times, and the average was taken.

We recorded sequences of depth images of six different test scenes with the Azure Kinect RGB-D camera in both of the two available modes, narrow and wide field of view (NFOV, WFOV). In NFOV, the depth is recorded in 640×576 at 30 Hz and in WFOV at 1024×1024 at 15 Hz. Each depth value is represented as a 16-bit integer (short). Four of those scenes were static, and two dynamic. In the first dynamic scene, the camera is fixed, and a person moves in front of the camera. In the second scene, the camera is handheld and moved around. For the static scenes, 30 frames were recorded, while for the dynamic ones, 120 frames are used. Additionally, we tested three single depth images from the Middlebury dataset [SP07] with very homogeneous depth values, and a dynamic scene of 600 frames from TUM [SEE+12], in which the camera (Kinect 1) is handheld, slightly shaken, and people moving, while seated.

In each test case, we omit the first recorded frame from the evaluation as we do a lot of initialization work for the algorithms here (e.g. reserving memory) which holds for the rest of the test. The measurements of all the other subsequent frames in a test are then averaged.

The Azure Kinect camera has the unique attribute that the output depth-image is rectangular, but depending on the mode, the content is only hexagonal or spherical. The corners are zero-pixels. While this is the standard output of the camera and could be representative for other cases, where significant static areas fail to get captured by an RGB-D sensor, it is a rather uncommon situation. In order to not only test the standard configuration but also more broadly comparable scenarios, we also evaluate depth maps cropped by 25 %. As a result, most of the static invalid regions are dropped.

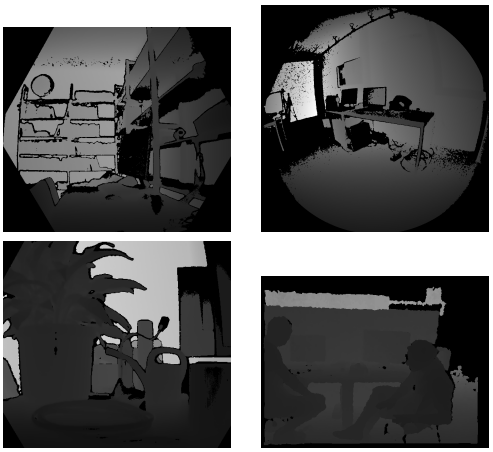

Figure 5: Example depth images of four different scenes. We record the first three with the Azure Kinect in different modes (NFOV, WFOV, NFOV with cropping). The last scene from TUM [SEE+12].

Figure 5 shows a selection of our test scenes. For an overview of all scenes and corresponding metrics, we refer to Figure 11 and Table 2 in the supplementary material.

We compare our novel compression algorithm against the original RVL algorithm [Wil17], PNG [RK99] as a classical lossless image compression algorithm that is widely used, and LZ4 as a fast representative for the LZ77 family of dictionary-based encoders. Additionally, as an example of a modern and efficient entropy coder, we decided to use an ANS implementation by F. Giesen [Gie14]. In empiric tests, this implementation performed best. Lastly, Zstandard as dictionary coder is also compared.
For our algorithm Pred, we chose a span size of 16 valid pixels. For our PredZ variant, which extends Pred by applying Zstandard for further bit reduction, we chose a span size of 16 valid pixels, and two as Zstandard compression parameter. These configurations yielded the highest compression ratios, while still achieving interactive speeds. For algorithms featuring a selectable acceleration (or compression) factor, we tested them in multiple configurations and chose the one, which minimized the difference of the compression ratio in comparison to our algorithm. In the case of PNG, a quality

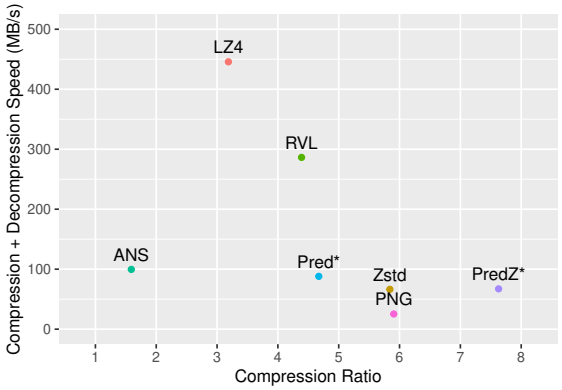

Figure 6: Average combined compression and decompression speed over compression ratio for all algorithms. Algorithms with asterisk mark our algorithms. Our PredZ algorithm achieves the highest absolute compression ratio, while still delivering real-time performance.

level of five was chosen, for LZ4, an acceleration factor of one, and for Zstandard a compression value of six.

The first test we conducted is a general comparison of all the algorithms in terms of their respective compression ratio *cr* and combined compression and decompression speed *sp*. Equations 8 and 9 describe the calculation of the metrics.

$$sp = \frac{2 \times ob}{ct + dt} \tag{8}$$

$$cr = \frac{ob}{cb} \tag{9}$$

ob stands for the original image size in megabytes, *ct* and *dt* for the compression and decompression times in seconds and *cb* for the compressed size in megabytes.

Figure 6 presents the results of this initial comparison, where for each algorithm, the thread configuration was selected, which yielded the best compression rate; therefore, the number of threads was not equal for all algorithms. For this test, the mean over all scenes and modes were taken. The asterisks identify our algorithm variants.

It can be seen that, on average, ANS is not competitive in our test, neither in speed nor compression ratio. LZ4 is the fastest but compresses rather poor. PNG has a high compression ratio of 5.9:1 but at the cost of a very slow combined speed. RVL achieves good performance and a compression ratio of nearly 4.4:1. Our span-based adaptive prediction Pred can boost the compression ratio on average by 6.5 %, but the performance decreases considerably. A regression in performance to a certain degree was to be expected by the inherently more complex delta computation and less efficient memory accesses. A possible explanation for the strength of the drop in performance, despite the rather low complexity predictors, might be the lack of optimization done compared to RVL. Interestingly, the general-purpose Zstandard algorithm can achieve an even higher compression ratio of roughly 5.8:1. However, it is also slower. Lastly, with our most sophisticated variant PredZ, we achieve the best compression ratio of more than 7.6:1, which is significantly higher than the ones of RVL or Zstandard. At the same time, the combined compression and decompression speed of 67.2 MB/s is more than sufficient to maintain interactive frame rates. For more comprehensive data of this test, we refer to Figure 12 in the supplementary material.

Predictor ID	0	1	2	3
Usage %	24.4	26.6	21.2	27.7

Table 1: Distribution of adaptively selected predictors, average over all scenes and modes.

To further analyze the effectiveness of our span-based adaptive prediction component, we counted the frequency how often each predictor is chosen as the best one per image. Table 1 shows the percentages averaged over all scenes and frames.

Each predictor is equally used, which indicates that the adaptive prediction works as intended, and all of the chosen predictors complement each other to effectively reduce the average residual in contrast to a static prediction like it is used, for example, in the original RVL algorithm.

For all subsequent tests, we only consider the most promising algorithms and omit the ones without competitive results, namely ANS, LZ4, and PNG.

In order to review our multi-threading implementation and analyze the behavior of the algorithms with increasing parallel execution, we performed all tests as well with two, four, and eight threads. Measuring initial thread creation overhead is prevented by early thread allocation. The multi-threading performance is shown in Fig. 7, in general, the combined speed rises.

As expected, the performance gains shrink with more threads as a result of diminishing returns. However, we were able to achieve considerable speedups for RVL, although it has low arithmetic complexity and therefore becomes quickly memory-bound. With four threads, the performance increased by factor 1.42. Our more complex RVL-based algorithm benefits highly from the parallel execution, as increasing the threads from one to four leads to a speedup of 2.16. While all algorithms gain performance by multi-threading, at least for up to four threads, the speedup is generally much lower than the theoretical optimum. This may be partly because not all parts of the algorithms are multi-threaded, e.g., merging the results, and that some sections of the algorithms are rather bandwidth- than compute-bound.

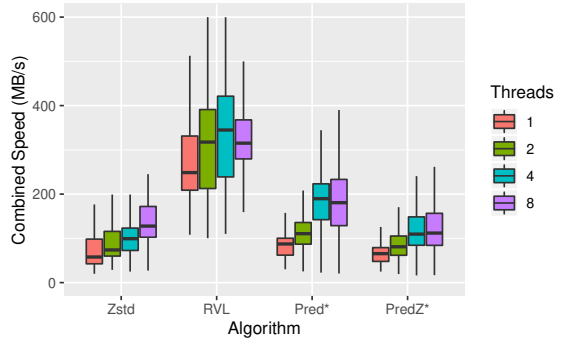

Figure 7: The image illustrates the impact of parallelization on the combined compression and decompression speed for different algorithms. All algorithms including RVL and our RVL-based ones benefit from parallel execution.

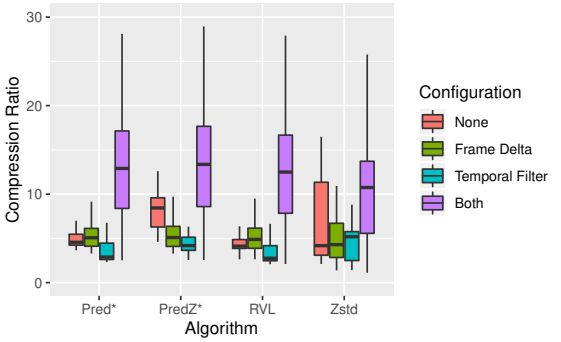

Figure 8: Comparison of the impact of our inter-frame delta computation on the compression ratio, both with and without a preceding temporal filter. The compression ratio increases hugely with the combination of temporal filter and frame delta, but using frame delta computation without the preceding filter also, most often, accomplishes notable gains.

An evaluation of the timings of the individual parts of our Pred algorithm indicates that the more computationally complex prediction stage benefits significantly more from the parallelization than the run- and variable length coding stage. Furthermore, the decompression component does not profit as much as the compression component. The compression ratio did slightly, but not considerably, decrease with more threads, as it was expected due to the separated image blocks. In the case of Zstandard, the compression ratio decreased the most with 2.68 %. RVL, on the other hand, had a decrease of less than 0.1 %. The compression ratio of our adaptive prediction dropped by 0.25 %.

To analyze the effectiveness of frame delta coding in RVL, each test is executed with and without our implementation of the addition as well as with and without a preceding temporal filter (see the last paragraph of section 3.2), which makes four variants. The filter we implemented and use is rather simple and only meant as an example. Nonetheless, we aimed at preserving the legit information (in contrast to the sensor noise) and avoiding motion artifacts. It works as follows: Pixels of an incoming image will get ignored, whose delta to the corresponding pixels of each of the last two images is below 2 %.

The combination of frame delta coding and temporal filtering increases the compression ratio for all algorithms the most, as is shown in Figure 8. While the combination of frame delta coding and temporal filtering, in general, leads to substantial improvements, our algorithm PredZ still achieves the highest compression ratio of up to 13.8:1. It should be noted though, that with temporal filtering, the compression pipeline is not strictly lossless anymore.

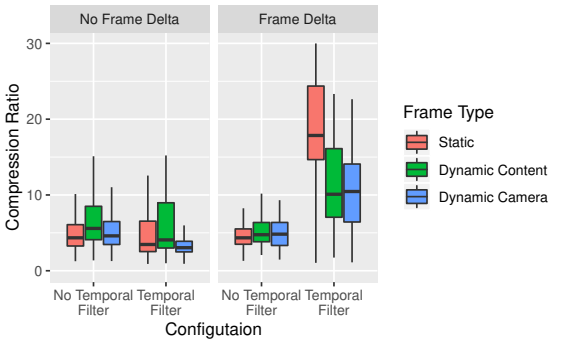

Figure 9: The image illustrates the impact of the scene type (static, dynamic content, dynamic camera) on the performance of the frame delta computation with and without filtering. Static scenes benefit the most by the combination of temporal filtering and frame delta computation.

With only the frame delta extension, the compression ratio still increases for Pred and RVL, which contradicts the statement of [Wil17]. It should be considered that for our tests we took the average of static and also highly dynamic scenes. Nonetheless, RVL's compression rate increased by 18.54 % and with our improved prediction by 11.1 %. Zstandard does not benefit from frame delta, but only from the temporal filter instead, although the confidence interval is very wide. Surprisingly, the data indicates that our PredZ performs best with the raw data compared to the others.

A detailed breakdown w.r.t. different classes of scenes and their influence on the compression ratio, while computing frame delta, can be seen in Fig. 9.

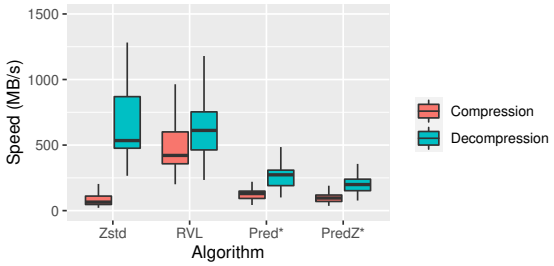

Figure 10: The image displays the individual compression- and decompression speed for different algorithms. The decompression is significantly faster for everyone.

The results indicate that the type of the scene, static or dynamic, or more specifically, the amount and the intensity of movement of the content and the camera itself, do have a significant impact on the compression rate. Without temporal filtering, the average compression rate of all algorithms is about equal. According to Fig. 8, not all algorithms benefit directly from frame delta. If, however, such a filter is used, the influence of the scene type raises strongly, specifically rather static scenes can be compressed extremely well. Interestingly, sometimes the scene which involves camera movement is better compressible using both, the temporal filter and frame delta, instead of the scene, where the camera is static and only parts of the captured environment move.

How the combined speed of the tested algorithms is distributed over compression and decompression can be seen in Fig. 10. The decompression speed generally outperforms the compression speed, especially in the case of Zstandard.

5 CONCLUSION AND FUTURE WORK

We proposed a lossless RVL-based algorithm for real-time depth image compression aimed at high compression ratios. Our experiments show that our algorithm achieves the highest compression ratios compared to competing real-time capable algorithms. With our method, depth images can be compressed up to 73 % smaller than with the original RVL algorithm and 30 % smaller than with the slower Zstandard. Using temporal filtering beforehand, we accomplish even higher compression ratios of 13.8:1, which is still the highest compared to the other evaluated algorithms but the compression pipeline becomes lossy. Thanks to parallel execution, our performance is still sufficient for typical RGBD-sensor frame rates and real-time streaming in bandwidth-limited applications. Finally, we were able to show that the original RVL can also be sped up with multi-threading, and it can, for some use-cases, benefit significantly from frame delta computation. As future work, further improvements could be made by optimizing the span-based prediction stage to lower the computation times, and maybe even SIMD can be applied. Also, it may be worth to investigate zigzag scan and to use a block-based adaptive prediction instead. The latter may exploit a possibly higher spatial coherency, however, increase the computational complexity further. Another promising idea would be to integrate the inter-image- into the intra-image delta computation by also using the neighbor pixel values from the previous frame as possible predictors.

6 REFERENCES

[BGTB12] F. Bannò, P. S. Gasparello, F. Tecchia, and M. Bergamasco. Real-time compression of depth streams through meshification and valence-based encoding. In *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, VRCAI '12, page 263–270, New York, NY, USA, 2012. Association for Computing Machinery.

[BKKF13] S. Beck, A. Kunert, A. Kulik, and B. Froehlich. Immersive group-to-group telepresence. *IEEE transactions on visualization and computer graphics*, 19:616–25, 04 2013.

[BÖK11] A. Bilgili, A. Öztürk, and M. Kurt. A general brdf representation based on tensor decomposition. *Computer Graphics Forum*, 30(8):2427–2439, December 2011.

[CJK15] C. Chen, R. Jafari, and N. Kehtarnavaz. Utd-mhad: A multimodal dataset for human action recognition utilizing a depth camera and a wearable inertial sensor. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 168–172, Sep. 2015.

[CK12] Y. Champawat and S. Kumar. Online point-cloud transmission for tele-immersion. pages 79–82, 12 2012.

[Col16] Y. Collet. Zstandard - a fast real-time compression algorithm. Github repository, 2016. `https://github.com/facebook/zstd`; accessed on 26. Februar 2020.

[Gie14] F. Giesen. Implementation of several rans variants. Github repository, 2014. `https://github.com/rygorous/ryg_rans`; accessed on 26. Februar 2020.

[HE19] H. Hamout and A. Elyousfi. Fast depth map intra coding for 3d video compression based tensor feature extraction and data analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2019.

[HKR18] S. Hemanth Kumar and K. R. Ramakrishnan. Depth compression via planar segmentation. *Multimedia Tools and Applications*, 78, 07 2018.

[JSM+14] B. Jones, R. Sodhi, M. Murdock, R. Mehra, H. Benko, A. Wilson, E. Ofek, B. Macintyre, N. Raghuvanshi, and L. Shapira. Roomalive: Magical experiences enabled by scalable, adaptive projector-camera units. 10 2014.

[KÖP13] M. Kurt, A. Öztürk, and P. Peers. A compact tucker-based factorization model for heterogeneous subsurface scattering. In S. Czanner and W. Tang, editors, *Proceedings of the 11th Theory and Practice of Computer Graphics*, TPCG '13, pages 85–92, Bath, United Kingdom, 2013. Eurographics Association.

[LBW+15] Y. Liu, S. Beck, R. Wang, J. Li, H. Xu, S. Yao, X. Tong, and B. Froehlich. Hybrid lossless-lossy compression for real-time depth-sensor streams in 3d telepresence applications. In *Advances in Multimedia Information*

Processing – PCM 2015, pages 442–452, Cham, 2015. Springer International Publishing.

[MBC17] R. Mekuria, K. Blom, and P. Cesar. Design, implementation, and evaluation of a point cloud codec for tele-immersive video. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(4):828–842, April 2017.

[MSA+13] R. Mekuria, M. Sanna, S. Asioli, E. Izquierdo, D. C. A. Bulterman, and P. Cesar. A 3d tele-immersion system based on live captured mesh geometry. In *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, page 24–35, New York, NY, USA, 2013. Association for Computing Machinery.

[MT17] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, Oct 2017.

[MZC+11] S. Mehrotra, Z. Zhang, Q. Cai, C. Zhang, and P. A. Chou. Low-complexity, near-lossless coding of depth maps from kinect-like depth cameras. In *2011 IEEE 13th International Workshop on Multimedia Signal Processing*, pages 1–6, Oct 2011.

[NIH+11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, Oct 2011.

[PKW11] F. Pece, J. Kautz, and T. Weyrich. Adapting standard video codecs for depth streaming. pages 59–66, 01 2011.

[RK99] G. Roelofs and R. Koman. *PNG: The Definitive Guide*. O'Reilly & Associates, Inc., USA, 1999.

[SEE+12] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[SMAP16] S. Shahriyar, M. Murshed, M. Ali, and M. Paul. Lossless depth map coding using binary tree based decomposition and context-based arithmetic coding. In *2016 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, July 2016.

[SOHW12] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, Dec 2012.

[SP07] D. Scharstein and C. Pal. Learning conditional random fields for stereo. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[SPB+19] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li, J. Llach, K. Mammou, R. Mekuria, O. Nakagami, E. Siahaan, A. Tabatabai, A. M. Tourapis, and V. Zakharchenko. Emerging mpeg standards for point cloud compression. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):133–148, March 2019.

[SST+18] C. Schröder, M. Sharma, J. Teuber, R. Weller, and G. Zachmann. Dyncam: A reactive multithreaded pipeline library for 3d telepresence in vr. In *Proceedings of the Virtual Reality International Conference - Laval Virtual*, VRIC '18, New York, NY, USA, 2018. Association for Computing Machinery.

[TCF16] D. Thanou, P. A. Chou, and P. Frossard. Graph-based compression of dynamic 3d point cloud sequences. *IEEE Transactions on Image Processing*, 25(4):1765–1778, April 2016.

[Wal92] G. K. Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

[Wil17] A. D. Wilson. Fast lossless depth image compression. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces*, ISS '17, page 100–105, New York, NY, USA, 2017. Association for Computing Machinery.

[WKJ+15] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald. Real-time large-scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research*, 34(4-5):598–626, 2015.

[WMS16] O. Wasenmüller, M. Meyer, and D. Stricker. Augmented reality 3d discrepancy check in industrial applications. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 125–134, Sep. 2016.

[WSBL03] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, July 2003.

[WSS96] M. J. Weinberger, G. Seroussi, and G. Sapiro. Loco-i: a low complexity, context-based, lossless image compression algorithm. In *Proceedings of Data Compression Conference - DCC '96*, pages 140–149, March 1996.

[WSS00] M. J. Weinberger, G. Seroussi, and G. Sapiro. The loco-i lossless image compression algorithm: principles and standardization into jpeg-ls. *IEEE Transactions on Image Processing*, 9(8):1309–1324, Aug 2000.

[ZCH+15] Q. Zhang, M. Chen, X. Huang, N. Li, and Y. Gan. Low-complexity depth map compression in hevc-based 3d video coding. *EURASIP Journal on Image and Video Processing*, 2015(1):2, 2015.