Analysis and comparison of the most common depth video compression techniques

João M. Alves (Machine Vision Engineer, Aivero)
Raphael Dürscheid (CTO, Aivero)

Extract:
3D cameras are becoming more widespread, but RGB-D video proves difficult to compress. We compare three lossless depth video compression techniques against Aivero's 3DQ compression.

Table of Content

Executive Summary

Since the release of the first Kinect camera for Microsoft's Xbox in 2010, researchers and industry alike have made use of the additional dimension of 3D video to simplify, improve and extend vision solutions in a wide range of industries.

Advances in compute power and the availability of public clouds, have enabled ever more powerful machine vision solutions.
However, challenges posed by 3D video such as

- novel data formats defying existing video compression,
- non-unified 3D camera interfaces and
- ever-increasing resolution and framerate are limiting the scalability of machine vision for 3D video.

This report analyses three widely used lossless (Lz4, PNG and RVL) and two relevant lossy compression schemes for depth data. Then we present Aivero's proprietary depth compression solution, 3DQ.

This document first provides background into 3D data, then we describe the main challenges of 3D video compression and the most commonly used approaches before presenting our solution - 3DQ. Finally, we study the performance of these solutions across open datasets and present a short discussion about the results.

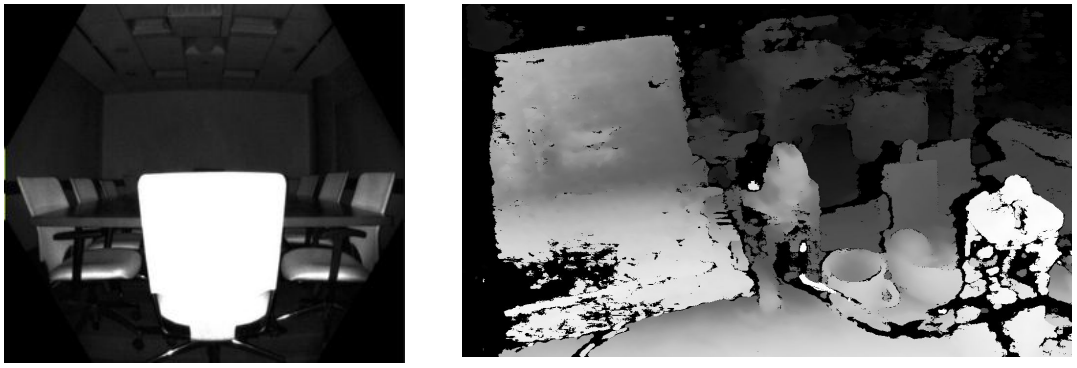
Our results show that 3DQ offers a flexible depth compression solution that can be catered to specific applications and setup, being able to achieve very high compression ratios while maintaining image quality.

Finally, Aivero's 3DQ based RGBD Toolkit for 3D video capturing, streaming, and storage is presented.

Introduction

Since the release of the first Kinect camera for Microsoft's Xbox in 2010, the use of depth data for machine vision applications has been on a steady climb. Researchers and industry alike made use of the additional dimension in video data to simplify, improve and extend vision solutions in industry areas ranging from gaming, robotics, smartphone user authentication, medical analysis, augmented reality, warehouse handling to autonomous navigation.

In recent years many new 3D cameras have hit the market, deploying a range of techniques to capture the depth components. The term depth imaging covers a range of technologies and approaches to measure distance such as LIDAR, stereoscopic imaging or stereo triangulation, time of flight, and structured lighting, synonyms are *range imaging* and *3D imaging*. Closely related are volumetric video and point clouds.



(a) *Azure Kinect* (b) Intel Realsense D415

Depth images produced by different depth sensors

Machine vision applications have historically been focused on analysing individual images, rather than a time series of images such as a video. Advances in computing power are now allowing to analyse individual frames at high frame rates or take real-time decisions based on changes in a time series of images.

Despite the increase in computing power, a major bottleneck of 3D machine vision is the data rate associated with RGB-D video streams of ever-increasing resolution and frame rate. Conventional 2D video compression has effectively tackled this issue and continues to improve. 3D video, however, has not sufficiently benefited from this development due to the differences in data formats, the ever-increasing list of new data formats, and simply the novelty of the technology and the market.

Most state-of-the-art depth data compression solutions entail lossless compression schemes due to the inability of conventional lossy compression methods to address the abrupt depth discontinuities usually present in these types of images, introducing compression artefacts that do not accurately portray the real-world geometry.

However, recently, some efforts have been made into using conventional lossy video encoders to compress depth data, by resorting to different colour encoding schemes and trying to apply some image processing techniques to reduce the compression artefacts issues.

This paper aims to illuminate the bandwidth bottleneck associated with depth image compression and transport, study the commonly used technologies for compression of depth video, based on multiple defined metrics, and present Aivero's proprietary depth image lossy compression codec, 3DQ.

In addition, this document presents Aivero's 3D video streaming toolkit that enables anyone to quickly have access to storage, transport, and machine vision solutions for multiple connected 3D cameras.

Background

In this section, we provide some background into 3D data acquisition and representation, main compression challenges and commonly used compression techniques.

3D Data acquisition and representation

A depth or 3D camera measures the distance between the camera and the subject and captures this information as an image with a given resolution in x and y. In RGB-D cameras, the 3rd dimension is often referred to as the z-axis, pointing out of the camera objective towards the subject.

Different depth cameras capture depth using different techniques. A complete explanation of the approaches is beyond the scope of this paper. Widely available cameras use the following technology:

- Kinect v1 to Azure Kinect use Time of Flight or TOF
- Intel RealSense L515 uses a LIDAR
- Intel RealSense D400 series, Stereolabs' ZED use stereo triangulation
- Zivid uses structured lighting

TOF and LIDAR measure the time taken for a light signal from the camera to reach the subject and return to the camera. Based on the speed of light the distance to the subject is calculated.

Stereo triangulation relies on two cameras, mounted a known distance from each other, that capture the same subject. By finding the same feature in both the left and right images one can triangulate the distance to the object.

Structured lighting projects a known pattern onto the subject and observes the distortion of the pattern to calculate the distance and surface orientation of the subject.

Depth cameras usually provide both a depth stream, as well as a visual stream such as colour or near-infrared (RGB-D). Access to these sets of frames is most often done through

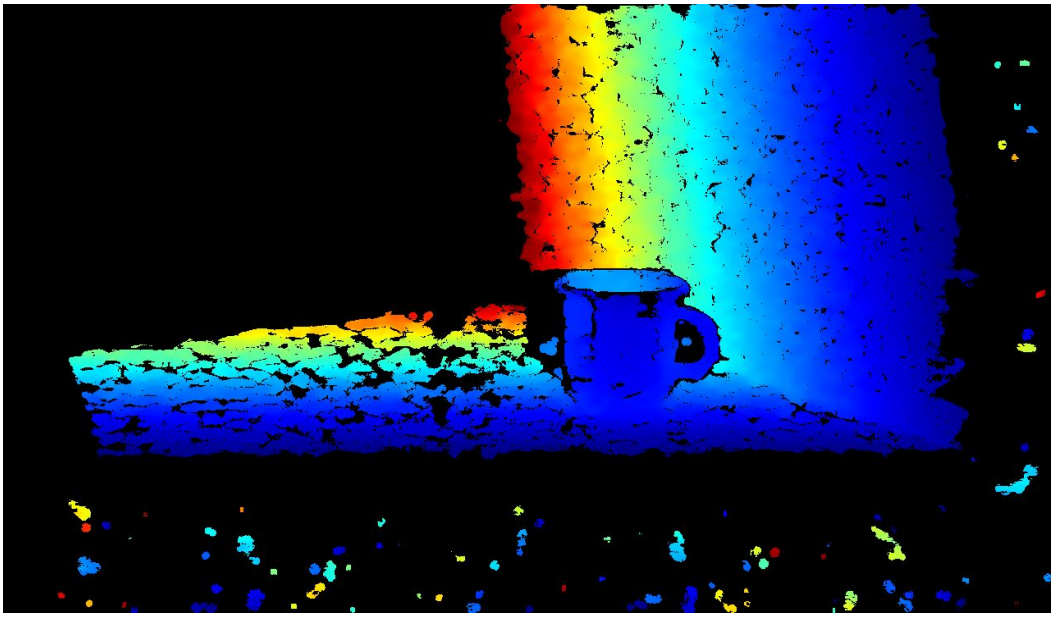
a camera-specific SDK, which makes it a lot of work to interface with different cameras. The SDKs each provide a different format for storing and loading recordings from their cameras.

- Intel RealSense uses the .bag format adapted from the Robot Operating System and applies LZ4 compression on this.
- Azure Kinect uses the Matroska container format. They compress the colour stream only, with M-JPEG.
- Stereolabs / ZED uses a proprietary format called SVO, which appears to contain left and right camera video streams, as well as metadata.

What all 3D cameras and their SDKs have in common is that they will give the user access to a **depth map**. A depth map is a 2D image where every single pixel represents the distance from the camera to the point on the object that is being imaged.

A depth map most often encodes the depth data as Z16 or *unsigned int of 16 bit*.
This format uses a single channel, 16-bit value to encode the distance from the camera to object for every pixel. The data can be visualised as a grayscale image where near to far pixels are shown from black to white, respectively.

To make it easier for humans to understand the image, the depth map is often coloured in using a *rainbow style* progression of colours. We call this a **colourised depth map**. Each step in value represents a fixed distance in the real world. The unit of this step is depending on the camera and needs to be negotiated separately. By default, the Intel RealSense cameras, for example, use 1mm as the unit. With 16-bits this allows for representing distances up to 65535mm or ~ 65.5m away, with value 0 being an invalid pixel. However, the actual maximum distance of depth data recovered greatly depends on the camera, the technology being used, and the overall scene texture.

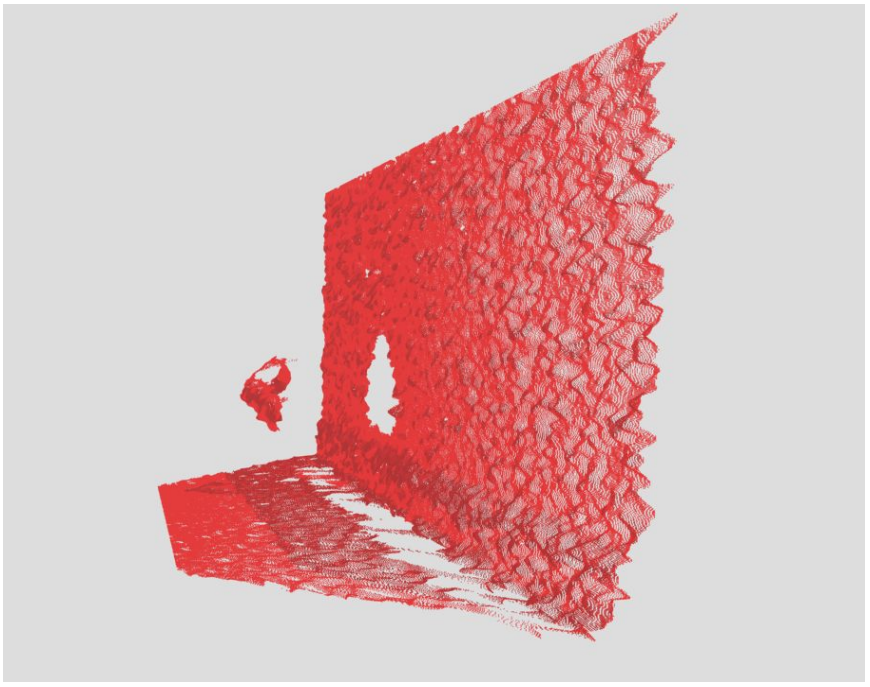


Colourised depth map using a JET colourisation showing the Aivero mug.

Many 3D cameras will additionally capture near-infrared or colour data and often provide additional metadata to allow mapping colour data onto the depth map. This combination of colour and depth data is often called **RGB-D**.

Colour image of the Aivero mug.

Virtually all 3D cameras will produce a depth map before converting the data to point cloud. A point cloud is an unordered list of occupied space in 3D, sometimes with attached colour information. It can be computed from a depth map by projecting the pixels into a 3D volume using the intrinsics of the camera used for capturing the data. Basically, by reversing the path the light travelled from the object to the sensor of the camera, we can compute the location in the real world that is represented by the pixel - the point. Doing this for every point in a depth map results in a list of points, called a **point cloud**.



Rendering of a point cloud showing a floating Aivero mug.

Since point clouds contain unstructured data that is localised in 3D, one can easily combine data from multiple cameras and render it from different viewpoints. Given that the geometric

relationship between the different cameras is known, the points produced by the individual cameras can be plotted in a common frame of reference.

Cameras operating in the **visible light** (colour) usually record 3 channels in 8 bit: R, G, B for red, green, and blue. Each channel contains values that can be between 0 and 255. How the mapping of colours to the real world is done is out of scope for this paper, but you can read up about the Gamut for that. With the availability of “HDR” displays more and more content is available that uses 10-bit or even 12-bit per channel and can represent values up to 1023 and 4096, respectively. This makes the colours `pop` and appear more realistic.

State of the art in RGB-D cameras records depth data at 1280 by 720p. The colour stream, however, is already being captured with 4k: 3840 by 2160 pixels, which are 9 times more pixels. With 8k on the horizon, in the future compression will be even more relevant than today.

Raw Bandwidth Study

Capturing raw uncompressed images produces, both for the 2D and 3D cases, a great amount of data. Due to this limitation, the RAW format is not usually used for data transport.

The following table contains some example formats, namely RGB colour, GRAY16 depth and point cloud according to the pcd format, that should help to gain some insight into the bandwidth cost of the RAW format. You can check the calculations for these values appended to this document.

	Typical representation	Bytes-per-pixel	Size per frame (1280*720p)	Bandwidth at 30 FPS
Colour (RGB)	3 channels @ 8-bit	3	2.8 MB	664 Mbit/s
Depth (GRAY16)	1 channel @ 16-bit	2	1.8 MB	442 Mbit/s
Point Cloud (pcd)	4 channels at 32-bit	16	14.7 MB	3539 Mbit/s

Raw data sizes using different formats.

Challenges of 3D video compression

Luckily, there is a plenitude of codecs available to compress video data. Widely used are VP8, VP9, H264/AVC, and H265/HEVC. These codecs are hardware accelerated on virtually every CPU or GPU, which makes them fast and energy-efficient. The codecs differ between implementations and hardware that runs them, but usually, they operate on 8-bit data, while some also support 10-bit or 12-bit. The exact inner workings of these codecs are out of scope for this paper.

Generally, these encoders will group parts of the image containing very similar pixel values and furthermore encode only the differences between a keyframe and subsequent frames to reduce bandwidth. When compressing depth images this amounts to some very noticeable artefacts that are known as flying pixels, especially around object edges, where invalid pixels (of pixel value 0) usually appear. These artefacts can be somewhat dealt with using some simple image processing techniques, either by simply removing them or trying to correct their value.

On top of this, the codecs are optimised for human perception and generally try to reduce the image quality in places where humans won't notice it. The encoder implementations do not directly operate on RGB video data but convert it to YUV. YUV is another pixel representation dating from the analogue TV times where both black and white TVs needed to be supported, as well as colour TVs. Instead of representing the colour as its components in Red, Green, and Blue, it represents colour as an intensity component Y and two chroma components U and V. The intensity alone allows for showing a grayscale image. The chroma components are a differential encoding of the colour components.

When converting an RGB image to YUV the intensity value (Y) is computed as a weighted sum of R, G, B and U, V are calculated based on those weights, too.

$$Y = 0.299R + 0.587G + 0.114B$$
$$U \approx 0.492(B - Y)$$
$$V \approx 0.877(R - Y)$$

From the equations above, one can see that this conversion favours the green components of the image. This is due to the fact that human visual perception is more sensitive to green. Additionally, YUV data is usually downsampled such that there is a Y component for every RGB pixel of the input image, but only a U and V component for every 2x2 pixel region of the input image.

The complete background on colour models is out of scope for this paper.

Depth image compression techniques

State-of-the-art depth data compression solutions usually entail lossless compression schemes due to the inability of conventional lossy compression methods to accurately preserve real-world geometry when applied directly to depth data. However, recently some work has been done into lossy depth data compression. In this section, we introduce the most commonly used techniques.

Lz4 is a generic lossless compression algorithm focussed on very fast compression and decompression on CPU. Intel RealSense uses the *lz4* compression to reduce the filesize in the .bag files produced by their capturing software. *Lz4* operates on a frame-by-frame basis, meaning it does not use temporal cohesion/inter-frame compression. Its typical compression ratio in this context ranges from 2x to 5x.

PNG is a widely used lossless compression method targeted at image compression, including 16-bit grayscale images. *PNG* does not use temporal cohesion/inter-frame

compression, boasting a typical compression ratio slightly better than *lz4*, but much slower compression times. Typical compression ratios for *PNG* range from 2x to 9x.

RVL is a lossless 16-bit depth image compression method developed by Microsoft that can achieve compression ratios similar to PNG, while being much faster both in compression and decompression stages. However, RVL makes assumptions about the original data that, if not met (rarely the case), may cause the method to increase the data size instead of reducing it, achieving compression ratios lower than 1. In the worst-case scenario, the compressed data size is 1.5 times larger than the raw data size. RVL does not use temporal cohesion/inter-frame compression and, in most cases, RVL achieves compression ratios similar to PNG. Typical compression ratios range from 3x to 10x.

The direct mapping of 16-bit depth data to RGB, followed by compression with H265 (*RGB+H265*) can be achieved by mapping the first 8-bits of the depth data into the R channel, the second 8-bits into the G channel, and leaving the B channel untouched.
As shown above, when considering a 16-bit depth image we cannot convert it to RGB and then YUV for further compression without loss of data. This is despite the fact that RGB consists of three 8-bit fields - 24-bits in total and appears to `fit` into that data since feeding this data to any encoder, causes information to be lost when transcoding into YUV. Furthermore, information is also lost when an encoder compresses the data with human perception as the target consumer.

A recently published whitepaper suggests the use of the HUE colour progression in order to convert depth data to RGB, allowing the use of classical image/video encoders such as webP (*HUE+WebP*). Compression ratios are reported to be high while maintaining medium image quality. Based on the available information we tried to reproduce the results, but fell well short of the reported image quality on our open data sets. A range of inconsistencies and lack of parameters as well lack of the reference dataset proved challenging. Nonetheless, we include our results and will continue the investigation for future iterations of this paper. The particular choice of the HUE colour space imposes a limit of 1530 unique values that can be encoded. For datasets where the relevant information is contained within a depth value interval smaller or equal to 1530, that interval is defined as the dataset's region-of-interest (ROI). In the case of depth data ranges larger than 1530 values, the data is to be discretized to be encoded. The canonical webP compression implementation prevents encoding in real-time even on powerful i9 processors unless quality is sacrificed.

3DQ Framework

3DQ is an RGB-D capture, storage and streaming solution based around a proprietary, lossy compression algorithm for 16-bit depth data, developed by Aivero AS. It handles both RGB and Depth data, as many applications require both in parallel.

It utilises conventional hardware acceleration featured in Intel CPU and Nvidia GPU, allowing for easy access and widespread availability.

Inter-frame/temporal cohesion is used, allowing for compression ratios of up to 20x and quality ranging from acceptable to near-lossless. It is computationally efficient and allows for high frame rates (30+FPS) on most machines, being especially well suited for edge-like devices streaming RGB-D data to a central, more powerful machine.

Aivero's Deepcore. An edge device tailored for capturing and streaming of 3D data using the *3DQ based toolkit.*

3DQ operates on an ROI that currently supports up to 3072 depth units, or 3.072 meters when using a 1 mm/step resolution.

Asides from compression, 3DQ interfaces with common 3D cameras such as the Microsoft Azure Kinect and Intel RealSense devices. It provides video transport across networks, as well as storage of depth video data.
It can be extended using an open RGB-D interface to provide support for new data sources.

Experiments

In this section, we describe the overall experimental process, including the setup, used datasets and experiments.

Setup

In order to assess how 3DQ fares against other available compression techniques, a series of experiments were run across multiple datasets utilising both lossless and lossy compression algorithms as introduced above. The chosen lossless methods consisted of PNG, RVL and LZ4, while the lossy methods chosen were HUE+webP, RGB+H265 and 3DQ.

Each algorithm's compression performance was evaluated by compressing and decompressing each dataset using the various compression methods and, for the lossy ones, comparing the raw original dataset frames with the frames obtained after applying the different compression and decompression methods.

ROI settings were not varied within a dataset but varied across the data sets.

The lossless encoding methods allowed for no configuration with the exception of PNG. Here the fast compression level 1 and the best quality compression level 9 was tested.

The lossy methods allowed for more parameters:

In terms of target bitrates, both 3DQ and RGB+H265 were exercised with 5000, 10000, 15000, 20000, 30000, 40000 and 102400kbit/s.

The equivalent setting for HUE+webP was the webP encoder's *quality* parameter, which was exercised with values 100,90,70,50,30,10,0. WebP furthermore allows for specifying a speed/quality tradeoff. This was left at the default value of 4.

The vaapih265enc encoder of the RGB+H265 encoding was used in *variable bitrate mode* with the default *quality preset* of value 4.

We use the peak-signal-to-noise-ratio (PSNR) between raw frames and the encoded/decoded frames to quantify the quality loss of the lossy compression methods. The compression ratio was measured to evaluate the compression performance, and the framerate obtained during compression and decompression to evaluate overall throughput. These performance metrics are explained further on in this document.

All experiments were run on an Intel i9-8950HQ CPU and 32GB of RAM.

Datasets

The used datasets consisted of multiple typical 3D scenes that span across different geometric complexities, noise conditions and overall dynamic motion of the scene. Most were captured using Intel RealSense cameras, but we also present results for depth video captured using the Azure Kinect camera from Microsoft.

Technical details regarding the used datasets can be found in an Appendix to this whitepaper.

We considered 4 datasets dubbed:

- *floating_mug*
- *fishy-fish*
- *plant*
- *OFFICE_WFOV_UNBINNED*

Both floating_mug and *fishy-fish* have been captured using an Intel RealSense D435 camera with the default settings using the official RealSense-viewer tool provided by Intel, meaning 1280x720 of image resolution with 1mm of depth step resolution at 30 FPS.

The *floating_mug* dataset shows a metallic mug moving in front of a white background, being relatively simple in terms of geometric scene complexity and representing a typical example of a somewhat dynamic 3D scene, with a static background and moving foreground. This dataset's ROI is 250 to 700 depth steps (250 to 700 mm) and most scene elements are within the working range of the camera, so there is a high percentage of valid data in this dataset.

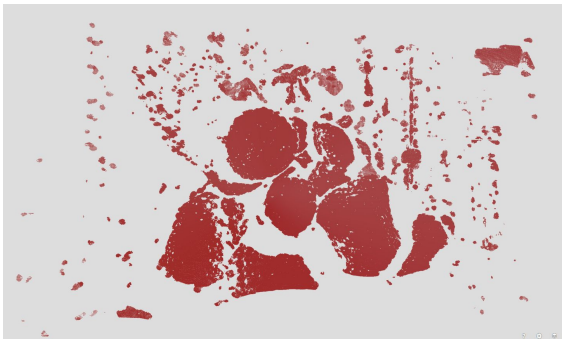
Fishy-fish shows a wooden segmented fish in front of a developer desktop setup, being more complex in terms of geometric scene complexity while being more dynamic than *floating_mug* due to the presence of more shadows in the scene, amounting to less valid pixel data. In this particular data set, the relevant information is spread out from 250 to 1600 depth steps (250 to 1600 mm) meaning it has a bigger ROI, and more interesting data to encode than the other datasets.

The *plant* dataset has also been captured using a D435 but uses a custom configuration to reduce the minimum distance from the camera at which valid depth data is recorded. Furthermore represents the depth data in steps of 0.5mm, rather than the typical 1mm. The *plant* dataset contains a rotating Chinese Money Plant, and is a very complex geometric scene, with many reflecting surfaces in the background that increase the overall noise, as well as normal foreground dynamics containing lots of shadows. This all amounts to a very dynamic and challenging dataset for the different compression techniques. It's ROI is from 300 to 700 depth steps (150 to 350 mm due to the 0.5mm resolution).

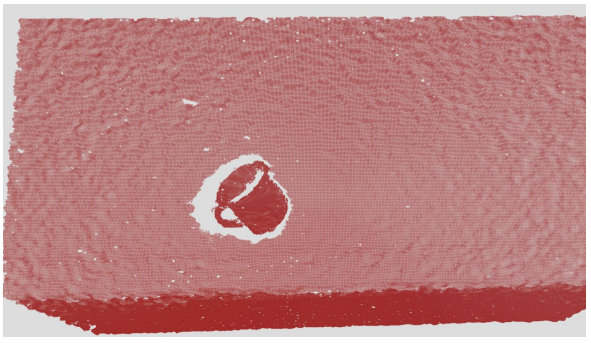
The final dataset called *OFFICE_WFOV_UNBINNED* is one of the 4 official data samples released by the Microsoft Azure Kinect team. It is captured in 1024 by 1024 pixels at 15 fps. It portrays a static scene of a board room, with an ROI of 1000 to 2530 depth steps (1000 to 2530 mm), being a lot less challenging than the previously described datasets, with no

dynamic environment whatsoever, other than the noise inherent to the camera, resulting in a very high percentage of valid pixel data.

Point cloud renderings of a single frame of uncompressed data for each dataset
- click to view 3D models -



plant dataset
Truncated to 300 and 700, 0.5mm/step



floating_mug dataset
Truncated to 300 and 700, 1mm/step

fishy-fish dataset
Truncated to 250 and 1600, 1mm/step

OFFICE_WFOV_UNBINNED dataset
Truncated to 1000 and 2530, 1mm/step

Performance metrics

PSNR (peak-signal-to-noise-ratio) was used as the main accuracy performance metric as it uses the mean squared error between the original data and the data after compression/decompression, to quantify the loss of information between the two.

PSNR is defined as follows.

$$PSNR = 10log_{10}(\frac{MAX_I^2}{MSE})$$

Where MAX_I represents the maximum possible value allowed in the pixel representation. In this case, since depth images are 16-bit, it is defined as below.

$$MAX_I = 2^{16} - 1 = 65535$$

MSE represents the mean squared error between the two images being compared. It is defined as follows.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - I'(i,j)]^2$$

Where I is the original image, I' is the compressed/decompressed data, while m and n are the image dimensions.

A high PSNR value represents a low loss of information, and more accurate compression, while a low PSNR value represents a high amount of lost information and less accurate compression. Of course, this only applies to lossy compression techniques used since lossless compression techniques would always yield an undefined PSNR value since the mean squared error between the images would be zero. To put PSNR values into perspective, a 16-bit image with 1280x720 dimensions, with a single-pixel changed by 1 unit would yield a PSNR of 156 dB. Typical values for transcoded 16-bit images are 60-80 dB PSNR.

Compression ratio was used as a performance metric for both lossy and lossless compression, as it illustrates the reduction in data size achieved by each compression algorithm. We defined compression ratio as follows.

$$Compression\ Ratio = \frac{Original\ data\ size}{Compressed\ data\ size}$$

Where $Original\ data\ size$ is the sum of the size of every raw depth frame in the dataset and $Compressed\ data\ size$ is the sum of the sizes of the compressed depth frames.

Framerate after compression and decompression was used as a way to measure the data throughput offered by each compression method since it provides a strong indicator for that method's computational complexity. For the sake of clarity, we only measure FPS when evaluating the datasets that were recorded at 30 FPS (all except OFFICE_WFOV_UNBINNED).

Results

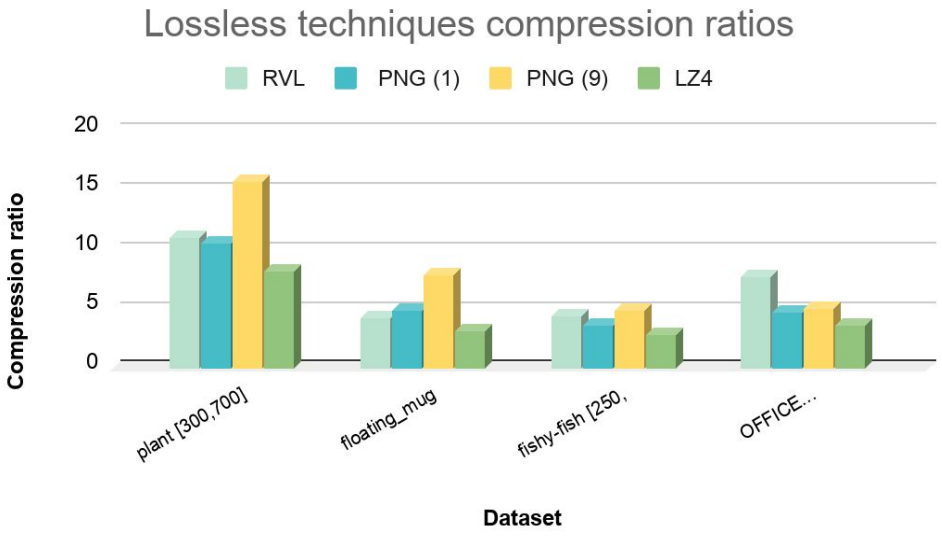
We compare the compression ratios achieved across the ROI truncated datasets for the lossless compression approaches and Aivero's 3DQ. We show example point clouds representing the different lossy compression results.

Lossless compression techniques

We evaluate the following lossless compression methods across the datasets:

- RVL
- PNG (compression level 1)
- PNG (compression level 9)
- Lz4

Compression ratio



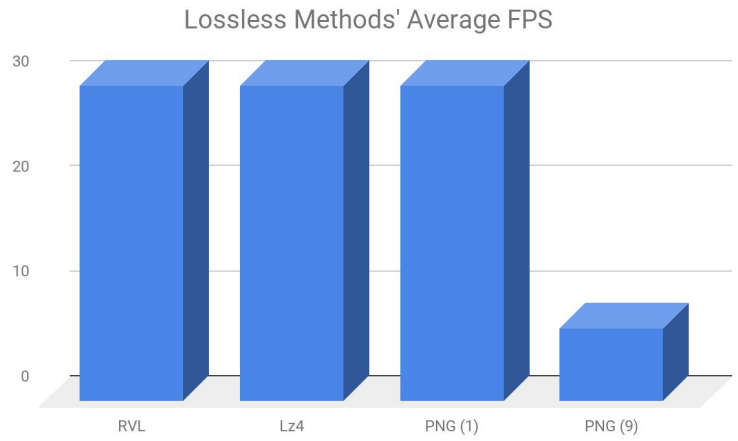
From the graph above, one can verify that, with the exception of the *plant* dataset, the compression ratios of the lossless compression techniques are clustered around a compression ratio of 5x. The tested compression methods managed to compress the *plant* dataset better than the rest. In this dataset, compression ratios ranged from around 8x to 15x.

For all datasets, PNG at compression level 9 achieved the best compression ratio here with around ranging from 5x to 15x depending on the dataset.

PNG compression level 1 and RVL achieved similar compression ratios (4x-5x) for all datasets but the Azure Kinect dataset, where RVL achieved a greater compression ratio (8x).

Lz4 achieves the lowest compression ratio in all datasets.

Framerate



The above graph tells us that RVL, LZ4 and PNG(1) managed to compress and decompress the data in realtime achieving the native 30 FPS of the datasets.

PNG(9) only managed to get 7 FPS out of the dataset's native 30 FPS, which makes it unsuitable for real-time applications.

Lossy compression techniques

We evaluate the following lossy compression methods on the same datasets:

- 3DQ
- HUE+webP
- RGB+H265

Note that a PSNR of a 70dB was considered the absolute lower cut-off when evaluating the quality as a function of bitrate for 3DQ.

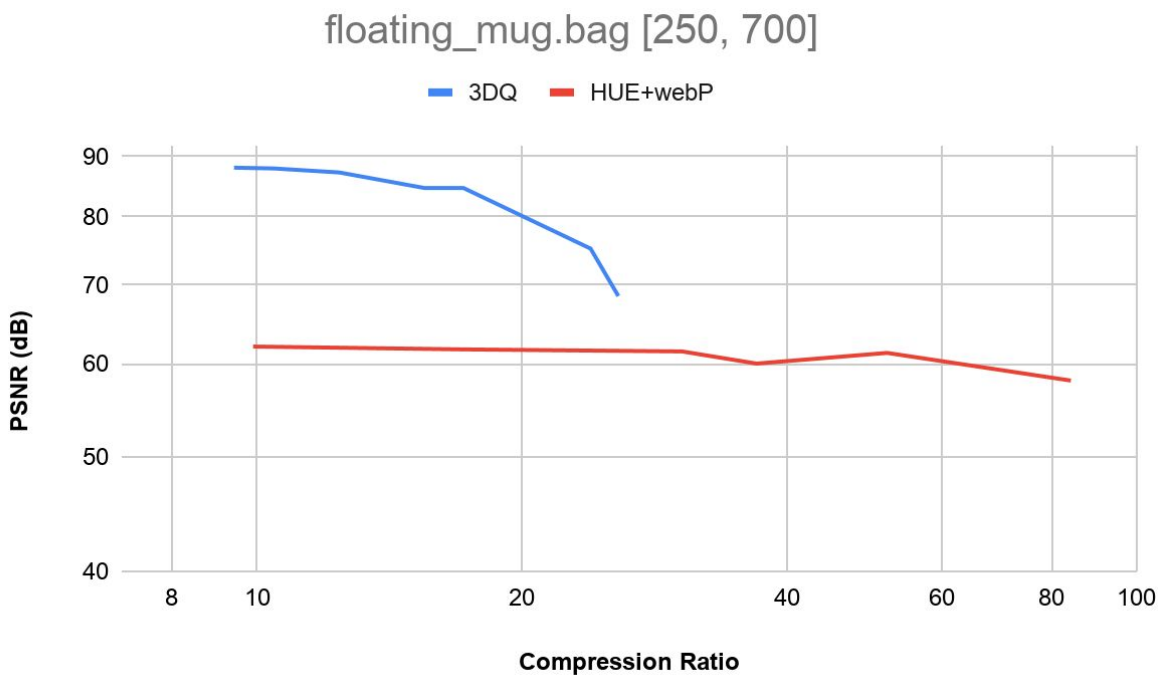
None of the colourisation approaches (HUE+webP and RGB+H265) achieved this on even their highest quality, but we are plotting their performance nonetheless.

Compression ratio

Across the data sets, 3DQ compression achieves compression ratios (c/r) between 7.5x and 17.6x while maintaining PSNRs safely above 70dB for all but the highest c/r, reaching a maximum of 89dB.

HUE + webP compression results in c/r between 7.3x and 254x, however, the highest PSNRs achieved is 62dB.

RGB + H265 compression results in extremely high c/r between 32x and 917x, at the cost of quality, producing very low PSNR results, with a maximum value of 55 dB. In order to keep the plots readable, these values have been excluded where necessary.

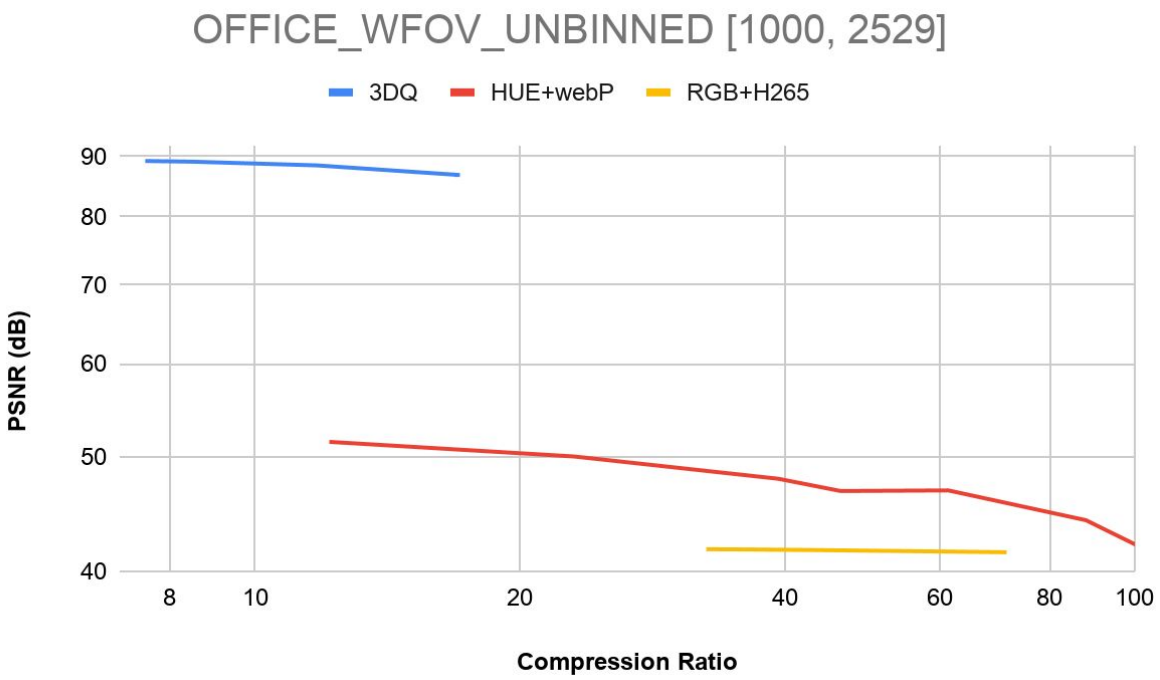


This data set has a low percentage of invalid pixels and all pixels are valid for both 3DQ and HUE+webP.

The *floating_mug* data set is being compressed with a retained quality of 88dB to 82dB PSNR for 3DQ, achieving a c/r of 9.4x to 17.2x.

HUE+webP achieves a PSNR of 63dB to 56dB, with c/r from 8.5x to 109x.

RGB+H265 achieves a PSNR of 46dB with a c/r of 388x to 917x (not shown).

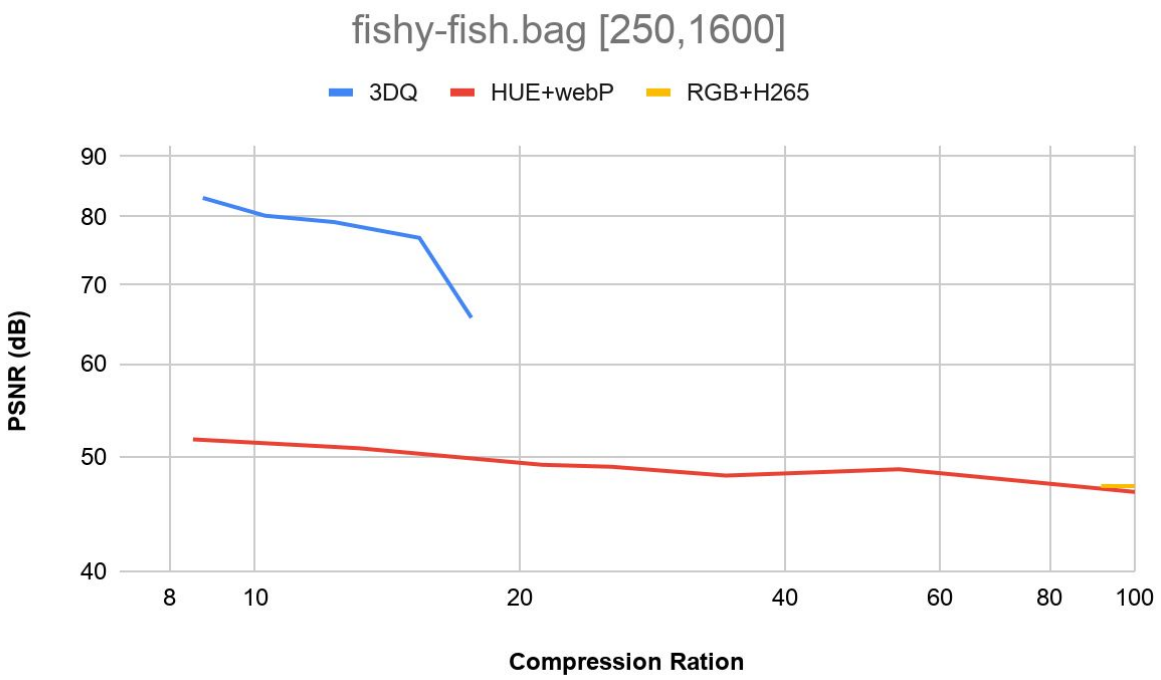


The *OFFICE_WFOV_UNBINNED* is fully static except for camera noise. We encode an ROI that is valid for 3DQ and HUE+webP without quantisation.

The *OFFICE_WFOV_UNBINNED* dataset is being compressed by 3DQ with a PSNR of 89.2dB to 86.8dB, achieving a c/r of 7.5x to 16.9x.

HUE+webP achieves a maximum PSNR of 51.5dB, which decreases to 49dB, c/r range from 12x to 149x.

RGB+H265 achieves a PSNR of ~41dB across the compression ratios of 32x to 71.5x.

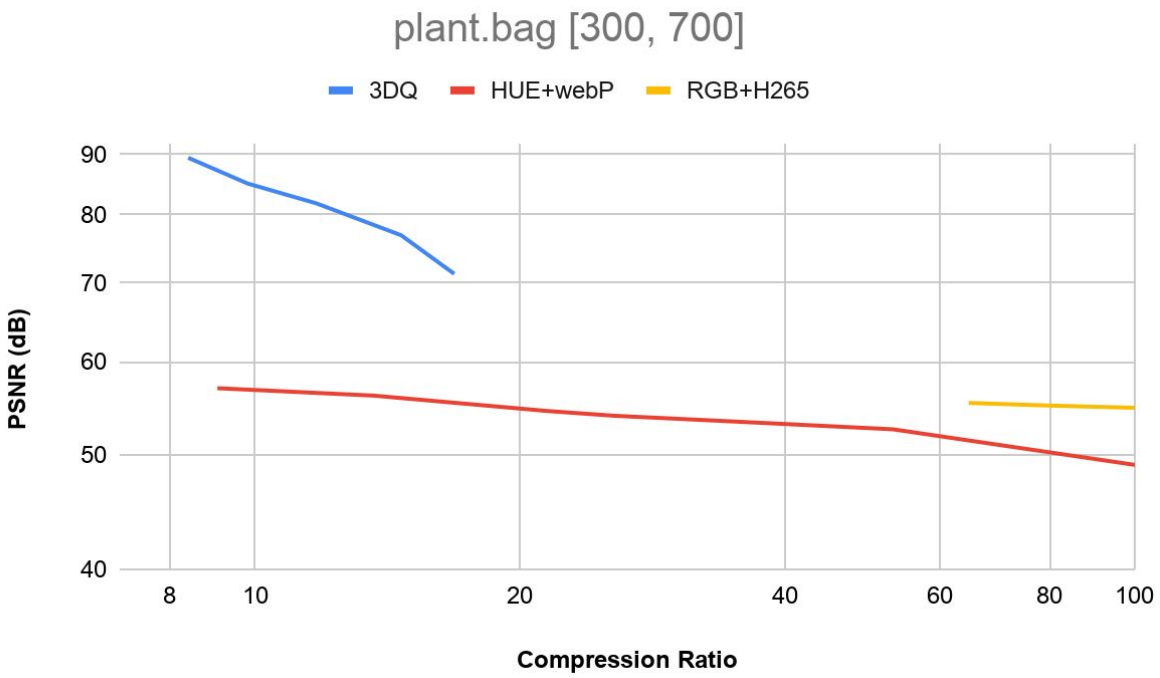


The *fishy-fish*.bag data set is dynamic and we encode an ROI that is valid for 3DQ as well as HUE+webP without quantisation.

Using 3DQ, *fishy-fish* starts at c/r 8.7x and 83dB PSNR.
At c/r of ~10x, PSNR still remains above 80dB and drops to 65.7dB at a c/r of 17.6x.

HUE+webP achieves a maximum PSNR of 51.8dB, which decreases to 46.4dB, c/r range from 8.5x to 108x.

RGB+H265 achieves a PSNR of ~47dB across the compression ratios of 106.5x to 207.7x.



The plant.bag dataset is a very dynamic data set recorded at 0.5mm depth steps. At a *near_cut* of 300 and a *far_cut* of 700, the dataset captures data at 150mm to 300mm from the camera. It has many discontinuities in the depth map.

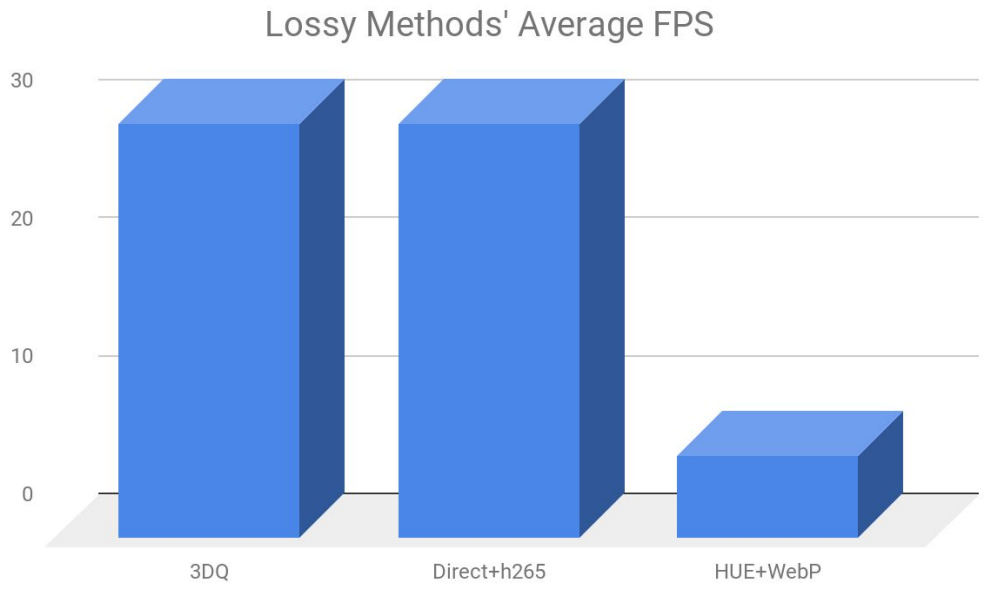
Using 3DQ, the *plant* data set starts with a c/r of 8.4x at 89.4dB PSNR, the quality decreases continuously until reaching 71.3dB PSNR at 16.8x c/r.

HUE+webP achieves a maximum PSNR of 57dB, which decreases to 49dB, c/r range from 9x to 100x.

RGB+H265 achieves a PSNR of 55.4dB, decreasing to 54.1dB across the compression ratios of 65x to 173x.

Point cloud renderings of data obtained through the 3DQ codec can be found appended to this document.

Framerate



As can be seen from the above graph 3DQ and RGB+H265 methods manage to achieve the dataset's native 30 FPS

However, HUE+WebP creates bottlenecks that limit framerate at 7 FPS. The webP implementation used was the canonical implementation released by Google. The speed preset was left at the default, 4.

Discussion

Regarding the compression ratios, lossless techniques behaved as expected, providing ratios that range from 4x-15x across the multiple datasets.

PNG(9) proved to be the most effective at compression data but proved to be the more computationally complex, since it only achieved 7 FPS after compression and decompression, while the others proved to be able to keep up to the dataset's native FPS.

As for lossy techniques, 3DQ emerged supreme in quality, achieving very high PSNR results, while HUE+WebP came in at a second place and RGB+H265 showed a very poor performance. This is likely due to the aforementioned issues with the depth data packing method it uses. Furthermore, lossy compression done by the h265 encoder may quantise the high significant bits, introducing errors and mangling the scene geometry. The compression ratio is extreme at > 900x, at a severe cost of quality.

The disparity between our results using HUE+WebP and the original authors' whitepaper could be explained by the use of more challenging datasets by us and the absence of post-processing techniques in our work. We decided to not post-process the data to evaluate the out-of-the-box performance across a range of datasets and two cameras. In addition, the

choice of webP encoding already prevented real-time encoding. Adding post-processing would furthermore reduce encoding frame rate due to the additional computations required.

3DQ, on the other hand, managed to match the dataset's FPS in all experiments, showing that 3DQ, well suited for real-time applications, unlike HUE+WebP since the latter method struggles to run even in a modern machine like the one used in our experiments.

Conclusions

In this work, we have compared commonly used depth compression techniques across a number of datasets that vary in respect to scene dynamics, depth range, noise levels and geometric complexity. We evaluated both lossless and lossy depth compression techniques and used key-metrics to draw conclusions about them.

Some conclusions can be drawn from the obtained results.

Firstly, lossless compression methods, due to carrying no information loss, are limited in terms of compression capacity as expected and shown in the previous section.

The evaluated available lossy compression methods, as expected, proved extremely greedy in terms of compression ratio, sacrificing quality for higher compression.

3DQ provided a significant reduction in data size for the majority of data sets while being computationally very efficient. It outperforms usual lossless depth compression methods RVL, PNG and LZ4 in all data sets in terms of compression ratio while being less computationally expensive than PNG(9). which was expected from a lossy compression method. However, it manages to do it while maintaining a very high image quality level as shown in the Results section above, especially when compared to other lossy compression methods that not only can not achieve the same quality level as 3DQ (HUE+WebP and RGB+h265), but also, in some cases, are much more computationally expensive (HUE+WebP).

Overall, 3DQ offers a flexible depth compression solution that can be catered to specific applications, balancing image quality and data compression to suit application requirements, it being in terms of available bandwidth, storage space, and camera setup, something that other depth compression methods lack.

3DQ framework for 3D cameras

While 3DQ is centred around the compression of RGB-D video, it is not solely a codec but a complete framework for capturing, compressing, streaming, storing and analysing depth data from one or multiple 3D cameras.

The 3DQ provides a real-time, low-latency streaming solution based on RTSP, RTP, and webRTC using the proprietary compression technology shown above. Furthermore, 3DQ allows for storing compressed RGB-D video streams to disk. Compressed data can be accessed in C++, Rust, Python, MATLAB, NVIDIA Deepstream SDK, Samsung NNStreamer, GStreamer, and with common deep learning tools such as Tensorflow and MXnet.

3DQ provides the tools for centralising the storage of any RGB-D video source.

Common, open interface for capturing RGB-D data

The 3DQ RGB-D compression, transport, and storage solution are camera agnostic, and new cameras can be added using an open-source interface. This interface is part of a set of open-source libraries developed by Aivero, allowing it to handle RGB-D video data in the widespread GStreamer media framework.

The Intel RealSense series, as well as the Azure Kinect, are currently supported. Any developer can integrate new cameras. Aivero also provides this service on demand.

Using 3DQ it is possible to deploy networked depth-based vision running on a central server for applications including; surveillance, autonomy, robotics and within security.

Read more about our 3DQ depth vision pipeline: www.aivero.com

Licensing and support

The Aivero 3DQ software is available for purchase and we offer a 30 days free trial.

Write us directly at sales@aivero.com or reach out to us via www.aivero.com

Acknowledgements

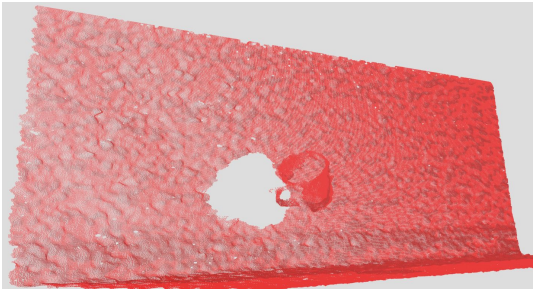
We would like to thank our dear dev team colleagues Tobias Morell, Niclas Overby, Andrej Orsula, Kasper Steensig, and Vojtech Jindra for their great work and dedication to 3DQ and providing feedback and insights. In addition Christian Rokseth and Martin Svangtun spent a lot of time helping to get this paper out.

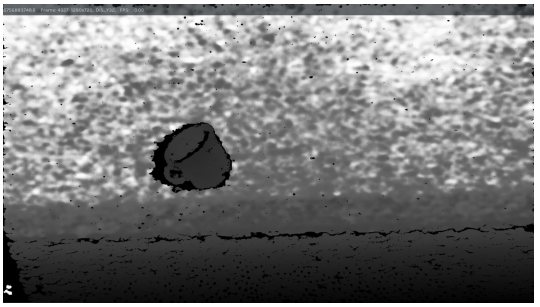
Appendix

Datasets

floating_mug

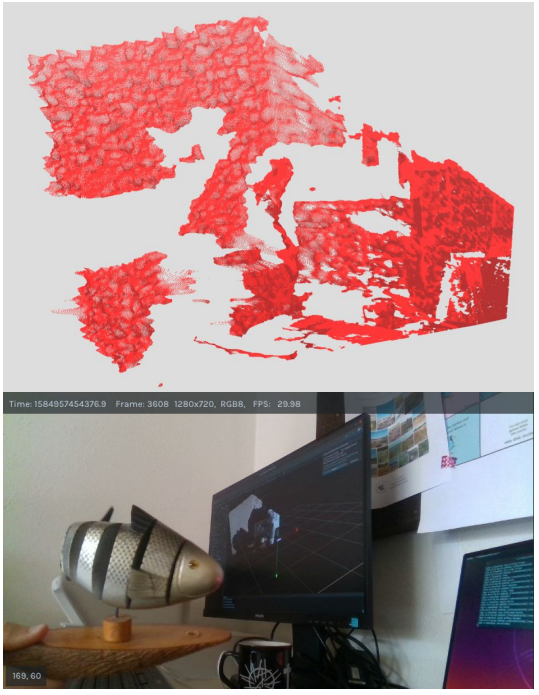
- Format: rosbag (.bag)
- Camera: Intel RealSense D435
- Intel RealSense configuration: Default
- RealSense-viewer finds incomplete video: yes
- Streams:
    - Depth
- Data Range
    - nearest: 259mm
    - furthest: ~667mm
    - step size: 1mm/step
    - range: 408 steps
- Invalid data due to hw constraints (too close): no
- Length: 8s
- Rosbag total size (all streams): 146.8MB
- Number of depth frames (from rosbag info): 245
- FPS: 30
- Resolution: 1280*720
- Dataset Download: https://bit.ly/3f9BrfE





fishy-fish

- Format: rosbag (.bag)
- Camera: Intel RealSense D435
- Intel RealSense configuration: Default
- RealSense-viewer finds incomplete video: no
- Streams:
    - Depth
    - Colour
- Data Range
    - nearest: 254mm
    - furthest: ~1568mm
    - step size: 1mm/step
    - range: 1284 steps
- Invalid data due to hw constraints (too close): yes
- Length: 24s
- Rosbag total size (all streams): 1.9GB
- Number of depth frames (from rosbag info): 731
- FPS: 30
- Resolution: 1280*720
- Dataset Download: https://bit.ly/3f8VATp

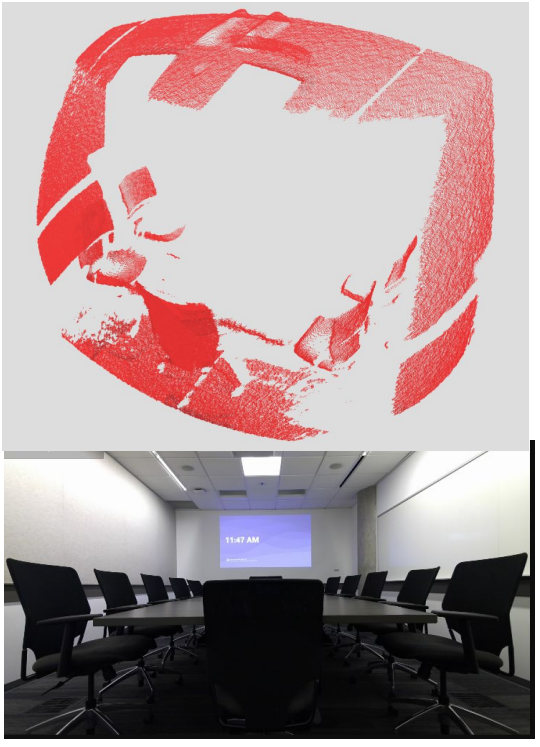


Plant

- Format: rosbag (.bag)
- Camera: Intel RealSense D435
- Intel RealSense configuration: Custom
- RealSense-viewer finds incomplete video: yes
- Streams:
    - Depth
    - Infra1
- Data Range
    - nearest: 164mm
    - furthest: 346mm
    - 0.5mm/step
    - Range: 364 steps
- Invalid data due to hw constraints (too close): yes
- Length: 11s
- Rosbag total size (all streams): 366.4MB
- Number of depth frames (from rosbag info): 347
- FPS: 30
- Resolution: 1280*720
- Dataset Download: https://bit.ly/3f960SH

OFFICE_WFOV_UNBINNED

- Format: Matroska Video (.mkv)
- Camera: Azure Kinect (example video)
- Streams:
    - Depth
    - Infrared
    - Colour
- Data Range
    - nearest: 500mm
    - furthest: ~6139mm
    - step size: 1mm
    - range: 5639steps
- Length: 10s
- Dataset total size (all streams):
- Number of depth frames: 153
- FPS: 15
- Resolution: 1024*1024
- Dataset Download: https://bit.ly/2Yjyt2e



Raw bandwidth calculation

Colour in RGB

$$1280 * 720 * 3\ bytes\ =\ 2764800\ bytes\ =\ 2764.8\ KB\ or\ 2.7\ MB/frame$$
$$2.7MB/frame\ *\ 30\ frame/s\ \ =\ 81\ MB/s\ or\ 4860\ MB/minute\ or\ 291.6\ GB/hour$$

Depth as a GRAY16 or Z16

A depth map uses a single, 16-bit channel, equivalent to two 8-bits, this will produce:

$$1280 * 720 * 2\ =\ 1843200\ bytes\ =\ 1843.2\ KB\ or\ 1.8\ MB/frame$$
$$1.8MB/frame\ *\ 30\ frame/s\ =\ 54\ MB/s\ or\ 3240\ MB/minute\ or\ 194.4\ GB/hour$$

Point Cloud as defined by the Point Cloud Library (PCL)

For a point cloud, each point is represented by

- a signed 32-bit (4 byte) floating point for x,y,z each.
- an optional colour representation, i.e. RGB packed into a 32-bit (4 byte) integer

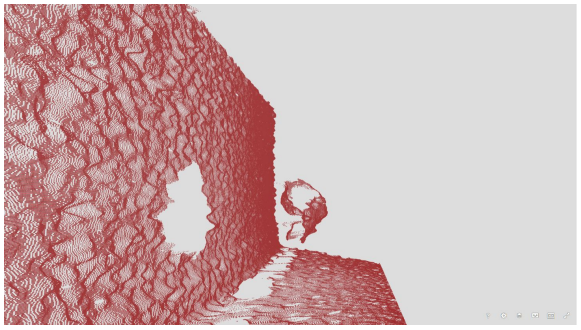
The 1280 by 720p image expressed as a point cloud will produce:

$$1280 * 720 * 3 * 12\ byte_{point}\ +\ 1280 * 720 * 4\ byte_{pixel} = 14745600\ bytes\ or\ 22.1MB/frame$$
$$13.5MB/frame\ *\ 30\ frame/s\ =\ 663MB/s\ or\ 39780\ MB/minute\ or\ 2387\ GB/hour$$

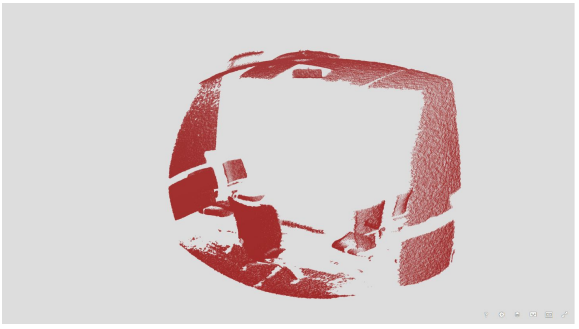
Point cloud Renderings

3DQ

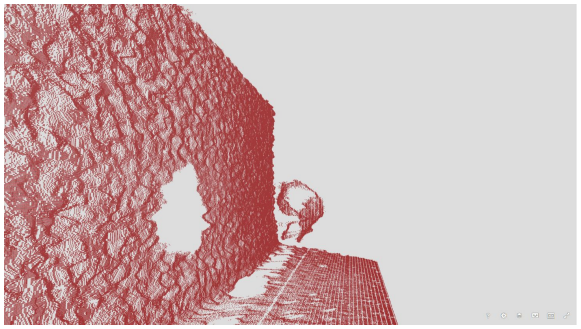
Floating_mug and *OFFICE_WFOV_UNBINNED*
Single-shot point cloud renderings - click to view 3D models
c/r from 0 to 17.2x



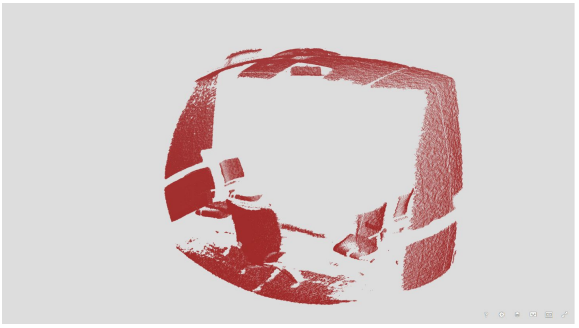
Uncompressed, truncated 250 to 700



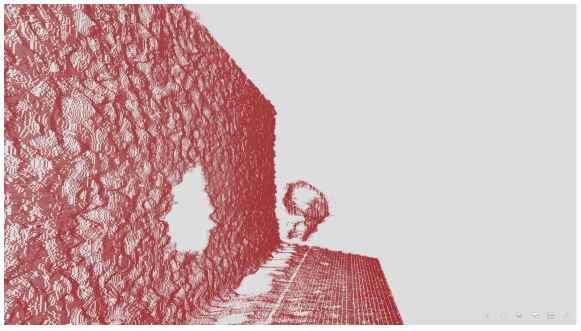
Uncompressed, truncated 1000 to 2530



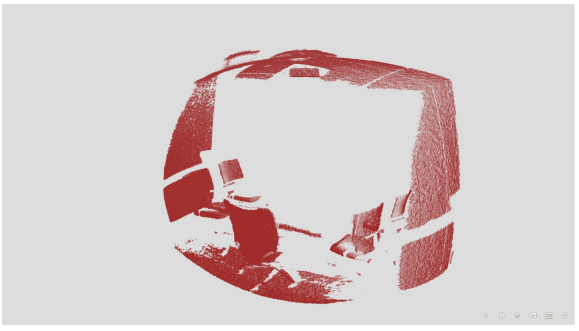
c/r: 9.4x, PSNR: 88dB



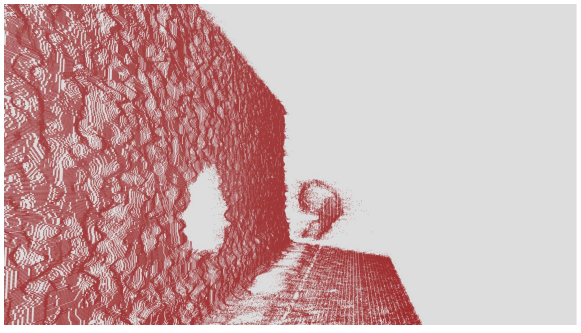
c/r: 7.5x, PSNR: 89.2dB



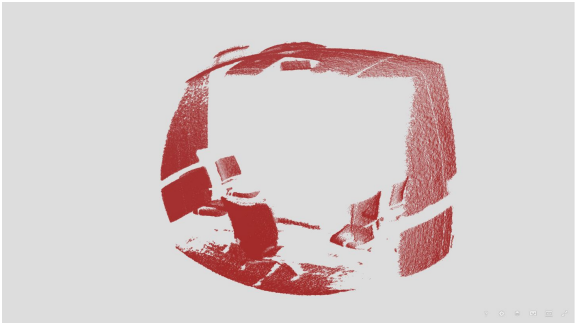
c/r: 12.4x, PSNR: 87.2dB



c/r: 11.8x, PSNR: 88.5dB



c/r: 17.2x, PSNR: 84.6dB

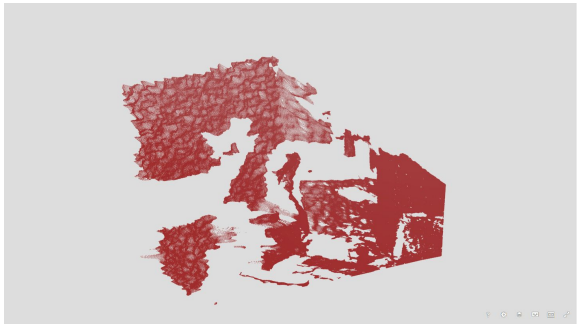


c/r: 16.9x, PSNR: 86.8dB

Fishy-fish and *plant*

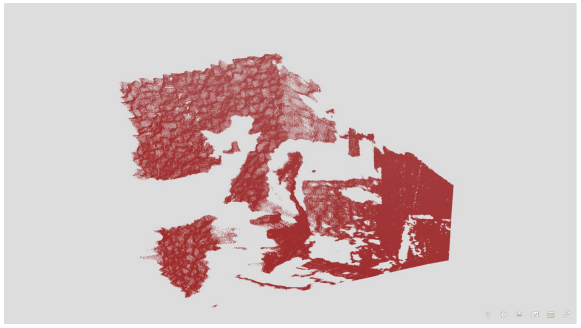
Single-shot point cloud renderings - click to view 3D models

c/r from 0 to 17.6x



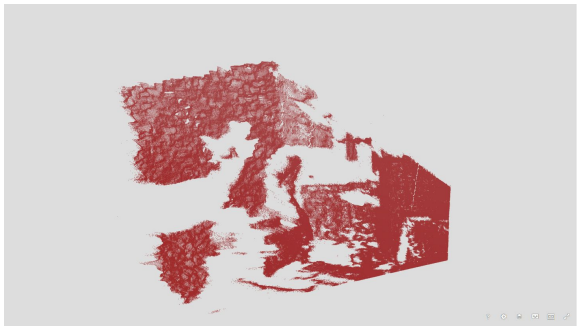
Uncompressed, truncated 250 to 1600

Uncompressed, truncated 300 to 700



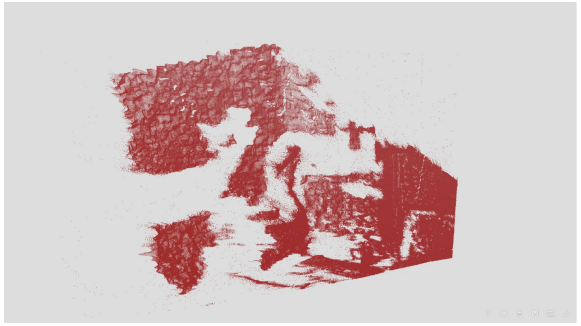
c/r: 8.7x, PSNR: 83dB

c/r: 8.4x, PSNR: 89.4dB



c/r: 12.3x, PSNR: 79.2dB

c/r: 11.7x, PSNR: 81.8dB



c/r: 17.6x, PSNR: 65.7dB

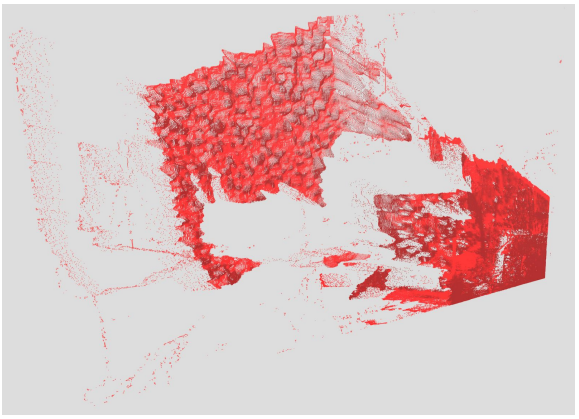
c/r: 16.8x, PSNR: 71.3dB

HUE+webP

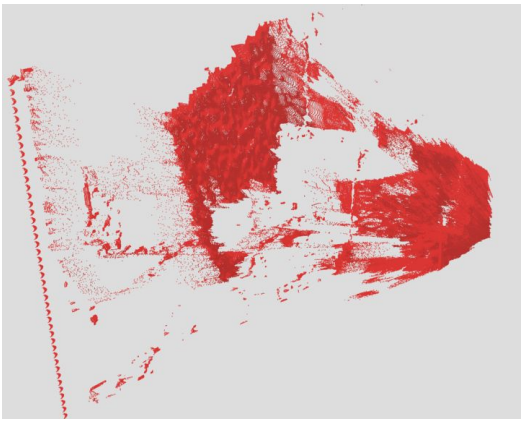
Since all PSNR values are below 70dB we only show the highest and second-lowest setting.

Fishy-fish

Single-shot point cloud renderings
Click descriptions for interactive renderings.



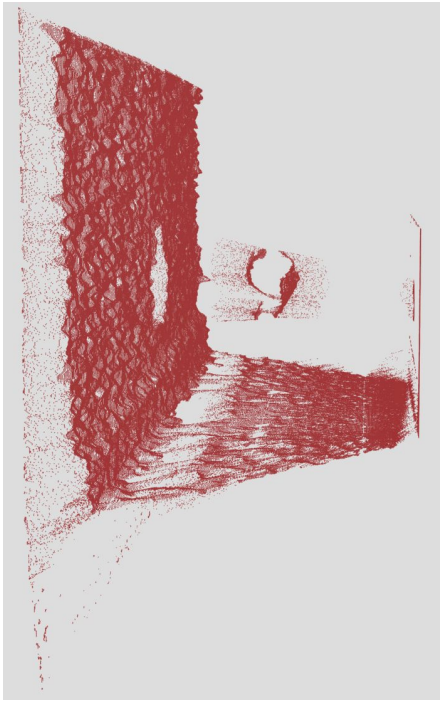
c/r: 106x, PSNR: 47.24dB



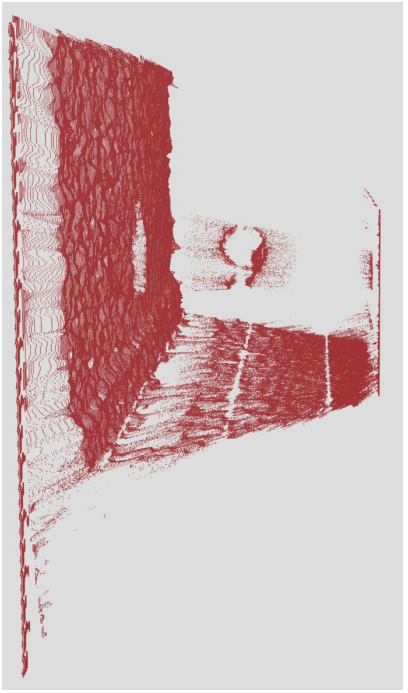
c/r: 207x, PSNR: 47.16dB

floating_mug

Single-shot point cloud renderings
Click descriptions for interactive renderings.



c/r: 9.9x, PSNR: 62.1dB

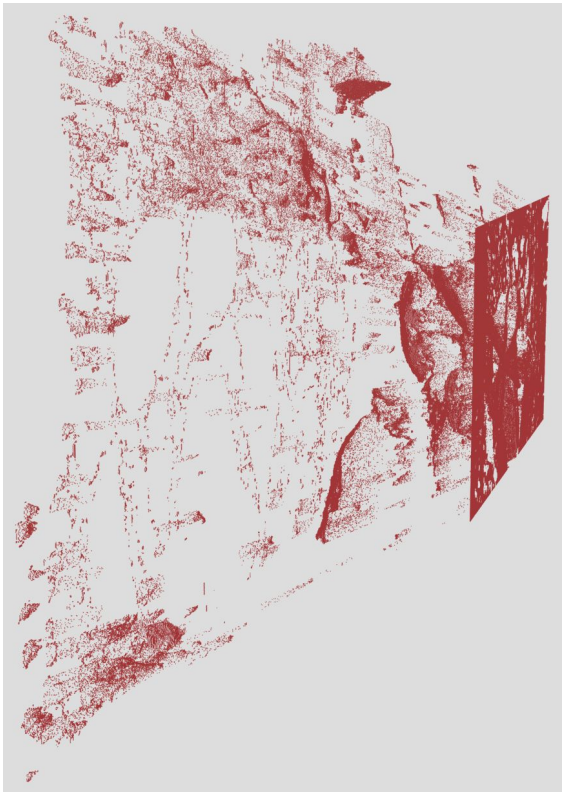


c/r: 52x, PSNR: 61.3dB

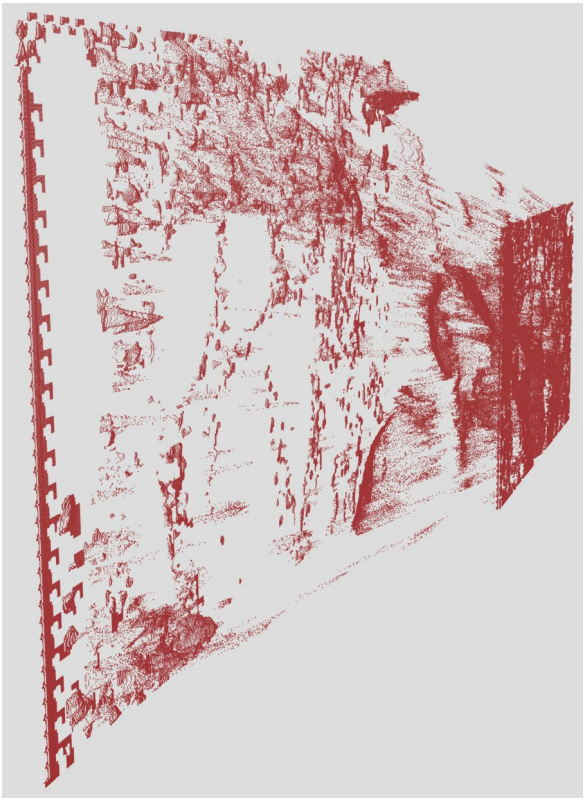
plant

Single-shot point cloud renderings

Click descriptions for interactive renderings.



c/r: 9.1x, PSNR: 57dB

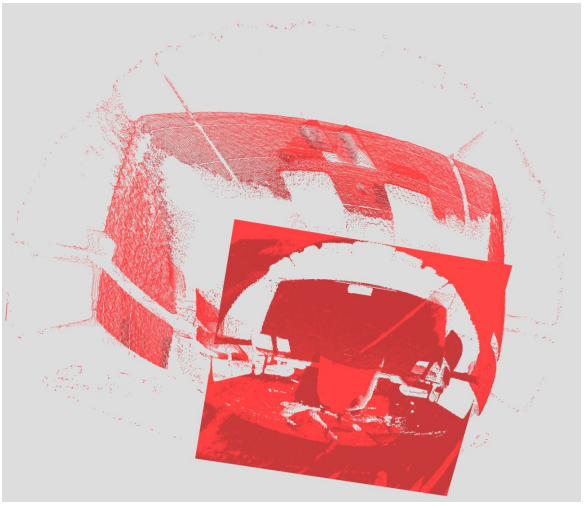


c/r: 53x, PSNR: 52.6dB

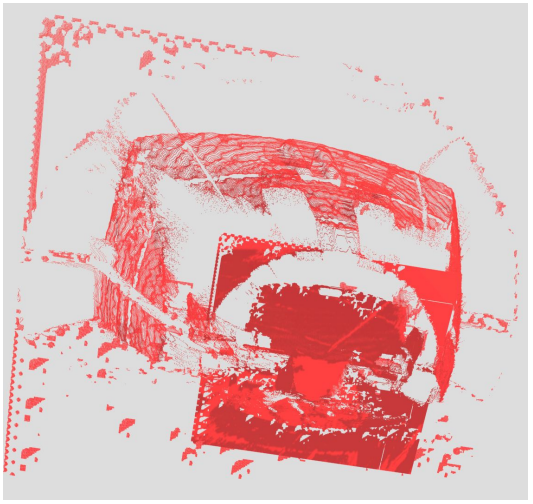
OFFICE_WFOV_UNBINNED

Single-shot point cloud renderings

Click descriptions for interactive renderings.



c/r: 12.1x, PSNR: 51.5dB



c/r: 88x, PSNR: 44.2dB