Depth Video Compression
Technical Whitepaper

Raphael Dürscheid (CTO, Aivero)
João M. Alves (Machine Vision Engineer, 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 existing and widely used approaches for 3D video compression and compares it with Aivero's proprietary compression solution, the 3DQ codec.

1. **Lz4** Lossless, general-purpose compression, very fast compression with very low compression ratios of 2-5x.
2. **PNG** Lossless image compression method, slightly better compression than `Lz4` (2-9x) at high computational cost, limiting real-time applications.
3. **RVL** Lossless, depth image compression, provides compression ratios en-par (3-10x) with PNG, much lower computational cost.
   In certain cases, the algorithm can increase the data size, rather than reduce it.
4. **3DQ** Lossy, depth image compression, allows for controlling the quality/bandwidth tradeoff. At virtually lossless quality, compression ratios (7-12x) match and exceed the above. At acceptable quality, compression ratio reaches up to 20x.

This report introduces the challenges of 3D video compression, discusses the key components of all four approaches before comparing their performance across an open dataset. Finally, it shows the additional functionality of Aivero's toolkit based around the 3DQ codec of capturing, transporting and storing of 3D video.

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, 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, stereo-scopic 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.
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 compute 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.

This paper aims to illuminate the bandwidth bottleneck, investigate three commonly used technologies used for lossless compression of depth video and compares their performance with the lossy compression performance of the 3DQ codec developed by Aivero.

The compression technologies LZ4 (1), PNG (2), RVL (3) and 3DQ (4) are applied to 4 datasets of depth video captured with an Intel RealSense D435 and a Microsoft Azure Kinect camera. We compare the performance in terms of compression ratios and for 3DQ further quantify the tradeoff between image quality and compression ratio.

Finally, we show that the lossy compression approach of 3DQ adds a valuable tool for further reducing data rates. In addition, Aivero's 3D video streaming toolkit enables customers to quickly create storage, transport and machine vision solutions for a growing number of connected 3D cameras.

Appendix A shows the datasets made available to facilitate investigation by researchers and developers

Depth cameras

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. In summary:

- 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.

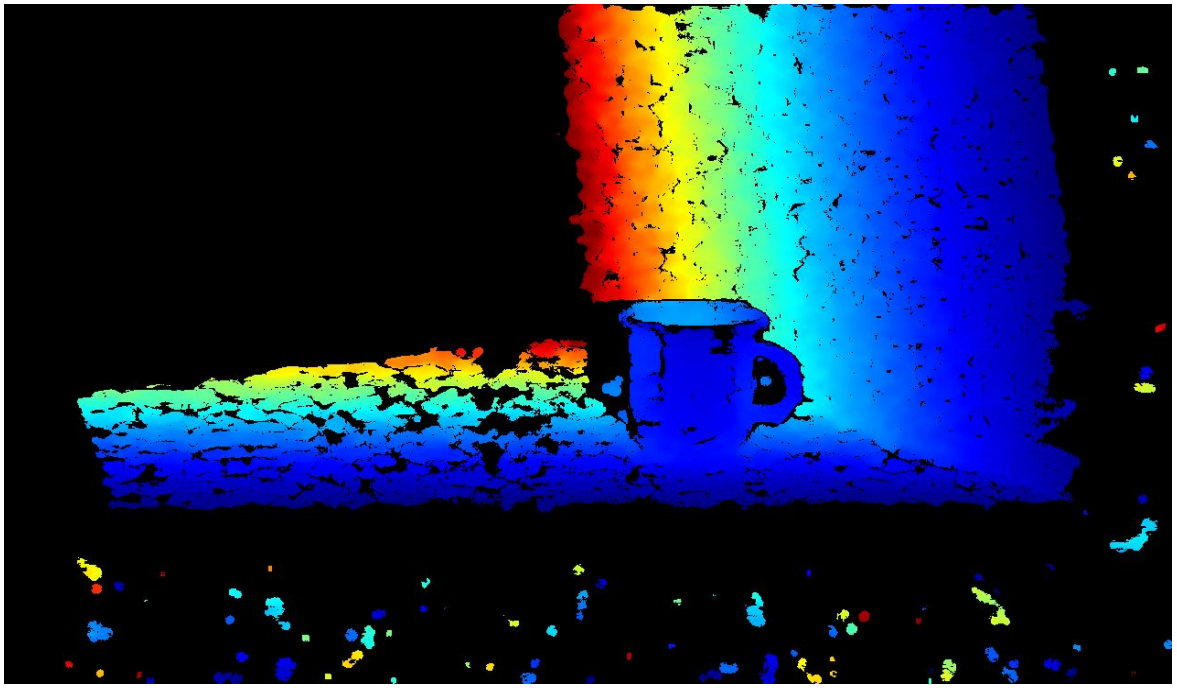
Depth cameras usually provide both a depth stream, as well as a visual stream such as colour or near-infrared (RGB-P). Access to these sets of frames is most often done through a camera-specific SDK, which makes it a lot of work to interface with many different cameras. The different SDKs each provide a different format for storing and loading recordings from their cameras.

- Intel RealSense uses a .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.

Challenges of 3D video compression

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 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 lighting conditions.

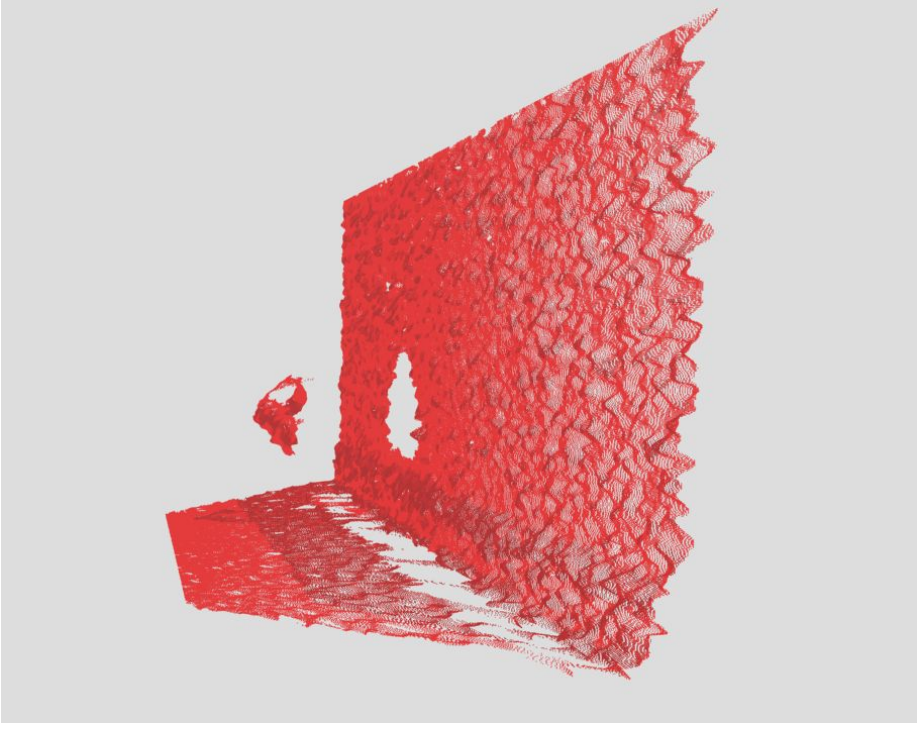


Colourised depth map using a JET colourisation showing the Aivero mug

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 these map to colours in the real world is out of scope here, 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

Uncompressed video data will always produce humongous amounts of data, for 2D and 3D data. Let us illustrate this with an example.

Colour in RGB

Let us consider a single 1280 by 720 pixel image or frame at 8-bit per channel - also known as RGB or RGB24. For every pixel there are 24 bits or 3 bytes:

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

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:

$$1280 * 720 * 3 * 4\ byte_{point}\ +\ 1280 * 720 * 4\ byte_{pixel} = 22118400\ bytes\ or\ 22.1MB/frame$$
$$13.5MB/frame\ *\ 30\ frame/s\ =\ 663MB/s\ or\ 39780\ MB/minute\ or\ 2387\ GB/hour$$

2D video codecs and why they cannot be used alone

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 artifacts that are known as flying pixels, especially around object edges, where invalid pixels (of pixel value 0) usually appear. These artifacts 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 consumption 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 analog TV times where both black and white TVs needed to be supported, as well as colour TVs.
Instead of representing a colour as its components in Red, Green and Blue it represents a 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.

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

The complete background on colour models is out of scope for this paper.
What this shows, is that this conversion alone favours the green components of the image. Furthermore, Y'UV 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.

Thus, when considering an 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 3 8-bit fields - 24-bits in total and appears to `fit` into that data. One can imagine mapping the first 8-bits into the R channel, the second 8-bits into the G channel and leaving the B channel untouched. As shown above, when feeding this data to any encoder we will be losing information when transcoding into YUV and furthermore will be losing information when an encoder compresses the data with the human perception as the target consumer.

Overview of approaches to compression of 3D video

In this section, we will introduce three commonly used approaches to compressing depth data, as well as our proprietary algorithm 3DQ.

Lz4

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 of their .bag files. Lz4 does not use temporal cohesion/inter-frame compression. Typical compression ratio in this context is 2-5.

PNG

PNG is a lossless compression targeted at image compression. It can compress 16-bit grayscale images. PNG does not use temporal cohesion/inter-frame compression.
The typical compression ratio is slightly better than Lz4, but compression time is much slower. Typical compression ratios are 2-9x.

RVL

RVL is a lossless 16-bit depth image compression method developed at Microsoft that can achieve compression ratios similar to PNG, while being much faster both in compression and decompression stages.
RVL makes assumptions about the original data that, if not met, 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 are 3-10x.

3DQ

3DQ is an RGB-D capturing and streaming solution based around a proprietary, lossy compression algorithm for 16-bit depth data, developed at Aivero AS. It utilises hardware acceleration featured in Intel CPU and Nvidia GPU and uses temporal cohesion, allowing for compression ratios of up to 20x, while still being computationally efficient and allowing for high frame rates (30+FPS) in most standard systems. 3DQ focuses on up to 720p resolutions but higher resolutions values can be supported. Asides from compression, 3DQ interfaces with common 3D cameras and video transport applications across networks. This makes 3DQ exceptionally well suited for depth storing and streaming applications. It can be extended using an open RGB-D interface to provide support for new data sources.

Experiments

Setup

In order to assess how 3DQ fares against other available compression techniques, a series of experiments were run across multiple datasets utilising the different compression algorithms and configurations.

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

Datasets

The used datasets consisted of multiple scenes that span across different geometric complexities, noise conditions and overall dynamic motion in 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*

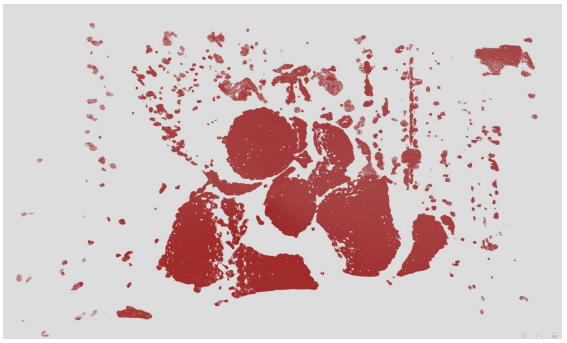
floating_mug and *fishy-fish* have been captured using an Intel RealSense D435 camera with the default settings using the official RealSense-viewer tool provided by Intel. The floating mug shows a metallic mug moving in front of a white background. *Fishy-fish* shows a wooden segmented fish in front of a developer desktop setup.

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. It furthermore represents the depth data in steps of 0.5mm, rather than the typical 1mm. The *plant* dataset contains a rotating Chinese Money Plant.

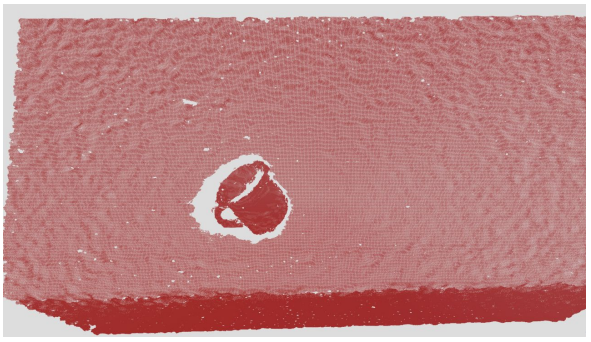
All of the above is captured at 720p resolution and 30 fps.

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. The dataset shows a board room.

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

ROI of depth data

Since 3DQ supports 768, 1536 or 3072 steps in the depth data, a fitting slice was chosen for each of the data sets. This slice is defined by a minimum cutoff value called *near_cut* and a maximum cutoff value called *far_cut.* Data outside of this region is set to zero.
To compare the lossless encoding strategies, these datasets were truncated before compression. The plant data set was truncated to a data range between 300 and 700, floating_mug was truncated with 250 to 700, fishy-fish with 250 to 1600 and the *OFFICE_WFOV_UNBINNED* with 1000 to 2530.

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

So 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 the other performance metric used 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.

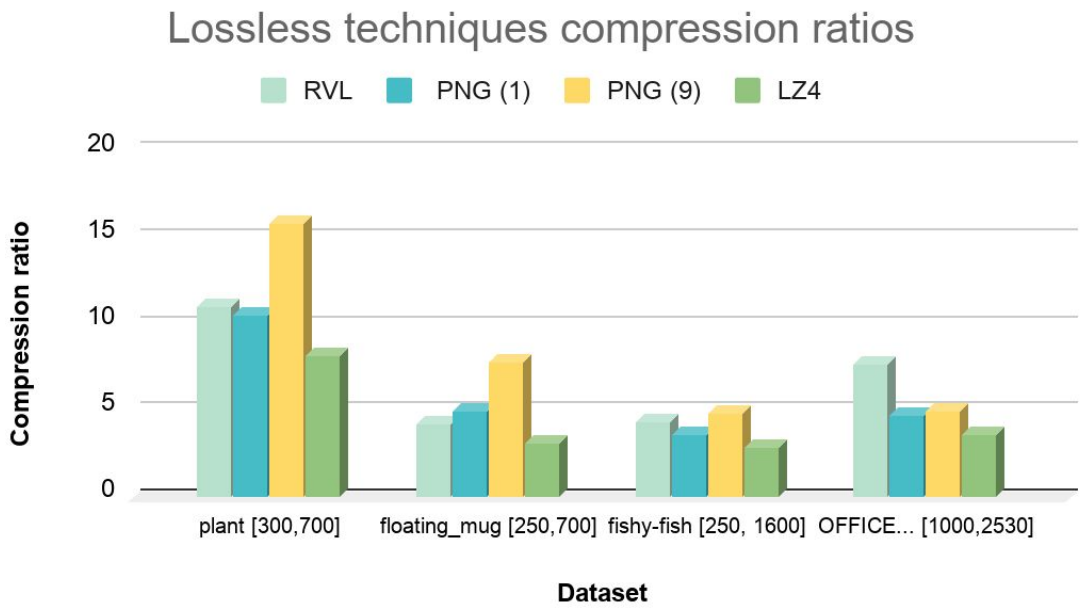
This performance metric was used to evaluate both lossy and lossless compression techniques used.

Results

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

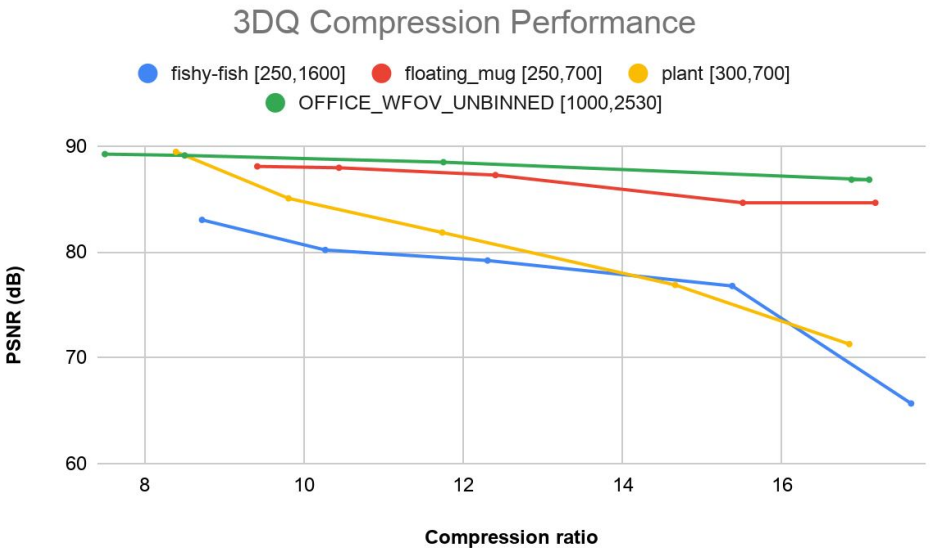
Lossless compression techniques

With the exception of the *plant* dataset the compression ratios of the lossless compression techniques are clustered around a compression ratio of 5x.
The *plant* dataset compresses more easily. In this dataset the depth step size is 0.5mm. At a *near_cut* of 300 and a *far_cut* of 700 the dataset captures data at 150mm to 300mm from the camera. PNG at compression level 9 achieves the best compression ratio here with around 15x. While not presented in this paper, PNG at compression level 9 runs at well below 3 fps on an Intel i9 CPU and is thus not usable for real time compression. PNG compression level 1 is tied with RVL for all but the Azure Kinect dataset. The Azure Kinect dataset *OFFICE_WFOV_UNBINNED* however is significantly affected by the truncation and therefore exhibits a significant amount of 0 valued pixels, something that RVL is especially good at encoding. Lz4 generally performs in the vicinity of RVL and PNG level 1, but should exhibit the lowest computational cost.



3DQ compression and image quality

Using the lossy 3DQ compression on the same data sets results in compression ratios (c/r) between 7.5 and 17.6x and PSNRs of safely above 70dB for all but the highest c/r.



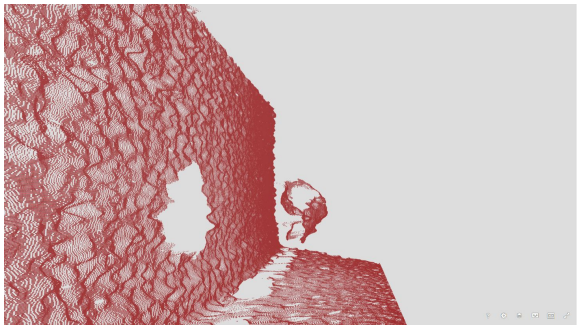
The *OFFICE_WFOV_UNBINNED* and *floating_mug* data sets are compressed with very high quality. Starting at a c/r of 7.5x and PSNR of 89.2dB and c/r of 9.4x and PSNR of 88db, respectively. Throughout the increase in c/r the PSNR stays above 84.6db and 86.8dB respectively.

The *fishy-fish* and *plant* data sets are more dynamic and experience a stronger drop in PSNR. The *plant* data set starts with a c/r of 8.4x at 89.4dB PSNR. The *fishy-fish* starts at c/r 8.7x and 83dB PSNR. At c/r of ~10x, PSNR still remains above 80dB, with the *plant* data set 5dB above. The *plant* data set continues roughly linearly until reaching 71.3dB PSNR at 16.8x c/r. The *fishy-fish* data set drops to 65.7dB at 17.6x c/r.

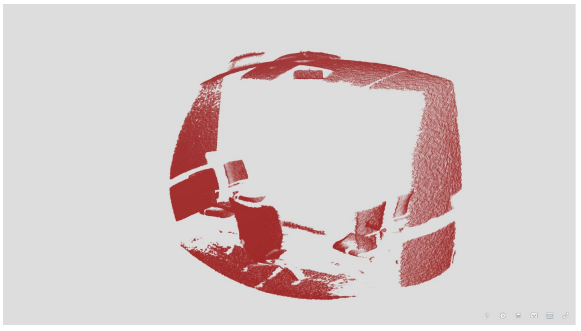
The more static data sets *OFFICE_WFOV_UNBINNED* and *floating_mug* benefit from the inter-frame compression, which is shown by the stable PSNR across the compression ratios. Even at the highest compression level measured the resulting, transcoded point cloud is barely affected.

The more dynamic *fishy-fish* and especially the *plant* data set are trickier to compress. The *plant* data set has plenty of discontinuities in the depth data: Every leaf has a discontinuity as a border and they are ever moving. Furthermore, due to the custom depth settings, the Intel Realsense D435 camera detects many floating patches, which are not connected to any other plane in the view. These pose a challenge to encode. Especially at higher compression ratios we can clearly see stalactite artifacts on discontinuities, significantly reducing PSNR.

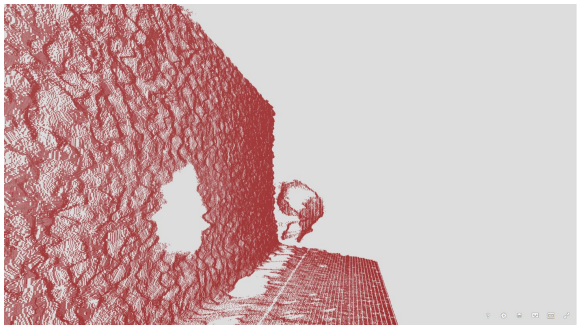
Floating_mug and *OFFICE_WFOV_UNBINNED*
Single-shot point cloud renderings - click to view 3D models
c/r from 0 to 17.2x



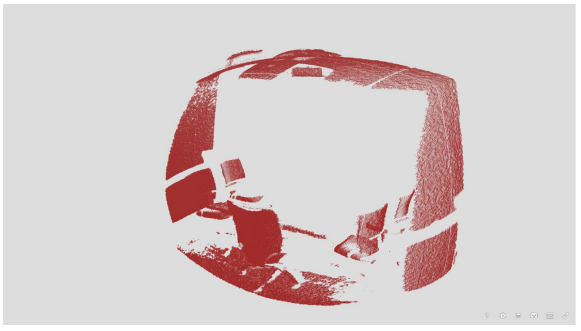
Uncompressed, truncated 250 to 700



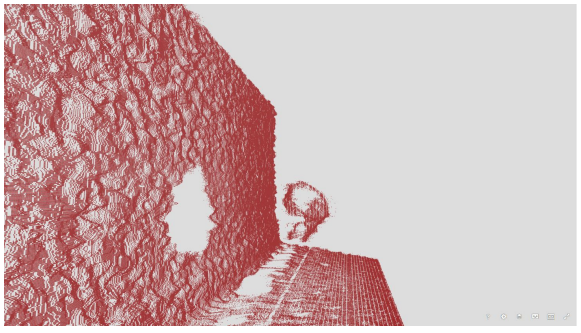
Uncompressed, truncated 1000 to 2530



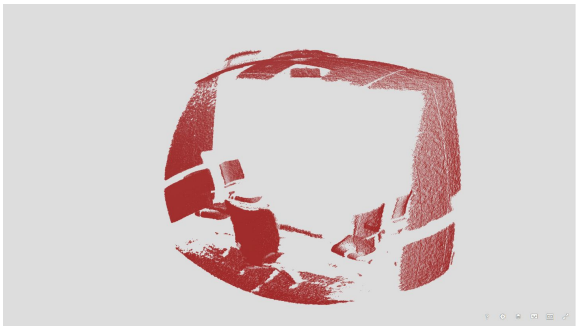
c/r: 9.4x, PSNR: 88dB



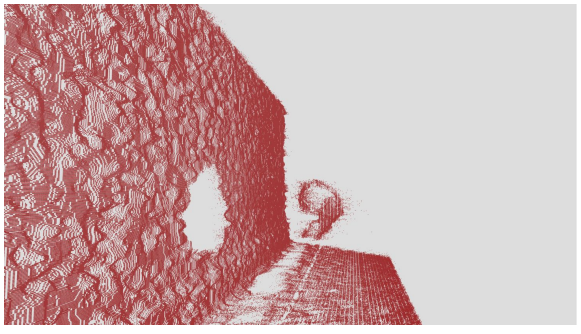
c/r: 7.5x, PSNR: 89.2dB



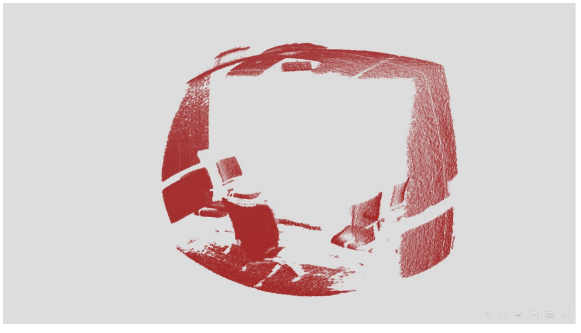
c/r: 12.4x, PSNR: 87.2dB



c/r: 11.8x, PSNR: 88.5dB



c/r: 17.2x, PSNR: 84.6dB

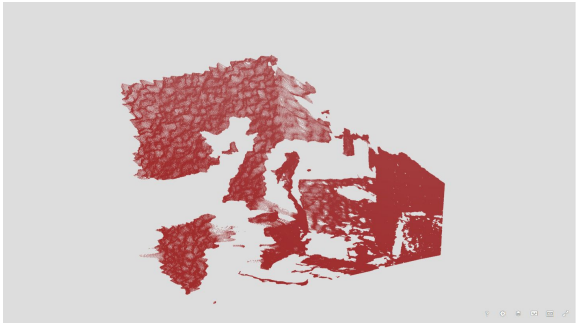


c/r: 16.9x, PSNR: 86.8dB

Fishy-fish and *plant*

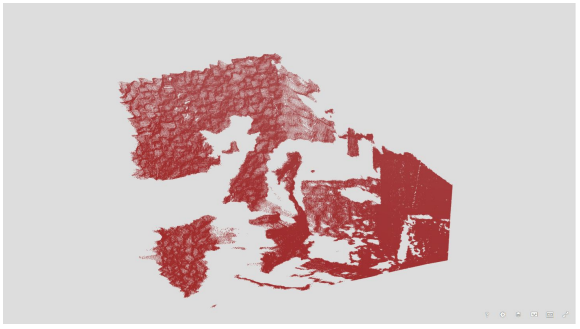
Single-shot point cloud renderings - click to view 3D models

c/r from 0 to 17.6x



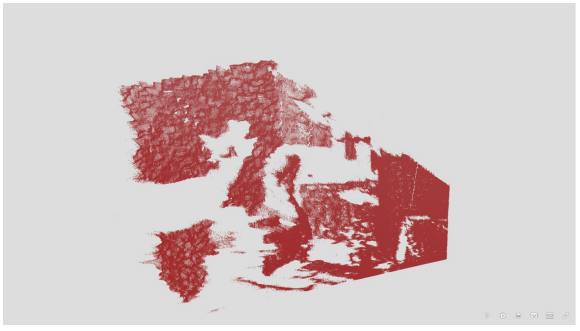
Uncompressed, truncated 250 to 1600

Uncompressed, truncated 300 to 700



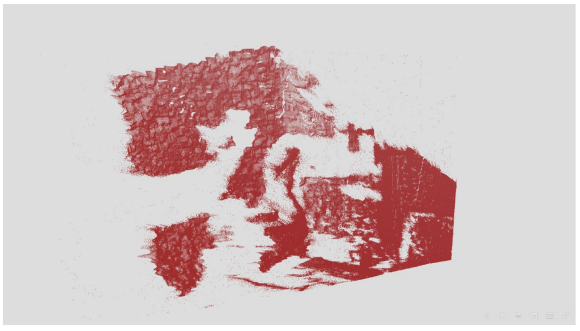
c/r: 8.7x, PSNR: 83dB

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

Discussion

The results shown above showcase the default compression tuning of 3DQ and can be further tuned for specific applications, offering flexibility regarding the depth camera used, available network bandwidth, storage space and overall image acquisition setup.

We have shown that the 3DQ provides a significant reduction in data size for the majority of data sets while being computationally very efficient. It outperforms RVL, PNG and LZ4 in all data sets in terms of compression ratio.

PNG with compression ratio 9 is a serious contender for 2nd place in terms of compression ratio. However, in our tests, it did only achieve frame rates of under three (3) fps and is therefore limited in its applications.

RVL, PNG compression level 1 and LZ4 achieved similar compression ratios and were especially effective in compressing the plant data set.

Along with 3DQ, RVL can certainly achieve framerates of over 30fps on modern machines, LZ4 will follow suit here, while PNG lv1 will create a higher CPU load than either of the others.

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 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 is 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

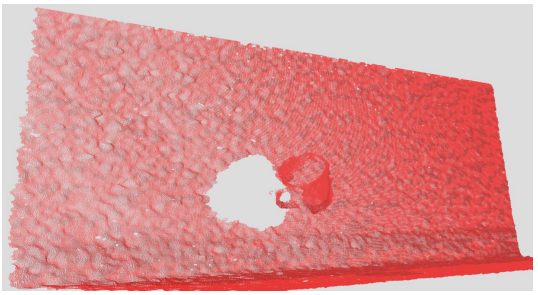
Acknowledgements

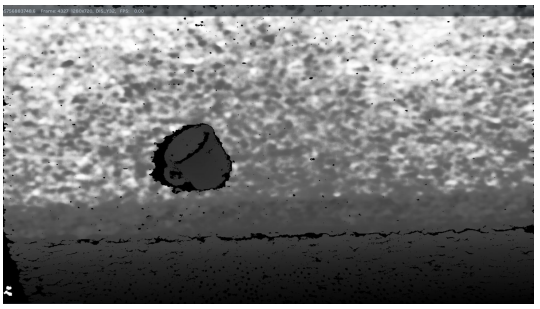
We would like to thank our dear dev team colleagues Tobias Morell, Niclas Overby, Andrej Orsula, Kasper Steensig and Vojtech Jindra for their great work and dedication to 3DQ and providing feedback and insights. Martin Svangtun spent a lot of time helping to get this paper out. Christian Rokseth gave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

Appendix - datasets

floating_mug

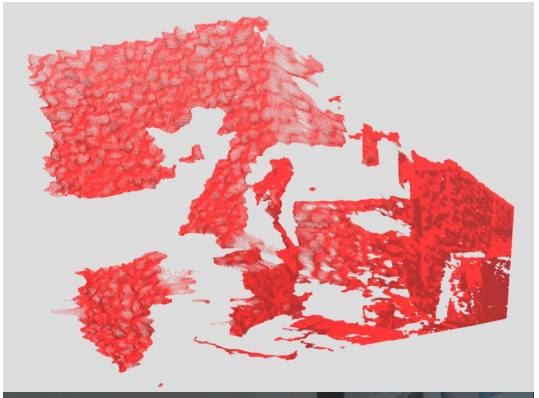
- Format: rosbag (.bag)
- Camera: Intel RealSense D435
- Intel RealSense configuration: Default
- RealSense-viewer finds incomplete video: yes
- Streams:
  - Depth
- Data Range
  - nearest: 259mm
  - furthest: ~667mm
  - step size: 1mm/step
  - range: 408 steps
- Invalid data due to hw constraints (too 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

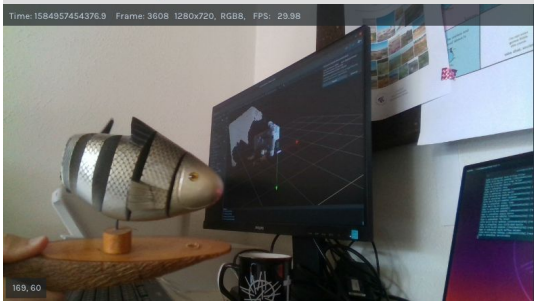




fishy-fish

- 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