Real-time Point Cloud Compression

Tim Golla[1] and Reinhard Klein[2]

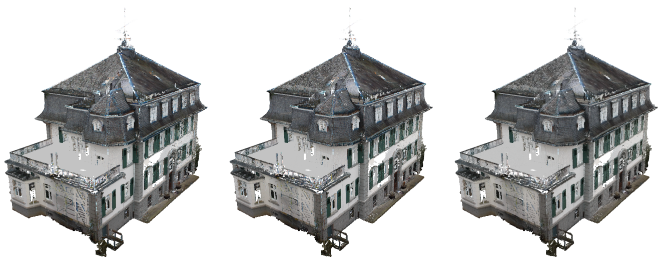
Fig. 1. Left: The original Frankenforst dataset with colors, consisting of 43.6 million points (624 MB). Center: Reconstruction from a compressed version with a file size of 4.20 MB, an $\mathrm{RMSE}_{\mathrm{SNN}}$ of 0.0065 m and an $\mathrm{RMSE}_{\mathrm{RGB}}$ of 17.09. Right: Reconstruction from a compressed version with a file size of 8.00 MB, an $\mathrm{RMSE}_{\mathrm{SNN}}$ of 0.0028 m and an $\mathrm{RMSE}_{\mathrm{RGB}}$ of 11.69. Note that some fine-scale detail is smoothed out, but the overall visual impression is still good, despite the high compression ratio.

***Abstract*— With today's advanced 3D scanner technology, huge amounts of point cloud data can be generated in short amounts of time. Data compression is thus necessary for storage and especially for transmission, e.g., via wireless networks. While previous approaches delivered good compression ratios and interesting theoretical insights, they are either computationally expensive or do not support incrementally acquired data and locally decompressing the data, two requirements we found necessary in many applications. We present a compression approach that is efficient in storage requirements as well as in computational cost, as it can compress and decompress point cloud data in real-time. Furthermore, it is capable of compressing incrementally acquired data, local decompression and of decompressing a subsampled representation of the original data. Our method is based on local 2D parameterizations of surface point cloud data, for which we describe an efficient approach. We suggest the usage of standard image compression techniques for the compression of local details. While exhibiting state-of-the-art compression ratios, our approach remains easy to implement. In our evaluation, we compare our approach to previous ones and discuss the choice of parameters. Due to our algorithm's efficiency, we consider it as a reference concerning speed and compression rates.**

This work was partially funded by the German Research Foundation (DFG) under grants KL 1142/9-1 and KL 1142/9-2 (Mapping on Demand) and by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 600908 (DURAARK) 2013-2016 and under grant agreement no. 323567 (Harvest4D) 2013-2016. The David point cloud is courtesy of the Digital Michelangelo Project, Stanford University. The Thermobremen dataset is recorded by Dorit Borrmann and Andreas Nüchter from Jacobs University Bremen gGmbH, Germany. The Frankenforst datasets are courtesy of the group of Prof. Kuhlmann, Institute of Geodesy and Geoinformation, University of Bonn.

[1] `golla@cs.uni-bonn.de`
[2] `rk@cs.uni-bonn.de`
Both authors are with the Institute of Computer Science II, University of Bonn

I. INTRODUCTION

Modern 3D laser scanners like the Velodyne HDL-64E can acquire 3D point cloud data at high rates of up to 1.5 million points per second. For many applications, e.g., robotics, it is desirable to transmit this data in real-time via restricted bandwidth networks. For this, reducing the data as far as possible at real-time rates is necessary. We consider an algorithm that is able to compress points at least at the scanner rate, i.e., at 1.5 million points per second, as real-time capable. One way to achieve real-time rates is OctoMap [7], which was developed especially for robotics applications. While not primarily intended as a compression approach but a data structure for mapping environments, it can be considered and used as such. It achieves real-time processing rates and can reduce, for instance, a dataset consisting of 315.5 million points to 45 MB at a root mean square error (RMSE) of 0.006 m [26]. Approximately 10 years ago, compression algorithms were developed in the computer graphics community that are able to achieve an additional compression factor of approximately 10. One example is the algorithm of Ochotta and Saupe [19], who parameterized the point cloud over planar patches and compressed the resulting height maps via a wavelet transform.

Unfortunately, this algorithm does not support incremental compression and local decompression, in the sense that the user can add new data to the representation and randomly extract data at specific locality. Later, the idea to use an atlas of height maps was introduced into the robotics area by Ruhnke et al. [25], [26], whose approach achieves similar compression rates, but also guarantees locality during the decompression. The drawback of all these approaches is that they require high computational effort and are currently far from real-time with respect to compression time.

In this paper, we propose a real-time out-of-core compression approach that is able to compress 3D points with compression ratios that are at least comparable to, but mostly better than the compression ratios of previous compression approaches. Furthermore, our algorithm supports incrementally acquired data, as is necessary, e.g., in online robotics applications. Decompression also works at real-time rates and is local, i.e., only the necessary parts of the data can be extracted. It inherently supports subsampling, i.e., a subsampled version of the point cloud can be reconstructed from the compressed representation. Where possible, we use standard techniques in order to make our algorithm easy to implement. Last but not least our approach is able to trade compression ratios for speed. To the best of our knowledge, our algorithm is the first to exhibit all these properties.

II. Related Work

There are a number of approaches that employ height map encoding over a planar domain as a basis. The first to describe a height map-based point cloud representation were Pauly and Gross [22], who proposed to use an atlas of height maps. They use it to perform spectral analysis of point clouds. Based on this idea, Ochotta and Saupe [19] introduced height map-based point cloud compression. They recursively subdivide a point cloud by splitting along a plane orthogonal to the first principal component vector, obtained by principal component analysis (PCA). The subdivision ends as soon as all points belonging to a half-space fulfill the normal condition, i.e., all normals belonging to the points of a half-space are contained in a cone with its apex at the origin and an apex angle that does not exceed a user-specified threshold. The height maps are compressed via a wavelet transform. In addition, binary masks (subsequently called occupancy maps) defining the patch shape are generated and compressed via arithmetic coding. The authors refined this approach in 2008 [20] and used a generalized Lloyd algorithm [18] to generate the atlas of height maps which provides partitions with a smaller number of patches and a smaller approximation error. While resulting in very good compression rates, compression times are not sufficient for a real-time application. Furthermore, the decomposition in both algorithms is performed on the whole point set and does not support out-of-core processing. Hubo et al. [12], [13] used the idea of height maps for compression by selecting a number of representative height maps and replacing all other height maps with these representatives. Their compression times are in the order of hours.

The approaches of Ruhnke et al. [25], [26] are based on a sparse coding of height maps, which are extracted from a patch decomposition of static point clouds. In [25], they extract rectangular patches from the surfaces, which are projected onto a regular grid whose orientation is determined by the local surface's principal axes. The compression scheme in [26] is extended by a recursive decomposition of larger patches that uses an adaptive subdivision criterion based on the reconstruction error, i.e., instead of patches of fixed size and resolution, the authors encode a hierarchy of surface patches. Digne et al. [4] follow a similar approach and decompose a point cloud by a greedy approach into circular planar patches. While they use the K-SVD dictionary learning algorithm described in [24], Ruhnke et al. [25], [26] rely on a modification named wK-SVD, that allows to specify weights for each element of the encoded signals. Their usage of wK-SVD stems from the need to account for empty or undefined pixels in the height maps. The dictionary approach allows for local decompression. Digne et al. report the best compression ratios on the David point cloud so far. For this point cloud they report a compression time of 8 minutes and a decompression time of 10 minutes. In contrast to most other approaches, they do not use occupancy masks and thus fill small holes in the point cloud, as they demonstrate on a point cloud recorded in the city center of Bremen.

In contrast to these approaches that compress height maps over planar domains, Schnabel et al. [28], [29] use height maps over different, also non-planar domains, namely geometric primitives like planes, spheres, cylinders, cones and tori. Instead of K-SVD, they use vector quantization for the height map compression. They achieve real-time decompression rates, but the compression times are too long for real-time purposes. This approach has later been accelerated and extended in order to support incrementally acquired data [5]. As the reported compression rates are not better than the ones reported by Digne et al. [4] and the decomposition is more difficult to compute, we do not consider non-planar domains in our new algorithm.

There is a number of tree-based approaches. For example Huang et al. [10] and Schnabel and Klein [27] hierarchically encode the tree structure and thus cannot extract randomly selected parts of the data. Their compression ratios are not on par with the height map-based approaches. The approach presented by Hubo et al. [11] allows random access, but also exhibits compression ratios not on par with later works.

Further compression methods are [9], [14], [15], [17], [32], which are all not real-time capable. Kammerl et al. [16] presented an algorithm for real-time point cloud compression. Their approach is well-suited for streams of point cloud data with a fixed-position camera, by exploiting redundancy between consecutive frames. It is inapt for point cloud data of different origin.

III. Our Method

A. Overview

In order to support out-of-core processing, incrementally acquired data and partial decoding, we first split the data into *compression chunks*. For this purpose, we decompose the world space into voxels of a user-specified size that are compressed individually. In combination with out-of-core sorting, this also allows for compressing static point clouds of arbitrary size. In the online scenario, incoming data is buffered for a voxel until new incoming data no longer falls into that voxel. The correct moment for compression can also be controlled by the robot navigation, i.e., it is started when an area has been completely explored. Since the voxel grid is only implicitly computed, it can grow as needed and thus scenes of arbitrary size can be compressed. Data size slightly increases as the voxel size decreases. This parameter thus has to be chosen considering a trade-off between locality and compression ratio.

If the application requires an immediate data transmission, it is also possible to skip the previously described buffering step and compress each scan individually with the method described in the following sections.

Like previous approaches, the compression of the point clouds belonging to a voxel is based on a decomposition into point *clusters* that can be represented by *patches*, parameterized over planar 2D domains. In order to represent fine-scale details, height and occupancy maps on these domains are generated. The height maps account for offsets of the original geometry from the domains, while the occupancy maps account for holes in the geometry, like windows in a building facade.

B. Point Cloud Decomposition and Patch Computation

Having a coarse decomposition of the point cloud into compression chunks, we now have to generate for each chunk a set of patches. Different patch generation techniques were suggested in the literature, e.g., in [4], [25] and [29]. We found that already a simple decomposition by a voxel grid, where each voxel corresponds to one patch, delivers good results. The choice of the cell size has a direct effect on the compression ratio and the reconstruction quality and is thus the primary control over size vs. quality. All points belonging to one voxel cell are considered as a cluster. All clusters are overlap-free. Although this decomposition method is much simpler than previous ones, we found its results sufficiently good. It is, however, possible to achieve better compression ratios and quality by using a data-adaptive approach. For this, the chunk is recursively subdivided into an octree structure until sufficiently small clusters have been generated. We specify a number of points n_l and subdivide until all of the tree's leaf node voxels contain at most n_l points, each of which is then represented by a patch. If normals are present, a normal cone criterion like in [19] can be used to subdivide further. Although slightly more involved than the voxel grid decomposition, this method is still fast enough for real-time applications.

Each patch is characterized by its position, orientation and size. We obtain the patch's normal n by a principle component analysis (PCA) of the associated point cluster. The patch's rotation around the normal could also be determined by PCA. Due to the decomposition by cubical shapes, the projection of the point clusters often have a rectangular shape. The principal components tend to point to this rectangle's corner. It is desirable to orient the patch such that the rectangle's edges are parallel to the coordinate system's axes. This leads to a better usage of the height and occupancy maps and to a better compression with oriented wavelet bases and the oriented discrete cosine transform, as these also are oriented along the coordinate system. We thus orient the local domain's coordinate system along the global coordinate system. It is computed as follows. Let e_1^p, e_2^p, e_3^p be the x-, y- and z-axis of the local coordinate system and $\times$ be the vector cross product. Then: $e_1^p = n \times (1,0,0), e_2^p = n \times e_1^p, e_3^p = n$. As the coordinate system's origin we choose the point cluster's center of mass.

C. Height Map and Occupancy Map Computation

The height and occupancy maps are 2D bitmap images of a predefined resolution $r_x \times r_y$. In all our experiments, we chose $r_x = r_y$ and in most cases $r_x = r_y = 16$. As the subsequently employed image compression techniques operate on image patches of size $8i \times 8j,\ i,j \in \{1,2,...\}$, choosing the patch size accordingly leads to better results. Additional maps can be used for colors, normals and other information associated with the original points. Having the same fixed resolution, the maps vary in physical dimension in order to account for their associated point cluster's extent. This has two advantages. Firstly, it avoids empty regions on the maps, which would lead to wasted space and unnecessary high frequencies, which would be harder to compress. Secondly, it makes – especially in combination with the octree-based decomposition – our approach data-adaptive: sparsely sampled regions are represented by larger patches, while densely sampled regions are represented by small patches. This leads to a more space-efficient usage of the height maps. We compute the cluster's bounding box in its associated coordinate system, which gives us the necessary extent.

We transform each of the cluster's points into the local coordinate system (e_1^p, e_2^p, e_3^p), yielding its coordinates (x^p, y^p, z^p). (x^p, y^p) determine the position in the height map and z^p the height h, which is stored at (x^p, y^p) in the height map. Also the respective position (x^p, y^p) in the occupancy map is set to 1. If several points are projected onto the same pixel, the height map's value at that position is set to the average of their heights. Note that the height values are floating point values and can be negative and – in principle – of arbitrary magnitude. We quantize them afterwards – see section III-D. Color and other maps are computed analogously to the height maps. We improve the compression ratio by smoothing out the unoccupied areas of the height and color maps, removing high frequencies. In our experiments we found recursively applying a box filter of size 5 three times to yield good results. This has no direct effect on the reconstruction results, as these areas are not used for reconstruction. It has an indirect effect however, as lower frequency information can be better represented by the employed discrete cosine, resp. wavelet compression.

D. Quantization and Compression

All non-integer values are quantized. We represent the patch orientation with three values, which are quantized to 8 bits each. The patch position is quantized to three 16 bit values and the size is quantized to 8 bits. Height map values are quantized to 8 bits as well, in order to be conformal with standard image compression algorithms. For all quantities, the minimum value m and the maximum value M within a compression chunk are determined and stored in a separate file. All values v are then set to $v' = \frac{v-m}{M-m}$ and quantized to the corresponding number of bits. We tile all maps of one type belonging to one chunk into one large map. Depending on the map type, these maps are compressed differently. Height and color maps are compressed with the JPEG or the lossy JPEG 2000 standard [2], [30]. JPEG is faster, while JPEG 2000 delivers smaller file sizes at comparable error rates. JPEG 2000's most important steps can be summarized as applying the Cohen-Daubechies-Feauveau 9/7 wavelet transform [3] and encoding the wavelet coefficients with a context-adaptive arithmetic coder. Occupancy maps are compressed via the lossless JBIG2 algorithm [8], [21]. We compress the patch positions, orientations and sizes with the Lempel-Ziv-Markov chain algorithm (LZMA) [23], an extended variant of the LZ77 algorithm [33].

E. *Decompression and Subsampling*

For decompression, the JPEG/JPEG 2000, JBIG2 and LZMA-compressed files are decoded, the big maps are split into separate ones, which are associated with their respective patches. Where a pixel on an occupancy map is marked as occupied, we look up the respective height and color values in the height resp. color maps and restore the point correspondingly by transforming the height value to a global position from the patch's local coordinate system. For subsampling with a factor of i, only each ith point is reconstructed.

IV. Evaluation

A. *Error Measures*

In order to compare our results to those presented in the related work, we describe the error measures for 3D point clouds used in the literature. We use each respective measure in order to be able to compare our results to those of the previous work. For all measures, we consider two point clouds P and Q, whose distance is to be measured. We consider P to be the original point cloud and Q to be its reconstruction from a compressed representation.

The *Nearest Neighbor root mean squared error (RMSE)* is computed by measuring the euclidean distance of each point p in P to its nearest neighbor q in Q. Employed e.g. in [29]. The mean squared error (MSE) is defined as: $\text{MSE}_{\text{NN}}(P,Q) = \sum_{p \in P}(p-q)^2/|P|$. *The root mean squared error (RMSE)* then is: $\text{RMSE}_{\text{NN}}(P,Q) = \sqrt{\text{MSE}_{\text{NN}}(P,Q)}$, where $|P|$ is the number of points in P. This is a single-sided error measurement. Thus, $\text{RMSE}_{\text{NN}}(Q,P)$ should be computed as well, in order to account for inliers and outliers.

The MSE and RMSE can also be defined for the color values of a point cloud [25], [26]. Let $p.r, p.g, p.b \in \{0, ..., 255\}$ be the RGB values of point p. For the red channel: $\text{MSE}_{\text{R,NN}}(P,Q) = \sum_{p \in P}(p.r - q.r)^2/|P|$, Here, $q \in Q$ is again the spatially nearest neighbor to p. $\text{MSE}_{\text{G,NN}}(P,Q)$ and $\text{MSE}_{\text{R,NN}}(P,Q)$ are defined analogously. $\text{MSE}_{\text{RGB,NN}}(P,Q) = \frac{1}{3}\text{MSE}_{\text{R,NN}}(P,Q) + \frac{1}{3}\text{MSE}_{\text{G,NN}}(P,Q) + \frac{1}{3}\text{MSE}_{\text{B,NN}}(P,Q)$

The *Symmetric Nearest Neighbor RMSE* is used in [25], [26]. It is the symmetric version of the Nearest Neighbor RMSE. $\text{RMSE}_{\text{SNN}}(P,Q) = \sqrt{0.5\text{MSE}_{\text{NN}}(P,Q) + 0.5\text{MSE}_{\text{NN}}(Q,P)}$. Analogously for the color values: $\text{RMSE}_{\text{RGB,SNN}}(P,Q) = \sqrt{0.5\text{MSE}_{\text{RGB,NN}}(P,Q) + 0.5\text{MSE}_{\text{RGB,NN}}(Q,P)}$. Unless otherwise stated, we use this measure in our evaluation.

The *moving least squares RMSE* is used in [4], [19] and [20] and defined as the sum of the shortest euclidean distance of P's points to the moving least squares (MLS) surface [1] of Q. While intuitively probably a better measure than the ones above, it has the disadvantage that the moving least squares surface is parameter-dependent. Hence, the distance measure of two point clouds varies, depending on the choice of parameters for the MLS surface. We thus do not use it in our evaluation.

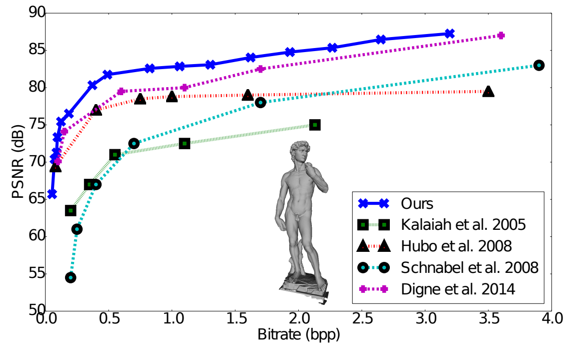

Fig. 2. Compression results on the David point cloud using the voxel grid decomposition. Bits per point (bpp) vs. PSNR. Our algorithm outperforms previous approaches. For comparability, PSNR is based on the RMSE_{NN}.

The *peak signal-to-noise ratio (PSNR)* is given as: $\text{PSNR} = 20\log_{10}\frac{\text{peak}}{\max(\text{RMSE}(P,Q),\text{RMSE}(Q,P))}$, where in the field of geometry, the peak is usually chosen as the bounding box diagonal [6]. In the related work, the PSNR is only used in combination with the Nearest Neighbor RMSE, although all of the RMSE definitions above could be used.

B. *Compression Results*

We performed the evaluation with a selection of point clouds of different origins and properties on a standard desktop PC with an Intel Core i7-4930K CPU. The David statue dataset with 28.2 million points has become one of the most used datasets in the computer graphics community. We thus employ it to compare our results to those reported in the previous papers [4], [13], [15], [29] – see Fig. 2. Considering a fixed bit rate, our approach achieved a higher PSNR, i.e., reconstruction quality, than previous methods. Compression times varied between 6.2 and 14.6 seconds, depending on the quality setting, i.e., from 1.93 to 4.55 million points per second were processed. Decompression times varied between 0.7 and 10.3 seconds.

The fr1/room dataset [31] consists of a sequence of RGB-D images. Like Ruhnke et al. [26], we generated a point cloud from it as the SLAM solution with the provided ground truth trajectory. The resulting point cloud consists of 315.5 million points. Our evaluation on this dataset showed that our approach is substantially faster than the ones described in [25], [26], while providing even better compression ratios – see Table I. The compression at a quality comparable to the one presented in [26] took 43 seconds, while their approach took 36 minutes. That is, we achieve a compression performance of 22.7 million points per second. To evaluate the effects of the compression chunk size on the compression result, we ran the compression multiple times with different voxel sizes. The results for the fr1/room dataset are plotted in Fig. 5. As can be seen in this plot, the resulting data size increases slowly as the chunk size decreases. The voxel size can be reduced down to about $10^{-3.5}\%$ of the whole dataset's volume with a negligible impact on the compression ratio in comparison to a voxel size of 100%, i.e. only

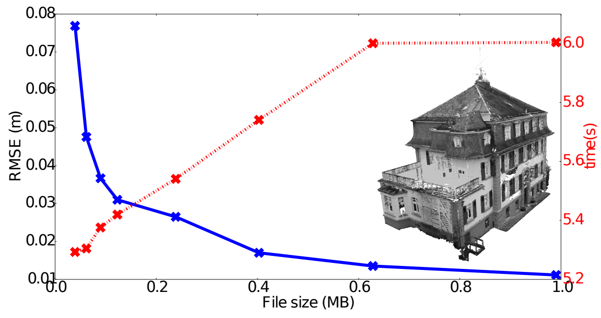

Fig. 3. Compression results of the Frankenforst dataset without colors, consisting of 43.6 million points (499 MB), Octree decomposition, JPEG compression. MB vs. RMSE (blue), MB vs. time (red, dotted line)

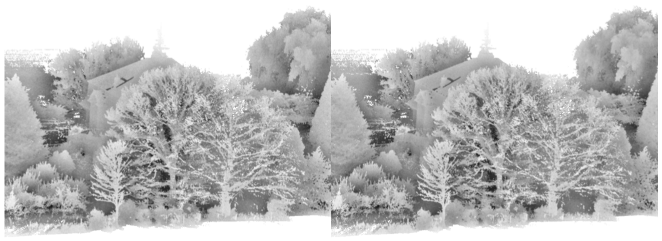

Fig. 4. The highly nonplanar Frankenforst 2 dataset ($262.2 \cdot 10^6$ points, 3001 MB). Left: Original. Right: Reconstruction after compression to 9.0 MB with an $\mathrm{RMSE}_{\mathrm{SNN}}$ of 0.028 m.

one voxel, which provides the best compression ratio. The Thermobremen dataset consist of several scans acquired in the city center of Bremen, Germany. The point cloud obtained by combining these scans consists of 40.7 million points. The compression results with our approach are plotted in Fig. 6. The compression for the complete dataset took between 3 and 6.5 seconds, i.e., 6.3 to 13.6 million points were processed per second. The Frankenforst dataset was generated from multiple scans performed with a terrestrial laser scanner. It consists of 43.6 million points – see Fig. 1. The Frankenforst 2 dataset contains the same building and also a larger surrounding area with vegetation. See Fig. 4 for a visualization and Table II for the compression results. Although this dataset contains highly nonplanar regions, it could be reconstructed at a low error. In order to evaluate the compression without buffering, we compressed the first scans of the fr1/room resp. the Thermobremen dataset. The compression was performed with at least 2.88 million points per second – see Table II.

V. Conclusion

We presented a compression approach that outperforms previous methods in terms of speed as well as in terms of compression ratio at comparable quality. Our approach is fast enough for real-time applications in terms of the number of points that can be processed per second. It is possible to immediately compress each single scan in real-time, however, for the best compression ratios, a buffering step is required, which might cause intermediate delays. Our approach supports incrementally acquired data as incident in

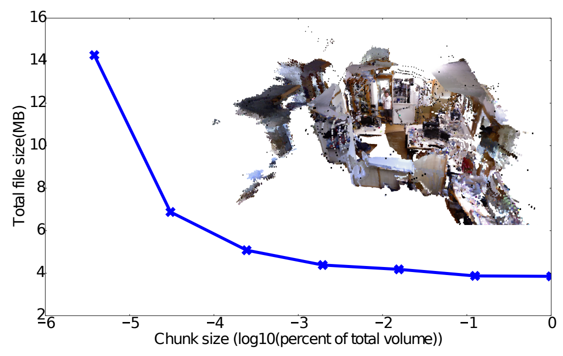

Fig. 5. Compression of the fr1/room dataset. Octree decomposition, JPEG 2000 compression. The $\log_{10}$ of the chunk/voxel volume relative to the scene's bounding box volume is plotted vs. the file size. The bounding box volume was $(16.68\,\mathrm{m})^3$. The smaller the chunk size, the finer the local granularity of the compression and decompression. Only below $10^{-3.5}\%$ volume, the increase in file size becomes more evident.

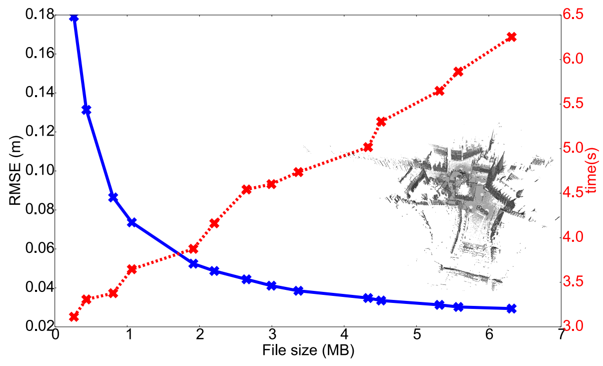

Fig. 6. Compression results of the Thermobremen dataset, consisting of 40.7 million points (466 MB), without colors, JPEG compression. MB vs. RMSE (blue), MB vs. time (red, dotted line), voxel decomposition

online robotics applications. Furthermore it supports locally decompressing parts of the data as they are required for navigation or visualization purposes.

References

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *Visualization and Computer Graphics, IEEE Transactions on*, 9(1):3–15, 2003.

[2] C. Christopoulos, A. Skodras, and T. Ebrahimi. The jpeg2000 still image coding system: an overview. *Consumer Electronics, IEEE Transactions on*, 46(4):1103–1127, 2000.

[3] A. Cohen, I. Daubechies, and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 45(5):485–560, 1992.

[4] J. Digne, R. Chaine, S. Valette, et al. Self-similarity for accurate compression of point sampled surfaces. In *Computer Graphics Forum*, volume 33, pages 155–164, 2014.

[5] T. Golla, C. Schwartz, and R. Klein. Towards Efficient Online Compression of Incrementally Acquired Point Clouds. In J. Bender, A. Kuijper, T. von Landesberger, H. Theisel, and P. Urban, editors, *Vision, Modeling & Visualization*, pages 17–22. The Eurographics Association, 2014.

[6] X. Gu, S. J. Gortler, and H. Hoppe. Geometry images. *ACM Transactions on Graphics (TOG)*, 21(3):355–361, 2002.

[7] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

TABLE I

EVALUATION ON THE FR1/ROOM DATASET, WITH A COMPARISON TO THE NUMBERS REPORTED IN [26]. THE DATASET CONSISTS OF 315.5 MILLION POINTS AND HAS A RAW INPUT STREAM SIZE OF 2.7 GB. s_c DENOTES THE COMPRESSED FILE SIZE, t_c AND t_d THE COMPRESSION RESP. DECOMPRESSION TIME, $RMSE$ IS THE SYMMETRIC NEAREST NEIGHBOR RMSE. *Voxel* AND *Octree* DENOTE THE DECOMPOSITION METHOD, *JPEG*, RESP. *JPEG 2000* THE MAP COMPRESSION METHOD. p/t_c DENOTES THE COMPRESSION SPEED IN POINTS PER SECOND.

Method	s_c	RMSE (D/RGB)	t_c	t_d	p/t_c
OctoMap	45 MB	0.006 m / 39.6	120 s	n.a.	$2.63 \cdot 10^6$
SCSM	8.1 MB	0.005 m / 29.9	1860 s	n.a.	$0.17 \cdot 10^6$
HSCSM	7.2 MB	0.005 m / 30.0	2160 s	n.a.	$0.14 \cdot 10^6$
Ours, Voxel, JPEG	7.07 MB	0.005 m / 26.6	43 s	2.7 s	$7.33 \cdot 10^6$
Ours, Octree, JPEG 2000	2.97 MB	0.005 m / 27.2	71.5 s	4.7 s	$4.41 \cdot 10^6$

TABLE II

EVALUATION ON DIFFERENT DATASETS. s_c DENOTES THE COMPRESSED FILE SIZE, t_c AND t_d THE COMPRESSION RESP. DECOMPRESSION TIME, $RMSE$ IS THE SYMMETRIC NEAREST NEIGHBOR RMSE. *V(oxel)* AND *O(ctree)* DENOTE THE DECOMPOSITION METHOD, *JPEG*, RESP. *J2000* THE MAP COMPRESSION METHOD. p/t_c DENOTES THE COMPRESSION SPEED IN POINTS PER SECOND. *Thb* IS THE THERMOBREMEN, *Ff* AND *Ff2* THE FRANKENFORST DATASETS. *w c.* AND *w/o c.* DENOTE WITH RESP. WITHOUT COLORS.

Dataset	Method	Input size	Points	s_c	RMSE (D/RGB)	t_c	t_d	p/t_c
Thb, w/o colors	V, J2000	466 MB	$40.7 \cdot 10^6$	1.95 MB	0.068 m / –	5 s	1 s	$8.14 \cdot 10^6$
Ff, w/o colors	O, JPEG	499 MB	$43.6 \cdot 10^6$	0.40 MB	0.017 m / –	5.7 s	1 s	$7.65 \cdot 10^6$
Ff2, w/o colors	O, JPEG	3001 MB	$262.2 \cdot 10^6$	9.0 MB	0.028 m / –	24.9 s	2.33 s	$10.5 \cdot 10^6$
fr1/room, 1st scan, w c.	V, JPEG	3.4 MB	$0.22 \cdot 10^6$	0.11 MB	0.0024 m / 9.64	0.07 s	0.03 s	$3.15 \cdot 10^6$
Thb, 1st scan, w/o c.	V, JPEG	6.08 MB	$0.52 \cdot 10^6$	0.15 MB	0.024 m / –	0.15 s	0.095 s	$3.46 \cdot 10^6$
Thb, 1st scan, w c.	V, JPEG	6.08 MB	$0.52 \cdot 10^6$	0.22 MB	0.024 m / 1.19	0.18 s	0.095 s	$2.88 \cdot 10^6$

[8] P. G. Howard, F. Kossentini, B. Martins, S. Forchhammer, and W. J. Rucklidge. The emerging jbig2 standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 8(7):838–848, 1998.

[9] Y. Huang, J. Peng, C. C. Kuo, and M. Gopi. A generic scheme for progressive point cloud coding. *IEEE Trans. Visual. Comput. Graph.*, 14(2):440–453, Mar. 2008.

[10] Y. Huang, J. Peng, C.-C. J. Kuo, and M. Gopi. Octree-based progressive geometry coding of point clouds. In *Proceedings of the 3rd Eurographics/IEEE VGTC conference on Point-Based Graphics*, pages 103–110. Eurographics Association, 2006.

[11] E. Hubo, T. Mertens, T. Haber, and P. Bekaert. The quantized kd-tree: Efficient ray tracing of compressed point clouds. In *Proc. IEEE Symp. on Interactive Ray Tracing*, pages 105–113, Sept 2006.

[12] E. Hubo, T. Mertens, T. Haber, and P. Bekaert. Self-similarity-based compression of point clouds, with application to ray tracing. In *Proc. Eurographics Symp. on Point-Based Graphics*, pages 129–137, 2007.

[13] E. Hubo, T. Mertens, T. Haber, and P. Bekaert. Self-similarity based compression of point set surfaces with application to ray tracing. *Computers & Graphics*, 32(2):221–234, 2008.

[14] M. Isenburg. Laszip: lossless compression of lidar data. *Photogrammetric Engineering and Remote Sensing*, 79(2):209–217, 2013.

[15] A. Kalaiah and A. Varshney. Statistical geometry representation for efficient transmission and rendering. *ACM Trans. Graph.*, 24(2):348–373, 2005.

[16] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach. Real-time compression of point cloud streams. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 778–785. IEEE, 2012.

[17] J. Krüger, J. Schneider, and R. Westermann. Duodecim - a structure for point scan compression and rendering. In *Proceedings of the Symposium on Point-Based Graphics 2005*, pages 99–146, 2005.

[18] S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, Mar 1982.

[19] T. Ochotta and D. Saupe. Compression of point-based 3d models by shape-adaptive wavelet coding of multi-heightfields. In *Proc. Eurographics Symp. on Point-Based Graphics*, pages 103–112, 2004.

[20] T. Ochotta and D. Saupe. Image-based surface compression. In *Computer graphics forum*, volume 27, pages 1647–1663. Wiley Online Library, 2008.

[21] F. Ono, W. Rucklidge, R. Arps, and C. Constantinescu. Jbig2-the ultimate bi-level image coding standard. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, volume 1, pages 140–143. IEEE, 2000.

[22] M. Pauly and M. Gross. Spectral processing of point-sampled geometry. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 379–386. ACM, 2001.

[23] I. Pavlov. 7-zip. http://7-zip.org, 2015. accessed on February, 28th, 2015.

[24] R. Rubinstein, M. Zibulevsky, and M. Elad. Efficient implementation of the k-svd algorithm using batch orthogonal matching pursuit. *CS Technion*, page 40, 2008.

[25] M. Ruhnke, L. Bo, D. Fox, and W. Burgard. Compact rgbd surface models based on sparse coding. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 1429–1435, 2013.

[26] M. Ruhnke, L. Bo, D. Fox, and W. Burgard. Hierarchical sparse coded surface models. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 6238 – 6243, 2014.

[27] R. Schnabel and R. Klein. Octree-based point cloud compression. In *Proc. IEEE/Eurographics Symp. on Point-Based Graphics*, pages 111–120, 2006.

[28] R. Schnabel, S. Möser, and R. Klein. A parallelly decodeable compression scheme for efficient point-cloud rendering. In *Proc. Symp. on Point-Based Graphics*, pages 214–226, Sept. 2007.

[29] R. Schnabel, S. Möser, and R. Klein. Fast vector quantization for efficient rendering of compressed point-clouds. *Computers and Graphics*, 32(2):246–259, Apr. 2008.

[30] A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. *Signal Processing Magazine, IEEE*, 18(5):36–58, 2001.

[31] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012.

[32] M. Waschbüsch, M. Gross, F. Eberhard, E. Lamboray, and S. Würmlin. Progressive compression of point-sampled models. In *Proc. Eurographics Symp. on Point-Based Graphics*, pages 95–102, 2004.

[33] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *Information Theory, IEEE Transactions on*, 23(3):337–343, May 1977.