Vision, Modeling, and Visualization (2014)
J. Bender, A. Kuijper, T. von Landesberger, H. Theisel and P. Urban (Eds.)

Towards Efficient Online Compression of Incrementally Acquired Point Clouds

Tim Golla, Christopher Schwartz and Reinhard Klein†

Institute of Computer Science II, University of Bonn

Abstract

We present a framework for the online compression of incrementally acquired point cloud data. For this, we extend an existing vector quantization-based offline point cloud compression algorithm to handle the challenges that arise from the envisioned online scenario. In particular, we learn a codebook in advance from training data and replace a computationally demanding part of the algorithm with a faster alternative. We show that the compression ratios and reconstruction quality are comparable to the offline version while the speed is sufficiently improved. Furthermore, we investigate how well codebooks that are generated from different amounts of training data generalize to larger sets of point cloud data.

Categories and Subject Descriptors (according to ACM CCS): E.4 [Coding and Information Theory]: Data compaction and compression—I.3.3 [Computer Graphics]: Picture/Image Generation—Digitizing and scanning I.4.1 [Image processing and computer vision]: Digitizing and Image Capture—Quantization

1. Introduction

Through advances in the field of robotics and in scanner technology, such as the Kinect, incrementally acquired point clouds are becoming increasingly important. We consider the scenario of a robot equipped with a 3D scanning device like a laser scanner traversing an urban scenario, incrementally building a 3D map of its surroundings. Thus, the robot continuously acquires point cloud data that is either stored on the robot itself or transmitted to a base station, leading to a huge amount of data that has to be stored for future access. As a consequence, a framework for the compression of point clouds with the following properties is required:

- Support for incrementally acquired data (from different scan positions)
- Level of detail (LOD) support for fast visualization and collision detection
- Fast compression and decompression capability
- Good compression ratios

Since most publications in this field focus on offline compression for static point clouds, to the best of our knowledge there is no algorithm that meets all of these basic requirements of the application scenario in mind. A promising approach is described in [SMK08]. It is based on a decomposition of the point cloud into clusters of which each is approximated by a geometric primitive, e.g. a plane. Fine-scale geometry is encoded as height fields over these primitives, which are compressed progressively via image pyramids and vector quantization.

The approach partially fulfills our requirements: it features fast decompression with a selected level of detail. The reported compression ratios are state of the art. The geometric shape primitive approximation is well-suited for our scenario, which consists primarily of man-made objects. However, the approach in its original form does not support incrementally acquired data. Naïvely, one could attempt to update the compressed representation by unpacking the data, adding the newly acquired points and compressing the result again. This has the disadvantage of leading to an unacceptable workload, as the amount of data to be compressed steadily increases. Also, since the compression is lossy, the process of iterated decompression and compression of the reconstructed point clouds introduces degradation of the data. An alternative would be to apply the unmodified offline compression algorithm to each incremental point cloud. However, for the vector quantization a codebook is required, which would have to be generated for each newly acquired point cloud. This would be computationally expensive.

† {golla, schwartz, rk}@cs.uni-bonn.de

DOI: 10.2312/vmv.20141271

We argue that in many scenarios prior assumptions about the structure of the data can be made. Thus, we introduce a codebook learning phase on representative sample data as a preprocessing step. We identified the original moving least squares-based height field generation procedure as another speed bottleneck of the algorithm. In order to remedy the aforementioned shortcomings, we extend the original approach by:

- Introducing a codebook learning phase as a preprocessing step
- Speedup of the height field generation algorithm by using GPU-based Delaunay triangulation and rasterization instead of moving least squares (MLS)

We evaluate which effect dictionaries generated from training data of different sizes and our suggested height field generation have on the compression ratio and reconstruction quality. Our evaluation indicates that the quality of the reconstructed results from iterative compression is comparable to those of the original algorithm executed on the integrated data.

2. Related work

A large number of offline point cloud compression algorithms have been published. Here, we give a short overview of the most related publications.

Many algorithms are based on space partitioning using octrees. Botsch et al. [BWK02] sample the characteristic function of the surface into an octree. Only the subdivisions of non-empty cells are stored via byte codes. They achieve data rates of 2 bits per leaf node. In general, more leaf nodes than original points have to be stored, resulting in overall higher bitrates. Gumhold et al. [GKIS05] use a prediction tree which is arithmetically encoded in a sequential way. Schnabel and Klein [SK06] build an octree for the point cloud whose occupancy is also arithmetically encoded. In order to reduce entropy, they predict the lower octree levels from the higher ones by intersecting a least-squares-fitted plane. A similar idea is presented in [HPKG06] and [HPKG08]. Merry et al. [MMG06] use a spanning tree to represent and predict the points. As in the other algorithms, this tree is being compressed.
An approach that is not tree-based is [WGE*04]. They perform a multiresolution decomposition of the point cloud, resulting in a spatial hierarchy. On this structure they perform a three stage coding: First differential coding, then zerotree coding and finally arithmetic coding.

In contrast to the methods above, Schnabel et al. [SMK08] proposed to segment the points by the means of "compatible" primitive shapes rather than a space partition. We summarize their approach in more detail in Section 3.
Recently, similar ideas were explored in [RBFB13,RBFB14] and [DCV14]. Instead of parameterizing the points on extracted shapes, these approaches fit oriented local surface patches by collecting data in the neighborhood of occupied cells on a regular grid. Instead of vector quantization K-SVD is used to find an overcomplete codebook and represent the data as linear combinations of codewords.

Little has been published on the subject of non-static point cloud compression. One recent example is [KBR*12]. They focus on streams of point clouds recorded by a static depth camera such as the Kinect. They represent each frame by an occupancy octree and then perform an XOR operation on each of a frame's octree's node representations to the respective ones of the previous frame. The XOR operation leads to a great entropy reduction if large parts of the point clouds are static or overlapping, yielding good compression results via arithmetic encoding. However, in the envisioned application, scans taken at different positions with possibly little overlap have to be integrated, rendering this approach inapt.

3. Our compression framework

As described in section 1, our framework is based on [SMK08]. We therefore briefly summarize the original method and then describe our extensions.

Initially, geometric primitives like planes or spheres are detected in the input point cloud by the RANSAC algorithm described in [SWK07]. Each point of the original cloud is by this means either associated to a geometric primitive or discarded. Usually, the number of discarded points is very low. The point cloud can already be very coarsely approximated by these primitives. In order to account for fine-scale geometry, a height field, i.e. a bitmap image containing elevation data, is constructed over each primitive. The bitmap is filled by intersecting the vector originating in each pixel and pointing in the primitive's normal direction with the moving least squares surface defined by the primitive's associated points. In order to account for holes in the original data, an additional occupancy bitmap is constructed. This bitmap contains a 1 for an occupied height field pixel and a 0 for holes. Each height field is represented as a Laplacian pyramid [AB81]. Such a pyramid consists of a predefined number $N+1$ of images and is obtained by first constructing a Gaussian pyramid. The Gaussian pyramid's lowermost level g_0 is the original height field. Level g_{i+1} is obtained from level g_i by convolving it with a Gaussian and downsampling. The Laplacian pyramid consist of the difference images of the Gaussian pyramid's levels. That is, level L_i of the Laplacian pyramid is $g_i - ug_{i+1}$, where u is an upsampling operation and $L_N = g_N$. This pyramid representation allows for level of detail reconstruction: if all levels of the Laplacian pyramid are combined, the original height field would be reconstructed. If only the coarsest or a subset of the levels is used, a downsampled version is obtained. Schnabel et al. [SMK08] replaced the Gaussian by the CDF$\frac{5}{3}$ wavelet [CDF92], due to its advantages concerning a GPU implementation. Finally, each level of all the Laplacian pyramids is vector-quantized. For this, all images are split into

equally-sized tiles of a size of e.g. 16×16 pixels. For these tiles, a dictionary with representative images is built using Lloyd's algorithm. Compression is performed by replacing the original image tiles with dictionary entry indices. Since all image tiles are known to the offline algorithm, a nearly optimal codebook can be generated.

To understand the consequences of our additions, it is important to note that the original offline algorithm performs a lossy compression. Points that are not assigned to a primitive during RANSAC shape detection are considered to be outliers and are discarded in all consecutive steps. Moreover, the representation of the points in a height field over the surface requires a re-parameterization from the likely irregular distribution of points into a regular image grid. The height field's resolution is chosen in such a way that it approximately matches the point cloud's resolution. Ideally, there should be only one point projected into each height field's pixel to avoid data loss. On the other hand, there should not be too many "empty" pixels, as this would lead to an inferior compression ratio. Nonetheless, after this step positions cannot be restored exactly any more. Finally, the compression of the image pyramid via vector quantization is a lossy method as well. Tiles of the images are represented by a common substitute from a smaller codebook. Here, the reconstruction quality depends on the choice of that codebook. Also note that the codebook generation step has to be performed on each dataset anew.

We argue that in certain applications, such as the envisioned urban exploration scenario, a reasonable codebook can be learned from selected training data. Our evaluation in Section 4 indicates that this is the case. Therefore, we implement an offline learning phase. On a given training point cloud, we detect shapes and create height fields for these shapes. Similar to Schnabel et al. we then use Lloyd's algorithm to obtain a vector quantization for this data satisfying a given maximum root mean square error (RMSE). The resulting codewords form the precomputed codebook. This training point cloud is usually chosen as a subset of the original point cloud, in a real-world application, this would be the first few initial scans. During online compression, shape detection and height field generation are performed on the incrementally acquired data. These height fields are vector quantized by expressing them using the precomputed codewords with minimal L^2 distance. An overview of the method is given in figure 1.

The shape fitting and the assignment of codewords can be performed reasonably fast. However, we found the moving least squares (MLS) interpolation used for the height field generation to be a bottleneck. We instead propose to compute the 2D Delaunay triangulation (DT) of the points projected into the parameterization domain of the shape. The points' x and y-coordinates are in the domain's coordinate system, while the z-coordinate can be considered the height value. We identify the 3D points with their projected 2D

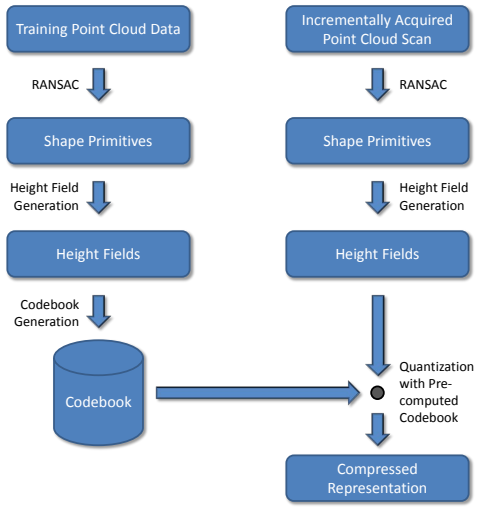

Figure 1: *Overview of the most important steps of the compression framework.*

counterparts. and subsequently assign the z-coordinates as color values to the 2D points. We employ the fast GPU algorithm by Rong et al. [RTC*08] to compute the Delaunay triangulation of the 2D points. Using the triangulation, linearly interpolated height values can efficiently be rasterized via OpenGL into the height field image as the points' colors are the original height values. Using GPU off-screen rendering, each height field pixel is assigned a color, equaling the interpolated height value. In our case, the linear interpolation over the triangles provides similar quality to MLS, because the resolution of the height fields has been chosen to have roughly one point per pixel. If more than one point of the original would fall into one height field pixel, the Delaunay triangulation would produce several triangles "belonging" to this pixel. In this case it would on of the triangle is selected quasi randomly for the rasterization, i.e. for the height value calculation. This can lead to a result differing severely from the result an MLS interpolation would deliver. In our case though, the number of triangles per pixel is usually very small, leading to a result similar to that of the MLS method.

4. Evaluation

For our evaluation, we use the commonly employed quantities bits per point (BPP), root mean square error (RMSE) and peak signal to noise ratio (PSNR). The BPP are the compressed file size divided by the number of points. The PSNR is defined as: $PSNR = 20\log_{10}\left(\frac{d}{r}\right)$, where d is the diagonal of the point cloud's bounding box and r is the RMSE.

All experiments were carried out using a machine with two Intel Xeon E5645 CPUs (at 2.4GHz) and a NVIDIA

Dataset	method	time	BPP	PSNR
David	MLS	455s	1.97	83.2dB
	DT	177s	1.79	82.9dB
Bremen	MLS	2488s	2.71	81.3dB
	DT	153s	2.59	82.6dB

Table 1: *Compression results obtained from MLS and DT based height field resampling.*

GeForce GTX 570 GPU. While parts of the compression are well suited for parallel execution, others were performed on a single CPU core. The reported timings are wall-clock times.

We first evaluated our method on the David point cloud from the Digital Michelangelo project, since it is widely-used for comparisons in the literature. Using the unmodified algorithm described in [SMK08] we were able to reproduce the reported compression rate of 1.97 BPP for a peak signal to noise ratio (PSNR) of 83.5 on the David point cloud. We evaluated the iterative compression on the David point cloud by subdividing it into 56 subsets. We found that for this point cloud the choice of the subset for dictionary learning had an impact on the compression results: when using a subset extracted from the statue's pedestal, we achieved a PSNR of 83.1757 at a bitrate of 2.94BPP. When using a subset of the head for dictionary learning, we achieved a bitrate of 2.26BPP at a PSNR of 83.2514.

To assess the performance of the approach on scans from urban environments, we tested it on the Bremen point cloud by Dorit Borrmann and Andreas Nüchter, consisting of terrestrial laser scans taken in the city center – see Figure 2. We used the first 27 scans of the dataset, which comprise full 360° views taken from three different scanner positions. We observed that on this data, the achieved compression ratios for a similar PSNR were slightly lower than for the David point cloud (see Table 1). This can be explained by the nature of the dataset. Compared to the David, the Bremen point clouds possess some more challenging properties. The sampling density in a single scan varies considerably, depending on distance and angle of the scanned surface. This is especially true for the ground, which is sampled extremely sparsely outside the direct vicinity of the scanner. Additionally, many of the scanned surfaces contain significant holes, e.g. windows in building walls, and the dataset shows a lot of thin structures, such as trees or overland-powerlines, which are comprised of only a handful of points. While tackling these issues presents an interesting avenue for future research, it was outside the scope of this paper.

We determined the change in performance by the proposed GPU based interpolation. In Table 1 we compare the compression results to those obtained using the originally described MLS. The height field generation part was sped up by a factor of 16.3 on the Bremen dataset. Using Delaunay triangulation yields in this case a compression ratio

	iterative	static
# points	19,451,257	
scene diameter	67.92m	
# point clouds	27	1
total compression time	502s	903s
# shapes	1720	1590
# discarded points	0.61%	0.5%
compression ratio	1:34	1:31
BPP (incl. codebook)	2.84	3.1
RMSE	4.86mm	5.37mm
decompression time	10.5s	

Table 2: *Performance of the iterative and static compression. In this experiment we used the optimal codebook from the static offline compression as input for the iterative compression. In both cases the faster DT interpolation was employed. Using a common precomputed codebook saved an additional* 242 *seconds in the iterative case.*

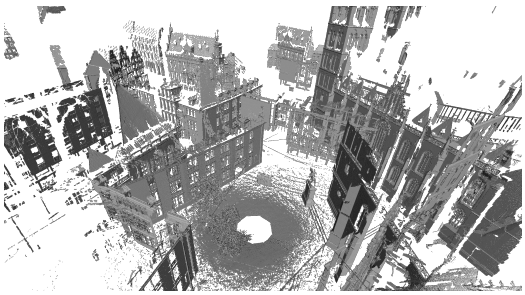

Figure 2: *Visualization of a subset of the Bremen point cloud*

and reconstruction quality similar to or better than MLS. In all consecutive experiments we therefore employ the proposed Delaunay triangulation. This accelerates the overall compression algorithm by a factor of about 5.

In the next experiment, we tested the influence of using iterative compression only, but maintaining the "optimal" codebook. For this, we first used the static compression on a point cloud obtained from merging the first 27 individual scans of the Bremen dataset. We then used the "optimal" codebook that was computed by the static compression as input to the iterative compression. During iterative compression all 27 point clouds, each containing about 720,000 points, were processed individually and added to the compressed representation. Table 2 lists the detailed results. It is noteworthy that the iterative compression finds a similar amount of shapes and outliers. This indicates that there is little overlap in the single scans. In summary, the iterative compression manages to achieve a better compression ratio with a higher PSNR at lower compression times. Average compression times per single scan were 18.6 seconds.

Finally, we investigate the influence of the precomputed codebook on the Bremen point cloud. For this, we learn the

n training scans	1	2	3	4	5	6
RMSE [mm]	4.1	4.7	4.1	5.4	4.2	3.9
BPP	2.7	2.6	2.8	2.8	2.8	2.7
time [s]	536	530	563	545	563	535

Table 3: *Performance of the iterative compression with respect to codebook learning evaluated on the Bremen dataset. Here, we selected n random scans from all available. We extract shapes and height fields from the n point clouds to learn a codebook. This is subsequently used to incrementally compress all* 27 *scans.*

Level of detail	0	1	2	3	4	5
RMSE [mm]	10.1	9.6	9.6	6.2	5	3.8
time [s]	0.01	0.05	0.17	0.53	1.8	6.2

Table 4: *Timing and reconstruction error for decompression of the compressed dataset at different levels of detail.*

codebook from a random selection of the input scans. Each experiment was carried out multiple times and we report the average values. As the results in Table 3 illustrate, even a single scan is sufficient to learn a descriptive set of codewords for the scenario. For a comparison see also Figure 3, which depicts the reconstruction results from the compressed representation of the static and iterative compression, where the former used a codebook learned from all scans and the latter a codebook learned from only 2 scans. Both used Delaunay triangulation for the height field computation. We conclude that for urban scenarios, the proposed incremental method with a precomputed codebook is robustly applicable.

5. Conclusion and future work

We presented a framework for the efficient online compression of incrementally acquired point cloud data. For this we modified a state-of-the-art offline point cloud compression algorithm to fulfill our requirements.

We were able to compress incrementally added point clouds while achieving a compression ratio and reconstruction quality comparable to those of offline algorithms.

The employed compressed representation allows reconstruction at different levels of detail. Lower levels offer real-time access with still acceptable reconstruction errors (see table 4). In the example application scenario of urban mapping with a robot, this facilitates the direct use of the compressed data for visualization purposes as well as for collision detection computations.

The algorithm is primarily well suited for input data that can be well represented by geometric primitives, i.e. man-made objects. The compression rate and reconstruction quality depends on the employed vector quantization dictionary. That is, the data chosen for the dictionary generation – usually the initial scan(s) – has to be chosen deliberately. It should be representative for the data to be acquired in the future. In our experiments, we found that for the Bremen dataset, most subsets of the point cloud provided similar compression results. In the case of the David point cloud, it was necessary to include parts of the head for the codebook generation in order to achieve compression rates comparable to those of the offline algorithm. Furthermore, our framework requires a GPU and a reasonably fast CPU and can thus only be used on autonomous robots with sufficient computing performance or on base stations, to which the data is being sent continuously. In the future we intend to work on further algorithmic optimizations and a comparison of our Delaunay triangulation-based height field generation to an MLS-based height field computation implemented on the GPU.

In summary we conclude that using a pre-learned codebook for vector quantization-based compression of incrementally acquired point clouds is possible and delivers compression results comparable to those of the offline algorithm.

References

[AB81] ADELSON E. H., BURT P. J.: Image data compression with the laplacian pyramid. *Proceedings of the Pattern Recognition and Information Processing Conference* (1981), 218–223. 2

[BWK02] BOTSCH M., WIRATANAYA A., KOBBELT L.: Efficient high quality rendering of point sampled geometry. In *In: EGRW '02: proceedings of the 13th eurographics workshop on rendering* (2002), pp. 53–64. 2

[CDF92] COHEN A., DAUBECHIES I., FEAUVEAU J.-C.: Biorthogonal bases of compactly supported wavelets. *Communications on pure and applied mathematics 45*, 5 (1992), 485–560. 2

[DCV14] DIGNE J., CHAINE R., VALETTE S.: Self-similarity for accurate compression of point sampled surfaces. *Computer Graphics Forum 33*, 2 (2014), 155–164. 2

[GKIS05] GUMHOLD S., KAMI Z., ISENBURG M., SEIDEL H.-P.: Predictive point-cloud compression. In *ACM SIGGRAPH 2005 Sketches* (2005), ACM, p. 137. 2

[HPKG06] HUANG Y., PENG J., KUO C.-C. J., GOPI M.: Octree-based progressive geometry coding of point clouds. In *Proceedings of the 3rd Eurographics/IEEE VGTC conference on Point-Based Graphics* (2006), Eurographics Association, pp. 103–110. 2

[HPKG08] HUANG Y., PENG J., KUO C.-C., GOPI M.: A generic scheme for progressive point cloud coding. *Visualization and Computer Graphics, IEEE Transactions on 14*, 2 (2008), 440–453. 2

[KBR*12] KAMMERL J., BLODOW N., RUSU R. B., GEDIKLI S., BEETZ M., STEINBACH E.: Real-time compression of point cloud streams. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on* (2012), IEEE, pp. 778–785. 2

[MMG06] MERRY B., MARAIS P., GAIN J.: Compression of dense and regular point clouds. *Computer Graphics Forum 25*, 4 (2006), 709–716. 2

[RBFB13] RUHNKE M., BO L., FOX D., BURGARD W.: Compact rgbd surface models based on sparse coding. In *Proc. of the National Conference on Artificial Intelligence (AAAI)* (2013). 2

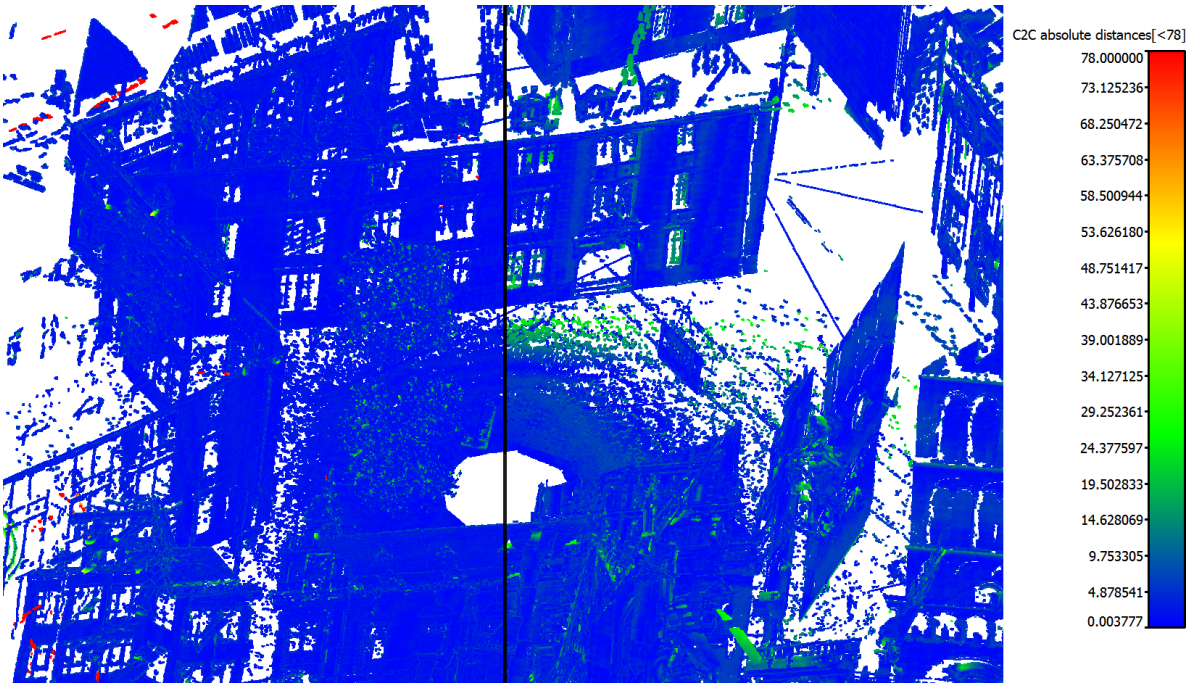

Figure 3: *The Bremen point cloud, reconstructed from the compressed representation. Colors denote each point's Hausdorff distance to the original point cloud [mm]. Left: iterative compression with codebook learned from two scans. Right: offline compression with codebook learned from all scans. Both used Delaunay triangulation for the height field computation. While the average error is similar, the distribution of errors differs. The seemingly higher errors visible in the reconstruction from the offline method, which might appear unintuitive at first glance, can be explained by the very inhomogeneous sampling of the laser scan. The globally learned dictionary contains more codewords to represent the sparsely sampled regions, whereas the online dictionary was in this case trained on the densely sampled regions and can therefore better represent such areas. In the case of the locally learned dictionary, the error is concentrated in a few small regions.*

[RBFB14] RUHNKE M., BO L., FOX D., BURGARD W.: Hierarchical sparse coded surface models. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)* (2014). to appear. 2

[RTC*08] RONG G., TAN T.-S., CAO T.-T., ET AL.: Computing two-dimensional delaunay triangulation using graphics hardware. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games* (2008), ACM, pp. 89–97. 3

[SK06] SCHNABEL R., KLEIN R.: Octree-based point-cloud compression. In *SPBG* (2006), pp. 111–120. 2

[SMK08] SCHNABEL R., MÖSER S., KLEIN R.: Fast vector quantization for efficient rendering of compressed point-clouds. *Computers and Graphics 32*, 2 (Apr. 2008), 246–259. 1, 2, 4

[SWK07] SCHNABEL R., WAHL R., KLEIN R.: Efficient ransac for point-cloud shape detection. *Computer Graphics Forum 26*, 2 (June 2007), 214–226. 2

[WGE*04] WASCHBÜSCH M., GROSS M., EBERHARD F., LAMBORAY E., WÜRMLIN S.: Progressive compression of point-sampled models. In *Proceedings of the First Eurographics conference on Point-Based Graphics* (2004), Eurographics Association, pp. 95–103. 2

Acknowledgments

This work was partially funded by the German Research Foundation (DFG) under grant KL 1142/9-1 (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 Bremen dataset is recorded by Dorit Borrmann and Andreas Nüchter from Jacobs University Bremen gGmbH, Germany. http://kos.informatik.uni-osnabrueck.de/3Dscans/