COMPRESSION OF 3-D POINT CLOUDS USING HIERARCHICAL PATCH FITTING

Robert A. Cohen, Maja Krivokuća[*]*, Chen Feng, Yuichi Taguchi, Hideaki Ochimizu, Dong Tian, Anthony Vetro*

Mitsubishi Electric Research Laboratories
201 Broadway, Cambridge, MA 02139, USA

ABSTRACT

For applications such as virtual reality and mobile mapping, point clouds are an effective means for representing 3-D environments. The need for compressing such data is rapidly increasing, given the widespread use and precision of these systems. This paper presents a method for compressing organized point clouds. 3-D point cloud data is mapped to a 2-D organizational grid, where each element on the grid is associated with a point in 3-D space and its corresponding attributes. The data on the 2-D grid is hierarchically partitioned, and a Bézier patch is fit to the 3-D coordinates associated with each partition. Residual values are quantized and signaled along with data necessary to reconstruct the patch hierarchy in the decoder. We show how this method can be used to process point clouds captured by a mobile-mapping system, in which laser-scanned point locations are organized and compressed. The performance of the patch-fitting codec exceeds or is comparable to that of an octree-based codec.

***Index Terms*—** point cloud compression, patch fitting, mobile mapping systems

1. INTRODUCTION

With the widespread use of camera and laser-based scanners for capturing 3-D environments, compression has become a critical element in systems that store and transmit data generated by these scanners. Representing these data as point clouds is well suited for applications such as virtual reality, mobile mapping systems (MMS), architecture, and geographic information systems. For these applications, a point cloud comprises a set of coordinates in 3-D space. Each point can be assigned a set of attributes, such as color, reflectance, normal vectors, and more. These points and associated attributes can be stored, transmitted, and rendered in a variety of ways, to allow users to interact with the data in meaningful ways.

If a point cloud contains connectivity information among points, it can be compressed using methods that have been developed to compress polygonal meshes, such as those surveyed in [1]. Point clouds captured using cameras or scanners, however, typically do not contain mesh information, unless it is added as part of a post-processing step. For complex point clouds, adding meshes can be a time-consuming process. For compressing point clouds that do not contain mesh information, several methods have been developed. In [2] and [3], octrees were used to partition 3-D point clouds into blocks of points that can be efficiently compressed. The method of [3] was implemented using the open-source Point Cloud Library [4]. This work was extended in [5], where JPEG was used to compress color attributes. Octrees were also used in [6], where graph transforms were used to compress voxelized point clouds that were created using the method described in [7]. Graph transforms were also used in [8] to compress compacted blocks of points. The octree-based methods were extended in [9], in which residuals from plane projections were signaled as an enhancement layer to a coarse octree coding.

Cameras and LiDAR sensors have long been used to capture large-scale outdoor environments such as roads and city infrastructures. Some commonly used representation formats for these kind of LiDAR and related data include the ASTM E57 [10] and LAS [11] file formats. For lossless compression, sequentially captured LiDAR points stored in the LAS format can be compressed using LASzip [12]. Near-lossless and lossy compression are also useful for compressing large-scale data. For example, a progressive or scalable coding system can be used to facilitate the streaming of point clouds to remote terminals, in which the quality or precision of the rendered point cloud can increase as a user zooms in, or when a region of the point cloud is dynamically updated. A discussion of some of the requirements related to compressing point clouds for 3-D dynamic maps is presented in [13]. For compressing static, dynamic, and dynamically acquired point clouds, standardization efforts are currently underway in MPEG. The associated use cases and requirements are described in [14] and [15], and a call for proposals for point cloud compression was issued in January 2017 [16].

In this paper, we present a method that uses hierarchical patch fitting to compress organized point clouds. For point clouds that are captured using LiDAR sensors, the scan-order information can be used to associate each point in 3-D space with a location index on a 2-D grid. By organizing the data in this way, locations that are next to each other on the 2-D grid are likely to be part of the same surface scanned in 3-D space. This organization eliminates the need in the subsequent patch-fitting process to perform nearest-neighbor searches or to generate spatial connectivity structures. A hierarchical partitioning of the 2-D grid is used to select points in 3-D space to which patch models are fit. Compression is achieved by coding residuals between each patch model and its corresponding set of input points. In Section 2, we describe how the point cloud is organized and how hierarchical patch models are generated. Section 3 describes how the data is compressed, and Section 4 presents experimental results. Conclusions are given in Section 5.

2. HIERARCHICAL PATCH FITTING OF ORGANIZED POINT CLOUDS

3-D point clouds generated from LiDAR-captured data streams can be organized onto a 2-D grid. This section describes how the point cloud is organized and how the hierarchical patch models are generated.

2.1. Constructing organized point clouds

In a dynamic-acquisition MMS system, a LiDAR sensor is mounted to the top of a vehicle. The sensor contains a laser which rotates at

[*]Maja Krivokuća is now with 8i, Wellington, NZ

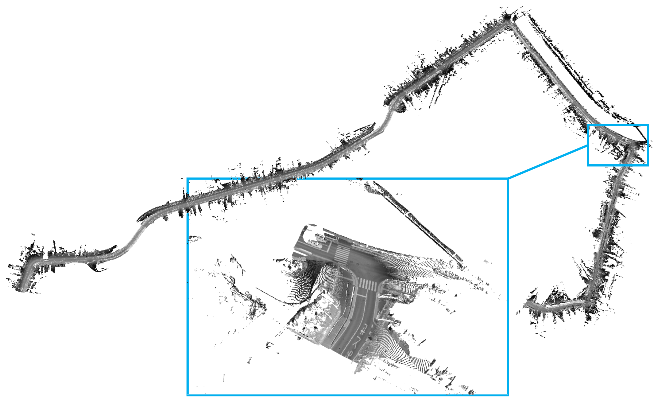


Fig. 1: Point cloud captured from a mobile mapping system

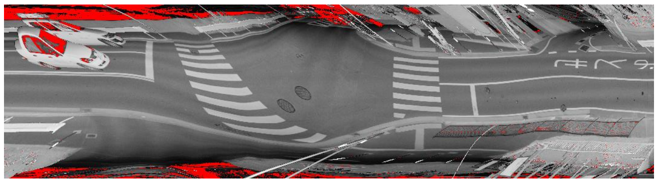


Fig. 2: Organized grid of attributes for road junction point

a frequency of f_r rev/s. Light reflected from objects is sampled uniformly at a rate of f_s Hz. The sensor is angled toward the ground in order to detect the road in front of the vehicle and objects or structures to the side of the road. Additional sensors can be mounted to capture other directions. The angular field of view of the scanner is A degrees. Each rotation of the sensor can therefore detect $(f_s/f_r) \cdot (A/360)$ samples. For a typical LiDAR scanner having a field of view $A = 190^\circ$, $f_s = 54$ kHz and $f_r = 100$ Hz, the maximum number of samples per scan is 285. For the MMS [17] used in this paper, up to 286 samples were captured per scan. By combining the LiDAR sensor data with the position and orientation of the vehicle obtained through a GPS system and inertial measurement unit, the (x, y, z) position of each captured point in a common coordinate system can be computed. Each scan thus generates 286 (x, y, z) points of the point cloud. Each point is associated with attributes such as the intensity of the reflected light, or color data fused from cameras that are also mounted to the vehicle. A point cloud with intensity attributes captured from a vehicle driving several hundred meters along different roads is shown in Fig. 1. The inset shows a magnified portion of a junction.

To organize the point cloud, we associate each scan of 286 samples to a column in a 2-D grid. The columns are added in the same order as they were scanned. In this way, each element in the organized grid serves as an index to an (x, y, z) point position in 3-D space. The grid also serves a second purpose: Each element in the grid is associated with the attributes of the corresponding point in the point cloud. If, for example, the attribute is intensity, then the attribute grid is an image, representing in effect a flattened image of the scanned environment. Because the position of the sensor is fixed relative to the vehicle, the entire set of scanned points is mapped to a long straight image, as shown in Fig. 2. For any sample in which no reflected light is picked up by the sensor, an empty placeholder value is assigned to the position in the 2-D grid or image, as indicated by the red elements in the figure.

Algorithm 1 Generate Bézier Patches Adaptively

```
1:  function FITBEZIERPATCHADAPTIVE($\mathcal{F}$)
2:      $B \leftarrow \varnothing$
3:      $R \leftarrow$ INITSPLIT(Domain($\mathcal{F}$))
4:      for each $r \triangleq [u_{min}, u_{max}] \times [v_{min}, v_{max}] \in R$ do
5:          $R \leftarrow R \setminus r$
6:          $b \leftarrow$ FITBEZIERPATCH($r, \mathcal{F}, d$)
7:          if $b$ is $\varnothing$ then continue
8:          if NEEDSPLIT($b$) and CANSPLIT($r$) then
9:              $\{r_0, r_1\} \leftarrow$ SPLIT($r$)
10:             $R \leftarrow R \cup \{r_0, r_1\}$
11:         else
12:             $B \leftarrow B \cup \{b\}$
13:     return $B$
```

2.2. Generating hierarchical patch models

Consider an organized point cloud $\mathcal{F}$, where $\mathcal{F} = \{p_{i,j} \in \mathbb{R}^3; i = 1, \cdots, M, j = 1, \cdots, N\}$, and the 2-D indices (i, j) and $(i \pm 1, j \pm 1)$ reflect the 3-D proximity relationship between corresponding points unless there are depth discontinuities. Our goal is to approximate $\mathcal{F}$ with a function defined on its index domain, to facilitate the subsequent compression.

There are many function models for such approximations. We adapt a spline surface model, which is a popular choice in geometry processing and CAD due to its flexibility to represent surfaces having different levels of details using different numbers of control points [18]. Conventional spline fitting algorithms always start the approximation from a coarse B-spline or Bézier surface. Then it is refined at places with large fitting errors by inserting more control points into its control mesh. After this, all input data points are used again to refine the new set of control points. This process is then iteratively performed until a satisfactory approximation is achieved.

During each iteration, every single data point will be accessed for a least squares fitting. For large-sized point clouds with many fine details, such methods might not be efficient for fast processing. However, unlike CAD modeling, we are not constrained by a single piece of approximation for compression. Thus we propose to increase such methods' efficiency by adaptively dividing $\mathcal{F}$ into a set of B-spline/Bézier patches, according to a prescribed fitting error threshold. Each of these patches corresponds to a rectangular sub-domain in the input data's parameter domain, i.e., the index domain for an organized point cloud. This adaptive patch generation is summarized in Algorithm 1 and detailed below.

Based on our notation, Domain($\mathcal{F}$)$\triangleq [1, M] \times [1, N]$. Similar to [19], we uniformly split the entire input domain into several regions by the INITSPLIT function, to avoid unnecessary initial patch fitting.

The FITBEZIERPATCH function takes all data points within the domain r to fit a Bézier patch b defined on that r. This is done by solving the following equation using either standard QR or Cholesky factorization [18]:

$$\mathbf{P}^\star = \arg\min_{\mathbf{P}} \| \mathbf{BP} - \mathbf{Q} \|_F^2 + \lambda \| \mathbf{SP} \|_F^2 . \quad (1)$$

Here each row of $\mathbf{P}$ represents a Bézier control point. Input data points within region r are stored in each row of $\mathbf{Q}$, and the (k, ℓ) entry of the $\mathbf{B}$ matrix stores the ℓ-th control point's Bézier basis function value evaluated at the parameter of the k-th data point in r. Note that $\mathbf{B}$ only depends on the size of the region r. Sometimes the FITBEZIERPATCH function cannot perform the least squares fitting

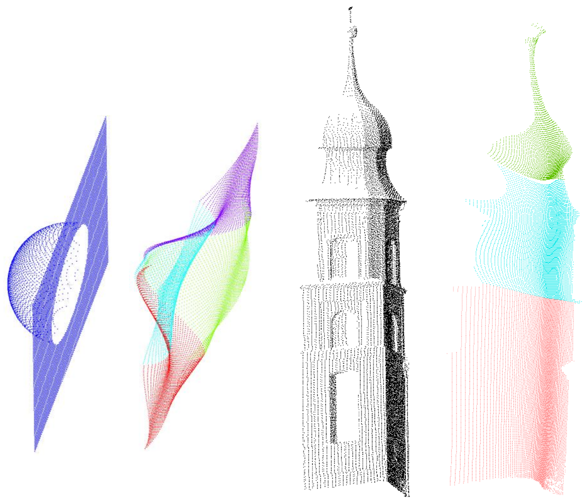


Fig. 3: Examples of point clouds and corresponding patches. The tower point cloud was generated using the dataset from [23].

due to rank deficiency of $\mathbf{B}$. This usually occurs at small regions with large detail variations, or regions with too many missing data. We either return an empty fit b and ignore the corresponding r, or add linear constraints $\mathbf{S}$ between control points with trade-off parameter λ to make the above system rank sufficient. Example constraints can be either simply forcing neighboring control points to be close to each other, or more sophisticated ones to suppress wiggling fit as explained in [20].

The NEEDSPLIT function can use different criteria to determine whether or not b is a bad fit and thus needs to be further split into smaller parts. If the input data is known to have the same isotropic error everywhere, then this function can check the L^∞ fitting error with a prescribed threshold. Otherwise, for example, for Kinect data which is known to have depth-dependent errors, this function can check the fitting error with a dynamic depth-dependent threshold [21, 22].

The CANSPLIT function tests whether a given domain r can be further split. In our implementation, if the split domains r_0, r_1 do not contain enough data for a valid Bézier fit, i.e., both $u_{max} - u_{min} + 1$ and $v_{max} - v_{min} + 1$ are smaller than $2d + 1$ (d is the prescribed degree of the spline function), then this r cannot be split. This is because after splitting, the resulting blocks will have less data points than control points, leading to a rank deficient system unless we perform the constrained fitting in Eq. (1). There is a case where a patch needs to be split but cannot be split. This usually happens at small blocks with too significant details that cannot be represented as a simple Bézier surface. There are two options to handle this case: either performing B-spline refinement on that Bézier surface until the fitting error is small enough, or directly signaling the input points. In our implementation we select the latter one because those tiny details are usually caused by sensor noise and signaling a full model to represent a few points is typically less efficient than sending the points directly.

The SPLIT function can have different behaviors: split at the patch center, or adaptively split according to fitting errors. In our implementation, we always split at the middle of the longer side of the domain to avoid thin domains which do not tend to give good fitting results. Some examples of simple patch models are shown in Fig. 3, where each patch is indicated by color.

3. COMPRESSION FRAMEWORK

Using the methods described in the previous section, the point locations or geometry of the 3-D point cloud are now represented as a set of control points for generating Bézier patches, the locations of empty positions on the organizational grid, a splitting tree to indicate the partitioning of the grid, and ancillary data such as the organizational grid size. The attributes such as intensity are similarly organized into an image. Compressing the image is straightforward using any image compression scheme. For empty positions, the pixel in the image can be assigned any value, as it is not associated with a point in 3-D space. The performance of the attribute compression scheme in this case would be that of whatever image compression scheme is used, e.g. 8-bit grayscale attributes could be compressed to below 1.0 bits per point by using JPEG, or lower using newer compression schemes. The remainder of this paper focuses on compressing the point cloud geometry.

For each patch model, we compute residual values, which are the difference in 3-D space between the input point cloud and the corresponding points in the patch. Note that there is a simple one-to-one correspondence between the points generated for the patch and the input points, as the input points are arranged according to the organized grid. Thus, no nearest-neighbor computations are needed for computing the residuals.

For compression, we quantize the residuals and optionally the control points. We apply a simple uniform quantizer for these results; future experiments will use a quantizer better matched to the distribution of residuals. For signaling into a bit-stream, we first send the width and height of the organizational grid, followed by a hierarchical binary splitting tree to indicate how the grid is partitioned. For every leaf node on the tree, we signal the control point values and then we signal either the organized grid of residual values or the input values themselves if no model was used for that patch. Typically, an entropy coder would be used to code these values. For this paper, we report entropies as an upper bound on performance and leave the tuning of a context-adaptive entropy coder for future work.

4. EXPERIMENTAL RESULTS

We first present point cloud geometry coding results for the simple 100×100 hemisphere point cloud shown in the left side of Fig. 3. We use the symmetric point-to-point geometric PSNR described by [24] as the performance metric. The input (x, y, z) positions are represented as 32-bit floating point values for each component. The uncompressed input point cloud positions, as well as the control points, therefore use 96 bits per point. Each patch uses a 4×4 set of control points. We compare our codec to the octree-based codec of [3]. The performance obtained when representing the point cloud only using patches is shown in Fig. 4(a). Here, we can achieve almost up to 40 dB with compression ratios of over 1000:1 by signaling only the control points. Performance including quantized residuals is shown in Fig. 4(b). Here, 40 to 100 dB is achieved as the quantization of the residuals becomes finer. The performance improvement of the patch-fitting codec over the octree-based codec ranges from 5 to 20 dB, and the octree codec is not capable of achieving the extremely low rates below a few bits per input point.

For the next set of experiments, we code the subset of the MMS point cloud shown in the inset of Fig. 1. This point cloud contains approximately 290,000 points organized to a 286×1072 grid. The resulting model contains approximately 300 patches when a squared fitting error threshold of 5×10^6 is used. The coordinates are in the range of ± 10000, so for the octree codec we specify an input

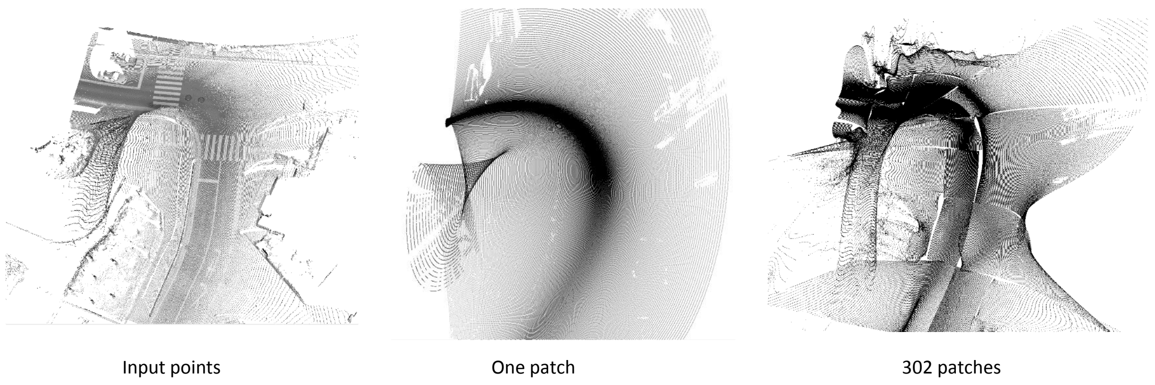
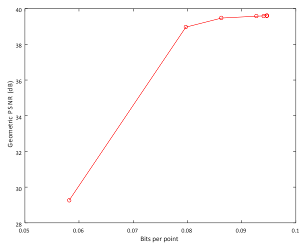
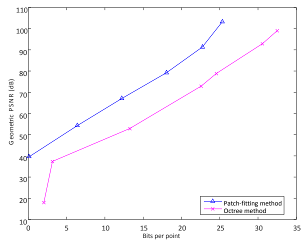


Fig. 6: Single and multiple patch models representing the input point cloud

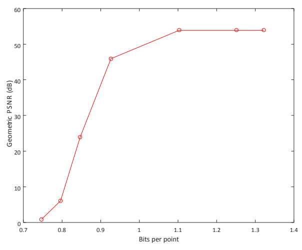


(a) Reconstruction uses only quantized control points, without residuals

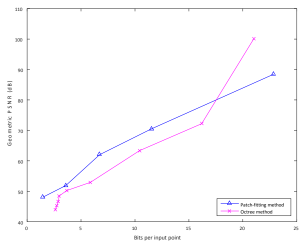


(b) Reconstruction uses near-lossless control points and quantized residuals

Fig. 4: Patch-fitting codec geometric coding performance for hemisphere point cloud



(a) Reconstruction uses only quantized control points, without residuals



(b) Reconstruction uses near-lossless control points and quantized residuals

Fig. 5: Patch-fitting codec geometric coding performance for MMS point cloud

resolution of 1.0, and we vary the octree resolution from 100 to 1200. As shown in Fig. 5(a), we achieve up to 55 dB using the patch-fitting codec with control points alone. Fig. 5(b) shows the performance when using near-lossless control points and quantized residuals. Here, the patch-fitting codec outperforms the octree codec by 2 to 5 dB, except at higher rates where the octree codec performs better. We believe that the octree performs better at higher rates because when there are fewer points per octree leaf node, coding leaf node positions via the octree splitting tree uses fewer bits than coding differential values among many points within an octree leaf node. However, both the patch-fitting and octree codecs would benefit from further tuning, taking into consideration the precision and resolution of the input point cloud, and in this case, the fact that the point cloud spans the x and y directions broadly, but the range of z values are limited due to the orientation of the MMS scan.

Fig. 6 shows the patch models for a portion of the input point cloud shown on the left side of the figure. One patch is capable of capturing the essence of the point cloud, in that it does a good job of following the curve in the road. The final patch model used for this experiment has 302 patches, as shown on the right side of the figure. The smaller more complex patches are grouped around complex objects such as the vehicles near the top of the figure. For the model having 302 patches, the organized grid of Fig. 2 is partitioned as shown in Fig. 7. Larger patches tend to be used to represent large flat surfaces such as the road, as expected.

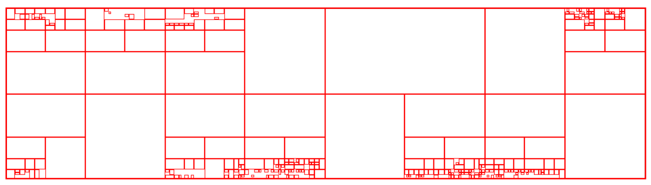


Fig. 7: Partitioning map containing 302 patches

5. SUMMARY AND CONCLUSIONS

In this paper we presented a method for compressing point clouds by organizing the 3-D points onto a 2-D grid and adaptively fitting them with hierarchical patches. For simple point clouds to which patches are easy to fit, the patch-fitting codec outperformed an octree-based codec by up to 20 dB for geometry compression. For complex point clouds such as those obtained from a mobile-mapping system, the performance of the patch-fitting codec was a few dB better than the octree codec, but at higher rates the octree codec performed better. Since we typically code the control points with high fidelity, there is a tradeoff between the coding performance and the number of patches, in that for very small patches, the number of bits needed to represent the control points can exceed the amount needed to directly code the input points. Future work will include determining optimal parameters for both codecs, improving the quantizer and related coding systems within the patch-fitting codec, and experimenting with other types and sizes of patches.

6. REFERENCES

[1] A. Maglo, G. Lavoué, F. Dupont, and C. Hudelot, "3D mesh compression: Survey, comparisons, and emerging trends," *ACM Comput. Surv.*, vol. 47, no. 3, pp. 44:1–44:41, Feb. 2015.

[2] J. Peng and C.-C. Jay Kuo, "Geometry-guided progressive lossless 3D mesh coding with octree (OT) decomposition," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 609–616, July 2005.

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

[4] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, CN, May 2011.

[5] R. Mekuria, K. Blom, and P. Cesar, "Design, implementation and evaluation of a point cloud codec for tele-immersive video," *IEEE Trans. Circuits Syst. Video Technol.*, Jan 2016.

[6] C. Zhang, D. Florencio, and C. Loop, "Point cloud attribute compression with graph transform," in *2014 IEEE International Conference on Image Processing (ICIP)*, Oct. 2014.

[7] C. T. Loop, C. Zhang, and Z. Zhang, "Real-time high-resolution sparse voxelization with application to image-based modeling," in *Proceedings of the 5th High-Performance Graphics 2013 Conference HPG '13*, Anaheim, California, USA, July 2013, pp. 73–80.

[8] R. A. Cohen, D. Tian, and A. Vetro, "Attribute compression for sparse point clouds using graph transforms," in *2016 IEEE International Conference on Image Processing (ICIP)*, Sep. 2016.

[9] K. Ainala, R. N. Mekuria, B. Khathariya, Z. Li, Y.-K. Wang, and R. Joshi, "An improved enhancement layer for octree based point cloud compression with plane projection approximation," in *Proc. SPIE 9971, Applications of Digital Image Processing XXXIX*, Sep. 2016.

[10] D. Huber, "The ASTM E57 file format for 3D imaging data exchange," in *Proceedings of the SPIE Vol. 7864A, Electronics Imaging Science and Technology Coonference (IS&T), 3D Imaging Metrology*, Jan. 2011.

[11] The American Society for Photogrammetry & Remote Sensing, "LAS specification version 1.4 R13," `http://www.asprs.org/LAS_Specification`, [Online].

[12] M. Isenburg, "LASzip: lossless compression of LiDAR data," in *Photogrammetric Engineering & Remote Sensing*, no. 2, Feb. 2013.

[13] R. A. Cohen, D. Tian, M. Krivokua, K. Sugimoto, A. Vetro, K. Wakimoto, and S. Sekiguchi, "Representation and coding of large-scale 3D dynamic maps," in *Proc. SPIE 9971, Applications of Digital Image Processing XXXIX, 99710T*, Sep. 2016.

[14] ISO/IEC JTC1/SC29/WG11, "Use cases for point cloud compression," document MPEG2016/N16331, ISO/IEC JTC1/SC29/WG11 *Coding of Moving Pictures and Audio*, Geneva, Switzerland, June 2016.

[15] ISO/IEC JTC1/SC29/WG11, "Requirements for point cloud compression," document MPEG2016/N16330, ISO/IEC JTC1/SC29/WG11 *Coding of Moving Pictures and Audio*, Geneva, Switzerland, June 2016.

[16] ISO/IEC JTC1/SC29/WG11, "Call for proposals for point cloud compression," document MPEG2017/N16732, ISO/IEC JTC1/SC29/WG11 *Coding of Moving Pictures and Audio*, Geneva, Switzerland, Jan. 2017.

[17] "Mobile mapping system – High-accuracy GPS mobile measuring equipment," `http://www.mitsubishielectric.com/bu/mms/features/index.html`, [Online].

[18] L. Piegl and W. Tiller, *The NURBS Book (2Nd Ed.)*, Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[19] H. Lin and Z. Zhang, "An efficient method for fitting large data sets using T-spline," *SIAM Journal on Scientific Computing*, vol. 35, no. 6, pp. A3052–A3068, 2013.

[20] D. Brujic, I. Ainsworth, and M. Ristic, "Fast and accurate NURBS fitting for reverse engineering," *The International Journal of Advanced Manufacturing Technolog*, vol. 54, no. 5-8, pp. 691–700, 2011.

[21] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of Kinect depth data for indoor mappingapplications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[22] C. V. Nguyen, S. Izadi, and D. Lovell, "Modeling Kinect sensor noise for improved 3D reconstruction ad tracking," in *Proc. Int'l Conf. 3D Imaging, Modeling, Processing, Visualizaion and Transmission (3DIMPVT)*, 2012, pp. 524–530.

[23] "Large-scale point cloud classification benchmark," `http://www.semantic3d.net`, [Online].

[24] R. Mekuria and P. Cesar, "Proposal for quality and performance metrics point cloud compression," document MPEG2015/m35491, ISO/IEC JTC1/SC29/WG11 *Coding of Moving Pictures and Audio*, Geneva, Switzerland, Feb. 2015.