2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

Real-time Compression of Point Cloud Streams

Julius Kammerl*, Nico Blodow†, Radu Bogdan Rusu‡, Suat Gedikli‡, Michael Beetz†, and Eckehard Steinbach*
*Institute for Media Technology, Technische Universität München, Munich, Germany
Email: {kammerl, eckehard.steinbach}@tum.de
†Intelligent Autonomous Systems Group, Technische Universität München, Munich, Germany
Email: {blodow, beetz}@in.tum.de
‡Willow Garage, Inc., Menlo Park, CA, USA
Email: {rusu, gedikli}@willowgarage.com

***Abstract*—We present a novel lossy compression approach for point cloud streams which exploits spatial and temporal redundancy within the point data. Our proposed compression framework can handle general point cloud streams of arbitrary and varying size, point order and point density. Furthermore, it allows for controlling coding complexity and coding precision. To compress the point clouds, we perform a spatial decomposition based on octree data structures. Additionally, we present a technique for comparing the octree data structures of consecutive point clouds. By encoding their structural differences, we can successively extend the point clouds at the decoder. In this way, we are able to detect and remove temporal redundancy from the point cloud data stream. Our experimental results show a strong compression performance of a ratio of 14 at 1 mm coordinate precision and up to 40 at a coordinate precision of 9 mm.**

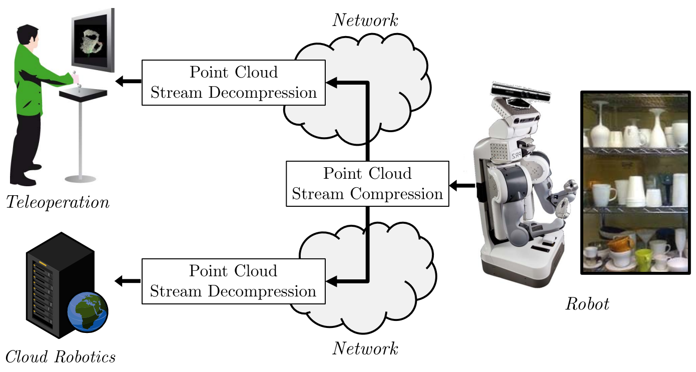

Network
Point Cloud Stream Decompression
Point Cloud Stream Compression
Teleoperation
Point Cloud Stream Decompression
Robot
Cloud Robotics
Network

Fig. 1. Schematic overview of compressed point cloud streaming scenarios.

I. Introduction

Recently, 3D data acquisition systems have become increasingly wide-spread and as technology is advancing, so is scan resolution and accuracy. Sensors such as the Microsoft Kinect, stereo camera systems or high-speed, high-resolution laser scanners can produce large amounts of point data at high frame rates. This sensor data can be efficiently represented by point clouds from which a robot can infer information about the shape and geometry of their environment, detect objects as well as to orient itself [1]. Additional applications are apparent in the field of 3D video and multimedia. However, due to the large size of the point clouds, the processing and transmission of point cloud information occupies a significant amount of resources. Particularly, the real-time transmission of point clouds in the context of remote data processing or in teleoperation scenarios is challenging (see Fig. 1). For bandwidth-limited scenarios such as online streaming or storage, it therefore becomes necessary to develop efficient compression solutions.

For sensors that produce *depth images*, several approaches have been proposed in the literature. Some adopt existing methods from the field of image compression, others employ special data structures and algorithms specifically designed for these regularly sampled and organized point clouds. However, in scenarios using multiple 3D camera systems [2], the overlapping image regions carry redundant information for which different solutions have to be considered. This, for instance, also applies to tilting laser scanners which are mounted on moving platforms, where — depending on the relative position and locomotion velocities — a single frame cannot be assumed to be taken at a unique point. Here, the incoming points are typically prealigned according to information concerning sensor trajectories which is often available (or can be estimated using various SLAM variants).

We approach this challenge by treating the data as *unorganized point clouds*, i.e., the point sets can be stored in arbitrary order, and the scans can have holes or irregular sampling densities. This makes our proposed system very general and applicable for a wide range of sensors and data acquisition strategies.

Compression algorithms can broadly be categorized in *lossless* (i.e. all information contained in the original data set can be reconstructed from the compressed representation) and *lossy*. For the latter, details are sacrificed — to a certain degree — in exchange for significantly higher compression ratios. Since in many applications the raw data is inherently noisy due to the physical process involved in the acquisition system, a lossy compression scheme is preferable as long as the introduced distortion is kept below or close to sensor noise levels.

Another important criterion for comparing different compression solutions is the computational efficiency, which — at least for online streaming and coding — we would like to keep as close to real-time (i.e. frame rate) as possible.

We therefore propose a general and extensible, lossy point

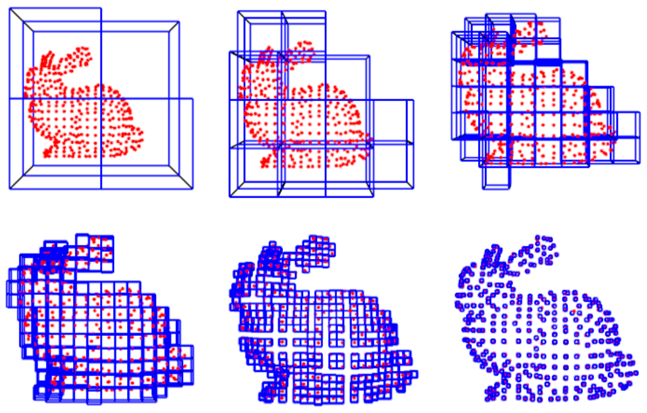


Fig. 2. Six levels of a sample octree for the Stanford bunny data set. Every voxel contains one or more points.

cloud stream compression system that can be tuned to meet different resolution, complexity and accuracy requirements, and allows the co-encoding of additional point details apart from merely point position, e.g. colors, intensities or geometric features such as normals or curvature values. Note that our approach focuses on keeping computational requirements minimal to reduce processing latency and enable real-time or close to real-time encoding and decoding performance. The presented compression approach in this publication has been implemented and released as part of the open-source Point Cloud Library (PCL) [3].

The main contribution of our system is its ability to efficiently detect and encode spatially and temporally redundant signal information from a stream of point clouds. In this context, we propose a modified octree data structure (double-buffering octree) that enables detection and differential encoding of spatial changes within temporarily adjacent unorganized point clouds. Another important contribution is the ability to reduce coding complexity by applying additional point detail encoding.

The authors are not aware of any standardized test sets for 3D point cloud streaming. We therefore analyze our system in terms of computational complexity and bit-rate reduction on point cloud streams generated by Microsoft's Kinect device.

The remainder of this paper is organized as follows. In the next section, we give an overview of related work on point cloud compression. Our octree-based compression approach for single point clouds is presented in Section III. In Section IV, we describe how to apply these techniques to the compression of point cloud streams. Details on the encoding of additional point details is given in Section V. We explain our compression architecture in Section VI, followed by the experimental evaluation in Section VII, and conclude in Section VIII.

II. Related work

Several approaches for point cloud compression have been proposed in the literature. Among those, progressive point cloud coding techniques constitute an important class. By performing a multi resolution spectral decomposition of the point data set, they allow for successively predicting and differentially encoding point clouds of higher resolution from their subsampled low resolution versions. In this way, a low resolution point cloud with several levels of refinement layers can be built which greatly reduces the entropy within the point cloud data set and, hence, allows for efficient compression of the point cloud data. In this context, [4] uses a kd-tree structure for recursively splitting and subdividing the point cloud space. Furthermore, [5] and [6] employ octree structures for this purpose which provide an efficient description of the spatial point cloud distribution in a binary stream format. Similarly, [7] discusses an octree-based point cloud compression approach which is combined with a specialized prediction technique for point sampled geometry and surface approximation.

In contrast, [8] presents a bottom up approach that builds a multi resolution point cloud hierarchy by successively grouping neighboring pairs of points and replacing them by their median. This enables the predictive encoding of the differences between the median and the original points which results in a data representation with reduced entropy. In [9], the authors propose to construct a minimum spanning tree over the point set allowing for predictive coding of point neighborhood and point distance information. A similar spanning prediction tree technique is discussed in [10]. Here, best predictable points are successively attached to the spanning tree, followed by the encoding of their corresponding prediction error. The authors in [11] introduce the concept of ”progressive point-set surfaces” using moving least-squares (MLS) surfaces. They initially downsample the point cloud from the MLS surfaces, followed by determining refinement layers based on point projections onto MLS reference planes.

Research focusing on compression of animated geometry mainly addresses the compression of animated vertices, triangle and mesh models. In this context, [12] proposes to match and encode affine transforms between consecutive frames which removes temporal coherence in the geometry data stream. Similarly, [13] calculates principal components for decoupling animation and geometry information. Furthermore, [14] introduces an octree-related compression approach for animated meshes. Here, motion vectors between vertices of consecutive frames are extracted to be further compressed by applying a hierarchical octree decomposition. All these compression techniques for animated geometry do, however, not directly apply to the compression of unorganized point cloud streams as they require consistent references between the consecutive frames. In contrast, unorganized point clouds do not provide any direct point-to-point references due to varying size, density and point order. Addressing this issue by applying point cloud registration techniques for estimating the missing point references would lead to a significant increase of computational complexity.

Interestingly, neither *real-time* compression techniques for static (single frame) point sets nor compression methods for dynamic point cloud *streams* have been intensively addressed

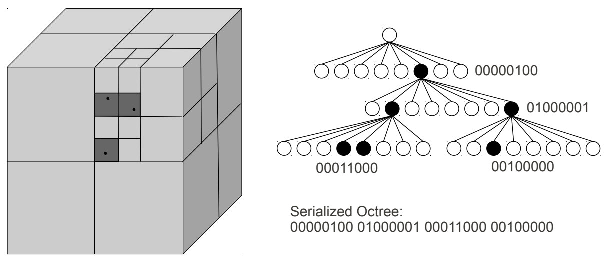


Fig. 3. Schematic overview of the octree data structure and its serialization.

in literature so far. Due to the large point cloud size and high temporal sampling rate, online point cloud compression is challenging and requires efficient and low-complexity methods for point data analysis, prediction and coding. In this work, we present a novel point cloud compression approach for online streaming which works with arbitrary point cloud streams.

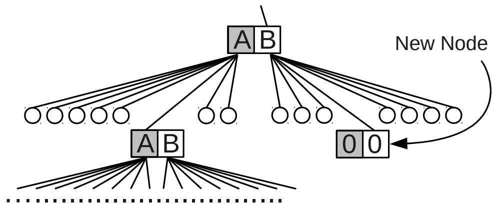
III. Octree-Based Point Cloud Compression

Octrees are used in various applications that require spatial decomposition of the point set, such as spatial indexing, nearest neighbor searches or collision detection in physics engines, etc. In this section, we discuss our proposed method for reducing the storage requirements of a single point cloud by employing an octree structure to compress the point data.

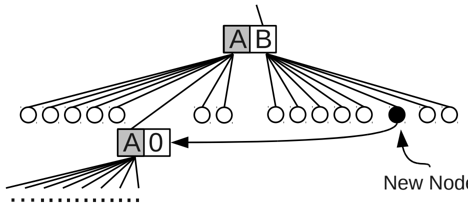
An octree is a tree data structure suitable for sparse 3D data, where every branch node represents a certain cube or cuboid bounding volume in space. Starting at the root, every branch has up to eight children, one for each sub-octant of the node's bounding box. An example is shown in Fig. 2 and in the left half of Fig. 3. Common implementations create an object in memory for each node, holding 8 pointers to other nodes. If a child node does not exist, it is represented by a *NULL* pointer.

The depth of the octree is limited either directly or by specifying a target leaf box size. When constructing the octree, every point is added iteratively by traversing the tree in depth-first order starting at the root, and determining the correct child octant in every branch, creating new children as necessary, until the point can be appended to the list of points stored in each leaf node. If only the leaf node occupancy is of interest, each leaf node contains a single boolean variable.

If one sets a single bit for every existing child of a branch node, this child node *configuration* can be efficiently represented in a single byte, assuming some consistent ordering of the eight octants. This is the underlying idea behind the octree-based point cloud compression. By traversing the tree in breadth-first order and outputting every child node configuration byte we encounter, we are able to efficiently encode the point distribution in space. Upon reading the encoded byte stream, the number of bits set in the first byte tells the decoder the number of consecutive bytes that are direct children. The bit positions specify the voxel/child in the octree they occupy. The right hand side of Fig. 3 illustrates this byte stream representation for the octree in the left half. Note that this stream encodes the whole octree structure, and the size of this stream is n bytes, where n is the number of branch nodes.



(a) Adding a new node to octree B that refers to a new voxel. Both pointer buffers are initialized with zero.



(b) Adding a new node to octree B referring to a voxel that already exists in the preceding octree structure A. Only the child pointers in the currently assigned buffer B are initialized with zeros.

Fig. 4. Illustration of the "double buffering" octree technique. The pointer buffer A contains the pointers of the previously built octree structure and the pointer buffer B is assigned to the current octree build process.

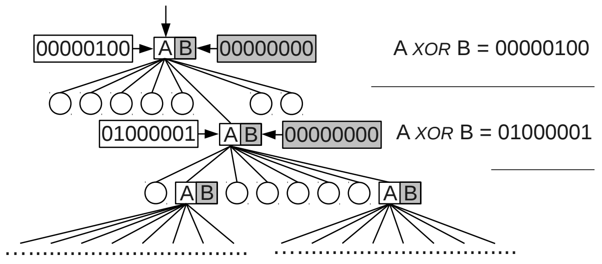
To fully reconstruct the point cloud, however, some additional parameters such as the root bounding volume and the octree depth need to be encoded. The encoding of leaf nodes will be detailed in Section V.

IV. Temporal Compression of Point Clouds

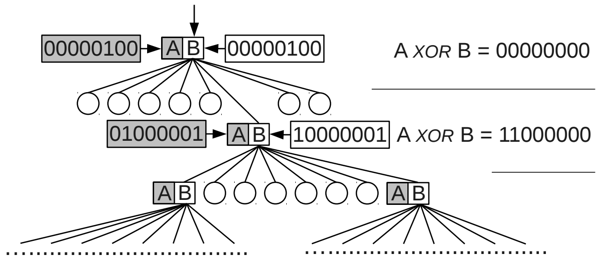
The output byte stream of the octree serialization is already a compact description of the *spatial* point distribution. However, as changes within the captured scene often occur only partially in confined areas, also *temporal* correlation within point clouds that are captured at adjacent time instances can be expected. Hence, by correlating and referencing the currently sampled point data to previously encoded point cloud information, further improvements in compression performance can be expected. In other words, we can exploit temporal redundancy by initially performing a full compression once for the first point cloud, and subsequently only encode extracted changes within the point distribution.

Due to the lack of direct correspondences between points of different point clouds in the stream, the detection of changes within the point distribution is challenging. To this end, we propose a novel double-buffering octree scheme which extends our octree capabilities and enables comparison and differential coding of consecutive octree data structures.

This is achieved by adding a second set of eight child pointers to every branch node now containing a total of $2 \cdot 8 = 16$ pointers to child nodes. Initially, the collection of all pointers in all branches belonging to the first pointer set are dubbed the first child node pointer buffer A and all pointers belonging to the second pointer set constitute the second pointer buffer B. From there on, the two child pointer buffers are alternately assigned to every incoming point cloud. This interleaved storage of two consecutive point clouds in one



(a) Encoding of an initial point cloud stored in pointer buffer *A*. Buffer *B* initially contains zero pointers.



(b) Differential encoding of a consecutive point cloud using pointer buffer *B*. Buffer *A* contains the previously processed octree structure.

Fig. 5. Illustration of the differential encoding technique of consecutive octree data structures. Structural changes within octree branch nodes are extracted during serialization and encoded using the exclusive disjunction operator (XOR).

octree data structure allows us to spatially match, reference and analyze the structural changes within consecutive octree structures.

Initial to every incoming point cloud, the root node of the currently assigned pointer buffer is initialized with zeros. From then on, the octree structure is recursively built according to the incoming point set (see Section III). Whenever new child nodes need to be created, we perform a lookup to the preceding pointer buffer within the current branch node. If no corresponding reference pointer can be found, the corresponding voxel did not exist in the previously processed octree structure and a new child node is created. If, however, a corresponding child node pointer can be found in the preceding pointer buffer, we simply adopt its reference and initialize the selected child pointer buffer with zeros. Furthermore, we expand the size of the octree whenever incoming points violate its bounding box. This way, we continuously ensure structural consistency to the preceding point cloud and corresponding octree data structure by adaptively matching spatially related voxels/octree nodes. A schematic overview of this procedure is illustrated in Fig. 4.

Once a new octree structure is built, we traverse the octree as in the single frame case outlined in Section III. However, in order to obtain and output the structural changes for every branch, we apply an exclusive disjunction operation (*XOR*) on the two bytes representing the child node configurations of the child pointer buffers *A* and *B*. As the very first octree structure is *XOR*-referenced to the initialized zero pointer buffers, we automatically obtain a full binary description of the octree. This makes it particularly easy to generate later reference frames as well, as only the preceding reference buffers have to be reset to zero. From then on, the two pointer buffers are alternately assigned to incoming point clouds and overwritten during their octree creation. If the child configuration in buffers *A* and *B* are identical, the *XOR* encoded configuration byte is zero. Accordingly, changes in branch node configurations evoked by either new or vanishing branches are described by set bits in the output stream. This way, the serialized output for all regions in the scene that have not changed in occupancy is zero which leads to reduced entropy of the output stream. This can be exploited for further entropy compression using e.g. an arithmetic coder, as outlined in Section VI. The proposed *XOR*-based differential octree encoding scheme is illustrated in Fig. 5.

During decoding, the serialized binary octree description is used to successively rebuild a consistent double-buffering octree data structure. It serves as the required reference for enabling the decoding and interpretation of the *XOR*-encoded branch node configurations. The reconstructed point cloud is obtained by extracting the centroids of encoded leaf nodes. In order to cope with transmission or synchronization issues, one can reset the reference buffers to zero after a certain number of point cloud frames, effectively inserting an intra frame. It is also important to note that during the serialization and deserialization traversal (at the encoder and decoder), one can delete all vanished branch nodes in the previously processed octree structure without much additional overhead.

Note that if the octree bounding volume needs to be increased between consecutive frames, a new intra frame can be triggered, or additional information must be encoded.

On a tangent, the proposed double-buffering octree scheme can also be used stand-alone to perform an online detection of spatial or temporal changes in unorganized point clouds, similar to visualizing intensity differences in a video stream. This is illustrated in Fig. 6, where all points reconstructed from newly generated leaf voxels are highlighted in red. Note that for better visibility, the two frames are taken roughly one second apart.

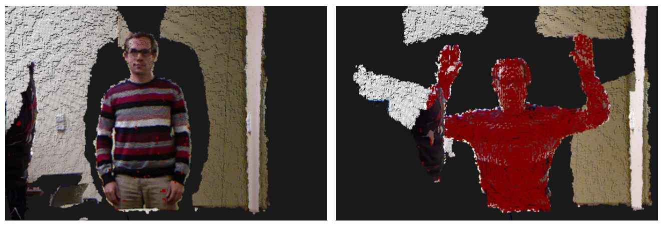
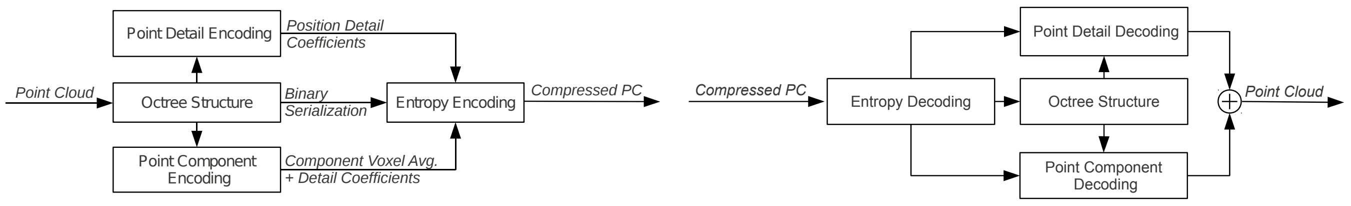


Fig. 6. Changes in voxel occupancy between consecutive frames are highlighted in red. Please refer to the online version of this paper for a color reproduction.

V. Point Detail & Point Component Coding

In the preceding sections, we outlined how we efficiently encode spatial and temporal information of point cloud streams using the proposed double-buffered octree structures. But by



(a) Encoder (b) Decoder

Fig. 7. Schematic overview of the encoding and decoding architecture.

itself, this only allows for representing the point cloud up to the resolution of the octree leaf voxels (cf. Fig. 9). Basically, all points assigned to one particular voxel are collectively represented by that voxel center only. This could be remedied by increasing the octree resolution which, however, would result in exponential computation and memory requirements jeopardizing the real-time processing constraints. For high resolution online compression of point clouds, we therefore propose to limit the octree resolution and augment the differential byte stream by appending additional point detail information, as follows.

For each occupied voxel in the octree, we compute the distances between each point and the leaf node origin, i.e. the voxel corner with smallest x, y, z dimensions (see Fig. 8). This voxel-local point location is discretized according to the specified target coding precision which results in positive integer representations (i.e. point detail coefficients) for all voxel-local point coordinates. These integer values are limited by the quotient of the voxel resolution and the target compression precision (e.g. an interval of [0,8] to achieve a 1 mm coordinate precision at an octree leaf resolution of 9 mm). During subsequent entropy coding, the reduced symbol set allows each point detail coefficient to be efficiently represented by a small number of bits. However, the distribution of points within octree voxels (and thus the corresponding symbol frequencies in this stream) must be considered to be uniformly distributed which impairs the compression efficiency during entropy coding. Hence, the point detail encoding technique allows us to reduce the computational complexity which is important for enabling real-time compression at the expense of reduced compression performance.

The principle of point detail coding can be applied to different kinds of point component information that is assigned to each point, e.g. color, normals, etc. Instead of referencing the voxel origin coordinate, we compute and transmit the

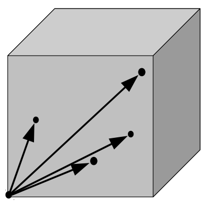


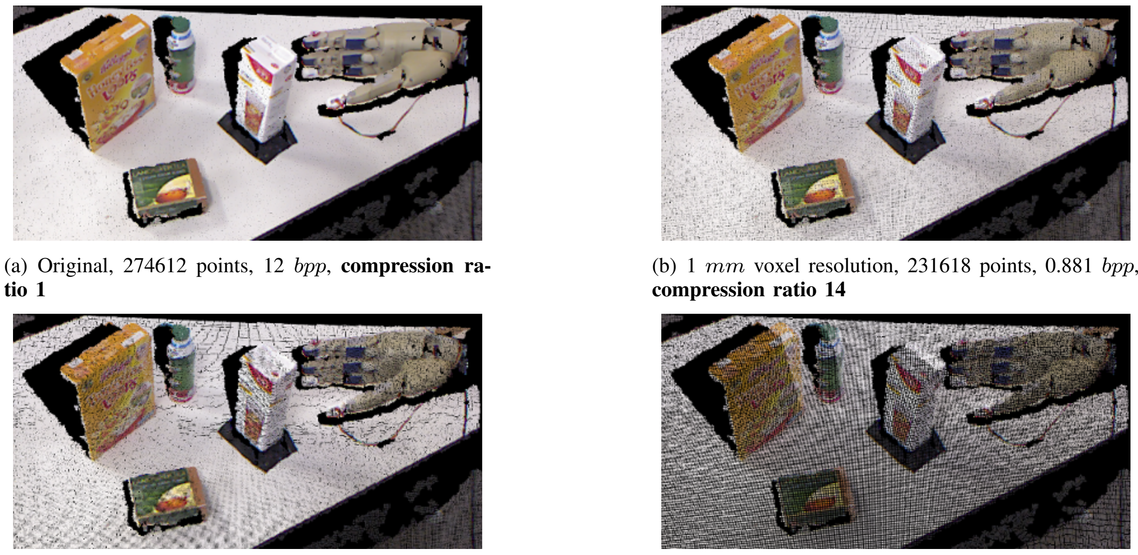
Fig. 8. Illustration of point detail encoding. All points assigned to a particular leaf voxel are differentially encoded relative to the voxel origin.

average color and perform an encoding of the color differences analogously to the cartesian point detail differences. For other information, e.g. surface normal vectors, different approaches have been shown to perform well, e.g., converting the normals from Cartesian space to polar space first [8]. Assuming similar component information of neighboring points, the differentially encoded point components are characterized by reduced entropy which is exploited during subsequent entropy coding. Since point clouds can be enriched with arbitrary additional dimensions that can exhibit largely varying properties, it makes sense to analyze and fine tune these encoding details as needed instead of relying on one generic approach. However, this falls outside the scope of this paper.

VI. Compression architecture

The resulting compression and decompression architecture is illustrated in Fig. 7. In Fig. 7(a), the encoder builds the doubled buffered octree structures and extracts their *XOR*-encoded binary sequence during serialization, as described in Section III and Section IV. Voxel-local point details are extracted and encoded separately during the performed serialization process, which generates a stream of position detail coefficients (all relative to their respective leaf voxel center) as described in Section V. Similarly, additional point components such as color and normals are serialized into a separate stream each, containing the component voxel averages and corresponding detail coefficients. Since their respective symbol frequencies are possibly different, these streams are consumed by the entropy encoder separately and multiplexed into a joint output data stream, which constitutes the compressed point cloud data to be written to e.g. a network stream or file. Note that for speed considerations, we use an integer-arithmetic variant of an arithmetic coder for entropy coding, a so called range coder [15]. Prior to every entropy encoded bit stream, corresponding symbol frequency tables are additionally encoded.

The decoder (illustrated in Fig. 7(b)) performs demultiplexing followed by entropy decoding of the individual data streams. The serialized binary octree description is fed to the respective deserialization routines used to decode and reconstruct a coherent double buffered octree structure. Whenever leaf voxels are constructed during the deserialization process, their absolute position is fed to the point detail decoding and point component decoding routine. This enables us to recover the original point set of the encoded point cloud together with



(a) Original, 274612 points, 12 *bpp*, **compression ratio 1**

(b) 1 mm voxel resolution, 231618 points, 0.881 *bpp*, **compression ratio 14**

(c) 2 mm voxel resolution, 204843 points, 0.491 *bpp*, **compression ratio 24**

(d) 3 mm voxel resolution, 144816 points, 0.380 *bpp*, **compression ratio 32**

Fig. 9. Performance of octree compression of a single point cloud. Points are represented by the centroid of the leaf voxels. Note the alignment of the encoded point cloud with the voxel grid of the octree. For better visualization, points are shown in color.

its respective point component information.

VII. Experimental evaluation & discussion

In order to assess the performance of our system, we use a single point cloud of a robotic kitchen scenario containing ca. 270k points and a 15 seconds test sequence of point clouds capturing a person standing in an office room and waving (see Fig. 6). Due to varying occlusion effects and sensor noise originating from e.g. distorted reflections of the depth sensor's infrared pattern, the number of points per captured point cloud frame varies between 230k and 270k points in the test sequence. Furthermore, the point sampling density decreases with distance according to the angular resolution of the depth sensor. As the dynamic point clouds are sampled at 30 Hertz, it containts a total of about 7.5 million points per second.

Each uncompressed point is described with 3 float values for x, y, z, for a total of 12 bytes per point (*bpp*). The symbol frequencies of the range coder [15] are encoded with 4 byte integer resolution. In our experiments, we use a single-threaded implementation of the proposed compression technique running on a standard consumer PC. Its source code and documentation can be found in the current version of the Point Cloud Library (PCL) [3].

In Fig. 9, the performance of the octree compression of a single point cloud frame is shown. In Fig. 9(a), the original point cloud is illustrated. Fig. 9(b), Fig. 9(c) and Fig. 9(d) show the compressed point set with 1 mm, 2 mm and 3 mm octree voxel resolution. The alignment of the encoded point cloud with the voxel grid of the octree can be recognized. Here, a data reduction from 12 *bpp* to 0.881 *bpp*, 0.491 *bpp* and 0.380 *bpp* is achieved which corresponds to a compression ratio of 1:14. 1:24 and 1:32, respectively.

Fig. 10 and Table I show the compression performance for the dynamic point cloud test sequence as a function of coordinate precision for the static octree encoding (frames compressed independently) and the proposed double-buffering differential octree encoding, both without point detail encoding. Here, only the octree structure of the first point cloud is intra encoded. Shown in gray bars, the static octree compression reaches on average about 1.75 *bpp* at a point precision of 0.5 mm. By encoding the stream with the differential octree compression method, we significantly reduce the data rate to 1.15 *bpp*, which is an improvement in performance of 34 %. This is mainly due to an increased amount of zero symbols in the serialized octree byte stream (up to 70%) reflecting

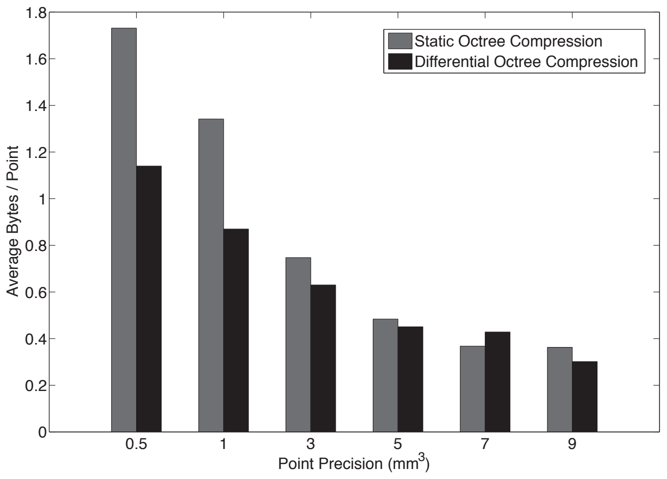
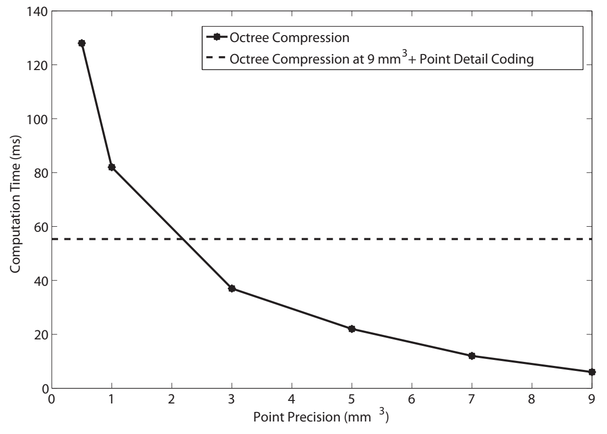
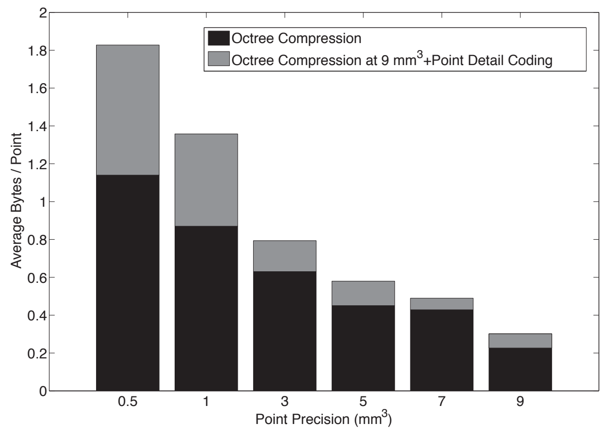


Fig. 10. Comparison of the compression performance using static octrees (each frame compressed seperately) and the differential double-buffering octrees, both without point detail encoding.



(a) Computation time of differential octree coding per frame (point cloud) as a function of point precision is shown by the solid line. The computation time of differential octree coding at a fixed point precision of 9 mm combined with point detail coding is visualized by the dotted line.



(b) The compression performance of differential octree coding at different point precision levels is shown in black. The overhead in compression performance of differential octree coding at a fixed point precision of 9 mm combined with point detail coding is shown in gray.

Fig. 12. Performance analysis of point detail coding.

unchanged branch nodes.

With reduced compression precision, the compression ratio further increases and reaches a rate of about 0.5 bpp at 5 mm and of about 0.3 bpp at 9 mm. Interestingly, the difference in performance between the static and the differential octree compression approach also decreases with reduced coding precision. Here, the data independent overhead originating from the symbol frequency tables of the range coder becomes visible. This is due to the fact that the fixed size of the frequency table encoding dominates the coded stream as the compressed data size decreases, leading to a saturation effect. Compared to the uncompressed rate of 12 bpp, we are able to achieve compression ratios between 9 and 44 for 0.5 mm and 9 mm coordinate precision, respectively.

In Fig. 11, the compression performance per point cloud for the first 300 point clouds (frames) in the stream is illustrated.

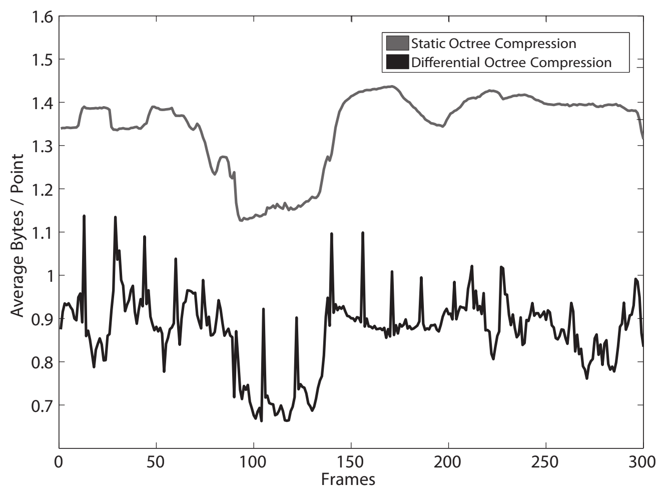


Fig. 11. Compression performance per point cloud frame for the first 300 frames of our point cloud stream.

The gray graph shows the results for static octree encoding, and the differential encoding is depicted in black. Apart from being generally lower, the results for the differential octree coding are also visibly peaked, which is an effect of the motion contained in the test sequence. For quasi-static scenes such as for 3D video conferencing, the occupied bandwidth would be considerably smoother over time. The dip in both graphs between frame number 90 and 140 is due to the person moving closer to the camera, creating fewer points due to occlusion and parallax effects.

We further analyzed the compression performance when encoding additional point detail information (see Fig. 12). In this experiment, we fixed the octree voxel resolution to 9 mm and used point detail encoding to achieve the different target resolutions. This offers two benefits. First, the octree structure occupies considerably less memory because of the reduced number of leave nodes. Second (and closely related), the computation time using point detail encoding becomes independent of the coding precision. This can be seen in Fig. 12(a). Here, the solid line represents the computation time for pure octree-based compression, which follows an exponential progression with increasing point resolutions. The combination of fixed octree compression and point detail coding (dashed line) on the other hand is constant in run time. At 9 mm resolution, we can observe the computation overhead of about $50ms$ when performing the additional point detail compression consisting of the calculation of point distances to voxel origin with subsequent entropy coding. As the complexity of the discretization of extracted voxel-local coordinates is independent from the applied quantization levels, this computation overhead stays constant regardless of the applied coding precision. While for lower resolutions, this constant time is higher than the time required for differential octree compression, there is a "break-even" point at around 2 mm. In addition, Fig. 12(b) shows the corresponding bpp rates as a function of coding precision. Black bars show the

Compression Method	Precision	Bytes / Point	Compression
uncompressed	∞	12 bytes	**1:1**
static octree	0.5 mm	1.73 bytes	**1:7**
	1 mm	1.34 bytes	**1:9**
	3 mm	0.75 bytes	**1:16**
	5 mm	0.48 bytes	**1:25**
	7 mm	0.37 bytes	**1:32**
	9 mm	0.36 bytes	**1:33**
differential octree	0.5 mm	1.14 bytes	**1:11**
	1 mm	0.87 bytes	**1:14**
	3 mm	0.63 bytes	**1:19**
	5 mm	0.45 bytes	**1:27**
	7 mm	0.43 bytes	**1:28**
	9 mm	0.30 bytes	**1:40**

TABLE I
EXPERIMENTAL RESULTS

performance of the differential octree coding and the gray bars represent the overhead involved in using the point detail coding approach. As can be seen in Figures 10 and 12(b), for our test system, the gain in compression performance by using differential octree encoding are roughly canceled out when applying the point detail coding scheme, at the advantage of reaching constant computation times per frame.

The obtained results are to be understood as being qualitative, as they depend on the run-time behavior of the implementation that may vary between parametrization and different hardware setups (core processor, cache sizes, bus, memory, etc.). Note that additional performance could be also achieved by parallelizing the proposed approach. For instance, the compressed frame rate can be doubled relatively easily by creating two compression instances, one for even and one for odd frame numbers. This way, every frame is encoded relative to the second-to-last frame. Both streams can then be processed in parallel, as they are completely independent of each other, and the point differences will likely be only slightly bigger than differences between directly adjacent frames. Alternatively, one could pre-partition the point space into subareas, rendering our approach suitable for e.g. multicore architectures. However, this falls outside the scope of this paper.

VIII. CONCLUSION AND FUTURE WORK

In this paper, a novel online compression framework for unordered point cloud streams is presented. By differentially encoding changes within octree data structures, we can exploit spatial and temporal redundancies between consecutive point clouds. Furthermore, we introduce voxel-local point detail coding which enables online compression with increased point precision at reduced computation and memory requirements. Our experimental evaluation of the proposed compression scheme shows promising performance results at varying coding precision levels.

Future work will concentrate on the integration of entropy-constrained point cloud decomposition, progressive compression of point components and optimal encoding of symbol frequency tables for the entropy coder. Another challenge to be addressed in the future is the compensation of camera motion and moving objects. Furthermore, parallelization on different levels, i.e. spawning multiple threads for the serialization of the octree (distributing individual octree branches), parallelizing the point detail computations, and splitting the single streams into separate, independent, interleaved streams as mentioned are to be investigated.

ACKNOWLEDGMENT

This work was supported, in part, by Willow Garage and, in part, by the DFG excellence initiative research cluster *Cognition for Technical Systems* [1]. We also would like to thank the Point Cloud Library (PCL) team, Armin Hornung, and Matthias Kranz for their valuable contributions and discussions.

REFERENCES

[1] R. Rusu, Z. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3d point cloud based object maps for household environments," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927–941, 2008.
[2] A. Maimone and H. Fuchs, "Encumbrance-free telepresence system with real-time 3d capture and display using commodity depth cameras," in *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality*, Basel, Switzerland, October 2011, pp. 26–29, To Appear in the International Symposium on Mixed and Augmented Reality (ISMAR).
[3] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.
[4] O. Devillers and P. Gandoin, "Geometric compression for interactive transmission," in *Proceedings of the International Conference on Information Visualization*. London, England, UK: IEEE, 2000, pp. 319–326.
[5] J. Peng and C. Kuo, "Octree-based progressive geometry encoder," in *In Internet Multimedia Management Systems IV. Edited by Smith, John R.; Panchanathan, Sethuraman; Zhang, Tong. Proceedings of the SPIE*. Citeseer, 2003, pp. 301–311.
[6] Y. Huang, J. Peng, C. Kuo, and M. Gopi, "Octree-based progressive geometry coding of point clouds," in *Proceeding of the Eurographics Symposium on Point-Based Graphics*, Zurich, Switzerland, 2006, pp. 103–110.
[7] R. Schnabel and R. Klein, "Octree-based point-cloud compression," in *Proceedings of the Eurographics Symposium on Point-Based Graphics*, Zurich, Switzerland, 2006, pp. 111–120.
[8] M. Waschbüsch, M. Gross, F. Eberhard, E. Lamboray, and S. Würmlin, "Progressive compression of point-sampled models," in *Proceedings of the Eurographics Symposium on Point-based Graphics*, Zurich, Switzerland, 2004, pp. 95–102.
[9] B. Merry, P. Marais, and J. Gain, "Compression of dense and regular point clouds," *Computer Graphics*, vol. 25, no. 4, pp. 709–716, 2006.
[10] S. Gumhold, Z. Kami, M. Isenburg, and H. Seidel, "Predictive point-cloud compression," in *ACM SIGGRAPH 2005 Sketches*, Los Angeles, USA, 2005, p. 137.
[11] S. Fleishman, D. Cohen-Or, M. Alexa, and C. Silva, "Progressive point set surfaces," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 4, pp. 997–1011, 2003.
[12] J. Lengyel, "Compression of time-dependent geometry," in *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, Atlanta, GA, USA, 1999, pp. 89–95.
[13] M. Alexa and W. Müller, "Representing animations by principal components," *Computer Graphics*, vol. 19, no. 3, pp. 411–418, 2000.
[14] J. Zhang and C. Owen, "Octree-based animated geometry compression," in *Proceedings of the Data Compression Conference*, Snowbird, UT, USA, 2004, pp. 508–517.
[15] G. Nigel and N. Martin, "Range encoder range encoding: An algorithm for removing redundancy from a digitized message," in *Proceedings of the Video & Data Recording Conference*, Southampton, UK, July 1979, p. 2427.

[1]CoTeSys; http://www.cotesys.org