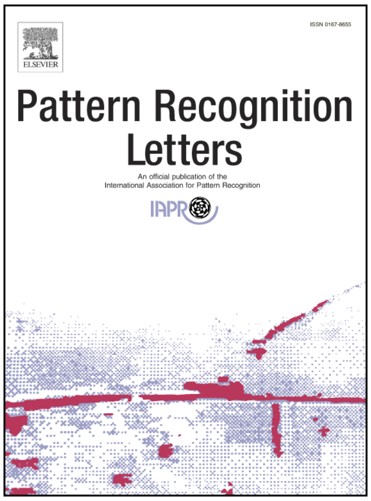

Geometric 3D point cloud compression

Vicente Morell, Sergio Orts, Miguel Cazorla, Jose Garcia-Rodriguez

PII: S0167-8655(14)00172-X
DOI: http://dx.doi.org/10.1016/j.patrec.2014.05.016
Reference: PATREC 6026

To appear in: *Pattern Recognition Letters*

Geometric 3D point cloud compression

Vicente Morell[a], Sergio Orts[a], Miguel Cazorla[a,*], Jose Garcia-Rodriguez[a]

[a]*Instituto de Investigación en Informática*
University of Alicante
PO. Box 99, E-03080 Alicante, Spain

ARTICLE INFO

Article history:

ABSTRACT

The use of 3D data in mobile robotics applications provides valuable information about the robot's environment but usually the huge amount of 3D information is unmanageable by the robot storage and computing capabilities. A data compression is necessary to store and manage this information but preserving as much information as possible. In this paper, we propose a 3D lossy compression system based on plane extraction which represent the points of each scene plane as a Delaunay triangulation and a set of points/area information. The compression system can be customized to achieve different data compression or accuracy ratios. It also supports a color segmentation stage to preserve original scene color information and provides a realistic scene reconstruction. The design of the method provides a fast scene reconstruction useful for further visualization or processing tasks.

1. Introduction

In recent years, the number of applications concerned with 3D data processing has increased considerably due to the emergence of cheap 3D sensors, like Kinect. RGB-D cameras provide useful data, which consist of 3D points with color information. That data could be used in different applications, such as medicine, entertainment industry, robotics, and many others. However, the huge amount of data provided by those cameras is unmanageable in common 3D methods like the Simultaneous Location And Mapping (SLAM) (Durrant-Whyte and Bailey (June. 2006); Bailey and Durrant-Whyte (Sept. 2006)) in mobile robotics field. We propose a 3D data compression method to reduce the amount of information but, at the same time, providing a descriptive objects or

*Corresponding author: Tel.: +34-965-903400 ext: 2992; fax: +34-965-903902;
e-mail: vmorell@dccia.ua.es (Vicente Morell), sorts@dtic.ua.es (Sergio Orts), miguel.cazorla@ua.es (Miguel Cazorla), jgacia@dtic.ua.es (Jose Garcia-Rodriguez)

scene representations. Detailed 3D object representations have proven to be useful in many different applications. However, a large amount of data is required to provide a detailed 3D object model that makes these representations difficult to manage. In this work, we propose the compression on the 3D data, so that the amount of data can be effectively reduced, preserving at the same time as much information as possible. Currently, most 3D object models are visualized using polygonal meshes, since such visualization provides well defined object boundaries and is suitable to visualize the structure of the 3D object. However, many existing 3D data acquisition techniques often obtain 3D point cloud data of the object, which needs to be further processed to obtain correct 3D polygonal surface meshes.

Several 3D compression methods exist in the literature. We can differentiate between lossless methods (the result of the decompression is equal to the original source) and lossy methods. They also can be classified in progressive/non-progressive methods. Progressive methods try to represent the 3D data information incrementally, giving an initial form and increasing the level of detail of the incremental reconstructed 3D data. This is a good approach for systems that have time constraints or interruption on the communication channel.

Most of these approaches work with 3D mesh information primitives (usually triangles and quads) to reduce the amount of information, using different techniques to encode vertex, edges and neighbors primitives. One of the first approaches to encode the connectivity of triangle meshes was the *Edgebreaker* (Rossignac (1999)). The Edgebreaker uses a finite state machine that moves from one triangle onto an adjacent one in a spiral. At each step it encodes whether the tip vertex and the left and right neighbors of the current triangle have already been visited. It encodes the complete connectivity of the triangle mesh in the *clers* string, a sequence of symbols, one per triangle, from the set {C,L,E,R,S}, where each letter indicates the way to compress and decompress the mesh. A trivial, fixed, code guarantees 2 bits per triangle encoding for the connectivity of any manifold triangle mesh without holes or handle. Several variations of this first algorithm were presented (Szymczak et al. (2000); Gumhold (2000); Isenburg and Snoeyink (2001)) in order to reduce the amount of bits per vertex or bits per edge. Other works, like Touma and Gotsman (1998), try to reduce the amount of connectivity information coding the valences (edges) of the points.

The algorithms based on 3D point clouds usually use the spatial organization of the points to encode them in a structure like an

Octree in order to reduce the amount of information. An Octree is a tree data structure in which their internal nodes have exactly eight children. Octrees make a partition of the three dimensional space by recursively subdividing it into eight octants. Another structure similar to the Octree approach is the Voxel Grid (VG). The VG sub-sampling technique is based on a grid of 3D voxels. This technique has been traditionally used in the area of computer graphics to subdivide the input space and reduce the number of points (Connolly (1984); Kobbelt and Botsch (2004)). VG algorithm defines a Voxel grid in the 3D space and for each voxel a centroid is chosen as the representative of all the points that lie on that voxel.

The use of structure like Octrees enables the system to perform some operations like fast searching of the neighbors, reduce the amount of points visualized or get a more or less precise representation of the point cloud. Most of these Octree based methods achieve a lossless compression giving the Octree enough levels to encode the source points as the cell centers of their leaves. This Octree based representation of the 3D data usually allows a progressive compression/decompression of the data because it easily generates a first approximation of the data using the first levels of the Octree.

In Botsch et al. (2002), an Octree based method is presented in order to improve the visualization and operations of 3D transformation and pixel shading. In Schnabel and Klein (2006), a lossless progressive compression method is presented which uses a novel method of position and color prediction of the children nodes points of each Octree cell based on a local surface estimation. Smith et al. (2012) propose another Octree based compression method that uses a marching cube polygonal estimation on each cell of the Octree and a given error tolerance parameter. The method prunes the cell nodes whose plane estimation error is lower than the tolerance parameter. This step allows to reduce the amount of data by coding this information into the compressed data which contains the planes of each cell.

In Peng and Kuo (2005), a 3D mesh compression method is presented that uses an Octree to represent the vertices of the mesh and the connections between them. With this representation, given an Octree depth level, a mesh can be obtained with different resolution and precision. Kammerl et al. (May 2012) propose a spatially and temporal compression of a point cloud stream. It also uses Octrees to encode the occupancy of the space and it allows to control the encoding rate and the encoding precision. It also presents a method to compute the difference between consecutive point clouds by comparing the Octrees of their representation.

Once compared, only the changes of the data are coded and sent. This approach is interesting on controlled scenarios where there are only a few changes between different frames and it provides compression ratios of 40x with a coordinate precision of 9mm.

Following this idea of temporal redundancy; some papers, like Fu et al. (July 2012), use a modification of the traditional video encoding methods to compress the depth images of kinect-like devices. They use a reference frames and use a depth prediction to generate consecutive frames. The results of this lossy method report a compression of 55-85%.

Our method tries to reduce the amount of data of a 3D point set but trying to preserve as much information as possible. Though it can be applied to any scenario, it is designed to work with man-made ones. The proposed method detects planes and represents them with its plane equation, extracting the points belonging to that plane. A color segmentation method is applied to those points in order to keep color information. At this point, we have groups of points belonging to a plane and with similar color information. We extract the border of the group, obtaining a set of points defining a (concave) hull. A Delaunay triangulation of the hull points is applied to decompose the surface of the plane using triangles. That is the information we keep: triangles with their vertex, distribution of the points and color information. The method can be controlled with an initial compression ratio: when the number of processed points is over the ratio, we stop the method. The rest of points in the initial point set is stored with initial coordinates and color information.

For decompressing, we use the information of the triangles and the number of points of the original surface of these planes to generate the reconstruction points inside of their surface, assigning them the stored color information. Then, an uniform point set is generated inside of each triangle getting a similar distribution than the original point set. Finally, the uncompressed point sets are added to complete the 3D point set reconstruction. Figure 1 shows an overview of the compression/decompression system.

The rest of the paper is organized as follows: Section 2 describes the compression step. Section 3 explains how the compression is reverted. In Section 4 we present some experiments and discuss results obtained using our approach and compare it with previous work. Finally, Section 5 draws the conclusions and directions for future work.

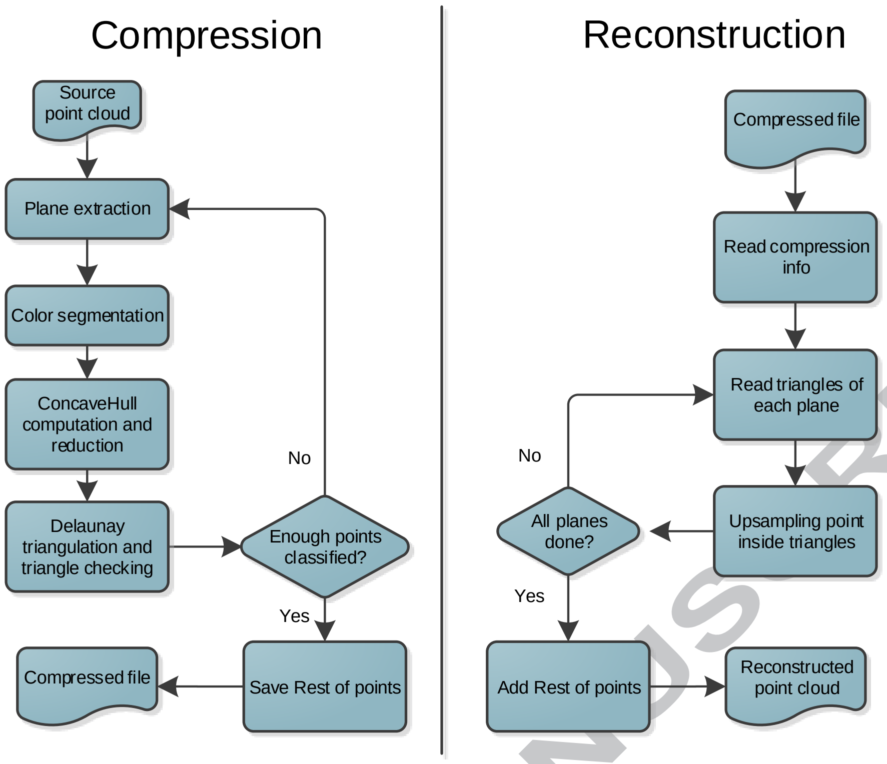
Compression
Reconstruction
Source point cloud
Plane extraction
Color segmentation
ConcaveHull computation and reduction
Delaunay triangulation and triangle checking
Enough points classified?
No
Yes
Save Rest of points
Compressed file
Compressed file
Read compression info
Read triangles of each plane
Upsampling point inside triangles
All planes done?
No
Yes
Add Rest of points
Reconstructed point cloud

Fig. 1. Scheme of the compression/reconstruction proposed system.

2. Compression phase

In the compression phase we process the source point cloud and extract the biggest plane of the current point cloud. Plane extraction is an important part of the compression method because the whole method is based on it. we use a RANdom SAmple Consensus (RANSAC) (Fischler and Bolles (1981)) method to extract the planes from the point cloud. RANSAC is an iterative method to estimate parameters of a mathematical model which, in this case, is a plane model. We introduce a parameter that defines the maximum distance between the points and the plane model. When a plane is detected, the inliers (points that belongs to that plane) are extracted from the source point set. This method uses a distance parameter to determine when a point must be included as a inlier in the plane. Once we obtain a new point cloud representing the inliers points in that plane, we filter the isolated points in order to discard these points in the compression. These discarded points are added to a non compressed point cloud. This filter removes the points which have less than k neighbors in the sphere with radius r. These parameters are fixed empirically based on previous experimentation.

Once a set of points belonging to a plane are extracted, we apply a color segmentation method. If the point set has no color information (for example, data coming from a 3D laser without color information), no further segmentation can be done. So the plane is stored as is. If color information is available, any image segmentation algorithm could be applied, as the points lay in a plane. Using a projection, 3D points can be easily transformed into 2D and then, any segmentation algorithm will work with that points. In our case, we have used the K-means algorithm (Coleman and Andrews (1979)), as it is easy to implement with 3D points, without projecting them to 2D. The K-means clustering method divide the points into K clusters in which each point belongs to the cluster with the nearest mean.

Once the different colored planes are detected, we need to compute the boundaries of each plane segment. These contours may be concave and may have holes. To get these concave hulls we perform an Alpha shapes (Akkiraju et al. (1995)) method that uses an alpha parameter to set the detail level of the resultant hull (the smaller the parameter is the more detailed is the hull). After the edges extraction, we apply an edge reduction step in order to reduce the amount of points needed to represent the plane. This method is proposed in Lowe (1987). Basically, it detects if consecutive points are in the same line and removes them taking only first and last points of that line.

Once we have a polygon of the surface represented by a plane, we triangulate the surface using a Delaunay triangulation. The Delaunay triangulation proposed by Delaunay (1934) results in a set of triangles which satisfy the Delaunay criterion for n-dimensional simplexes (in this case n=2 and the simplexes are triangles). This criterion states that a circumsphere of each simplex in a triangulation contains only the $n + 1$ defining points of the simplex. This triangulation gives triangles whose angles tend to be as bigger as possible or in other words, the triangles tend to be as regular as possible.

After the Delaunay triangulation, we compute the number of points that belongs to each triangle and if this number is lower than a given threshold, the triangle is excluded. While we are detecting the correct triangles of the surface, we compute the area of each triangle for further calculation of the number of points that should be generated inside each triangle. Then, for each plane extracted, we have a set of triangles. The information of the triangles that belong to each plane is stored in a file. For each triangle, we have got the following information: its vertex, the distribution of the points inside the triangle and its color information. In our implementation, we have used a simple segmentation approach, so each triangle has only a color RGB value which is stored with

Algorithm 1 Pseudo algorithm for compression.

```
Require: P3D: 3D point cloud.
Require: percProcess: minimum percentage of points to process.
Require: minNumPointsPlane: minimum number of points in a plane.
Require: relArea: threshold which allows to reject triangles with few points.
Ensure: File containing the remaining points and the planes obtained.
 1: $initialNPoints = nPoints(P3D)$ where nPoints is the number of points.
 2: while $nPoints(P3D)/initialNPoints > percProcess$ do
 3:   Extract the largest plane $P$ from $P3D$ using Ransac.
 4:   if $nPoints(P) < minNumPointsPlane$ then
 5:     Go to 28.
 6:   end if
 7:   Extract the points $Ppoints$ belonging to $P$ and delete them from $P3D$.
 8:   Apply the radius outlier removal to $Ppoints$.
 9:   Perform the color segmentation of $Ppoints$ and get $ColoredPlanes$.
10:   for Each color plane $cp$ in $ColoredPlanes$ do
11:     Calculate the concave hull of $cp$. The result is a contour $C$.
12:     Reduce the contour $C$ into $Cr$.
13:     Find a Delaunay triangulation $Dt$ from $Cr$.
14:     $numPointsPlane = 0$
15:     $areaPlane = 0$
16:     $listTriangles = void$
17:     for Each triangle $t$ in $Dt$ do
18:       Find the number of points $m$ inside the triangle $t$ and its area $A$.
19:       if $m/A > relArea$ then
20:         Add the triangle to $listTriangles$
21:         $areaPlane = areaPlane + A$
22:         $numPointsPlane = numPointsPlane + m$.
23:       end if
24:     end for
25:   end for
26:   Store the plane, formed by $numPointsPlane$, $areaPlane$ and $listTriangles$.
27: end while
28: Store the remaining points from $P3D$ in a file.
```

the triangle. There is not need to use further compression. But if other segmentation method is used, another color coding might be used in order to compress color information. For example, the color entropy method used for JPEG images (Wallace (1991)).

Once all the triangles in a plane are processed, the next plane is processed. The algorithm stops when the compression ratio is reached. This ratio is given by the user and indicates a percentage of processed points with respect to the number of initial points. When that ratio is reached, the remaining points (not processed) are also stored, together with the triangles.

Thus, in the compressed file we save the triangles detected (with its vertex, point distribution and color information). We also save the uncompressed points to accumulate them with the generated point cloud. The complete pseudo algorithm for compression is shown in Algorithm 1.

3. Decompression phase

The reconstruction or decompression part (Algorithm 2) of the proposed compression system is simpler than the compression part. Mainly it takes each triangle saved and generates a proportional number of points to its area assigning the same color to each point in the triangle. Once we have the reconstruction of all the triangles we add the uncompressed points.

Algorithm 2 Pseudo algorithm for decompression.

```
Require: $3DFile$: File containing the 3D points and triangles.
Ensure: 3D point cloud $P3D$.
 1: $P3D$ void point cloud.
 2: Add to $P3D$ the 3D points in $3DFile$.
 3: for Each plane $P$ in $3DFile$ do
 4:     $A$ is the area of $P$.
 5:     $nPoints$ is the number of points in $P$.
 6:     for Each triangle $t$ in $P$ do
 7:         $At$ is the area of the $t$.
 8:         $nPointsToGenerate = (At/A) * nPoints$.
 9:         Randomly generate $nPointsToGenerate$ inside the triangle $t$ and add them to $P3D$.
10:     end for
11: end for
```

Figure 2 shows an example of compression and decompression of a 3D point cloud.

4. Experimentation

We have implemented our method in C++ using some operations of the Point Cloud Library (PCL) (Rusu and Cousins (2011)). Experimentation results are computed on a Intel i5-2320 3GHz. In this section we have developed several experiments in order

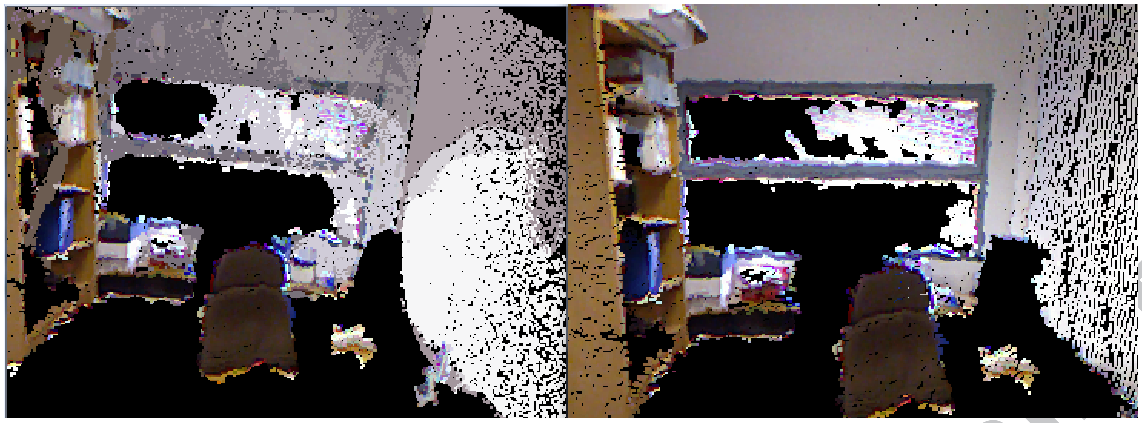

Fig. 2. Reconstruction results with a compression rate of 0.50 with a RMS distance error of 0.0074 and color error of 59.7. Left: reconstructed point cloud; right: original point cloud.

to test the validity of the method. In Section 4.1 we have measured the effectiveness of the method. Section 4.2 presents some experiments with color information.

4.1. Parameter settings

Due to the different algorithms used to obtain the compression of the point cloud, we should establish some parameters in order to achieve good results in terms of compression and quality rates. Some of these parameters are set empirically to obtain good results with man-made environments and point clouds obtained with sensor devices like the Kinect. In this first experiment, we tested our method with different values of the percentage ratio called *percProcess* in Algorithm 1. This parameter allows to determine the desired compression level.

For this experiment, we have selected several synthetic RGB-D data, shown in Figure 3. Kinect has a depth error which might affects the comparison with our method. In some cases, our method provides error reduction in the planes extraction from acquire data. In the situation that the camera is in front of a plane wall, the camera provides a point cloud which forms a plane but with some depth error. Our method is able to provide a plane for that situation (depending of the camera error and the point to plane distance parameter of RANSAC). So, our method will reconstruct a perfect plane point cloud, eliminating the error introduced by the camera.

In order to measure the quality of the results we show the compression rate (the lower value is the most compressed) and the Root Mean Square (RMS) error which represents the average of the distances of the closest points between the source and

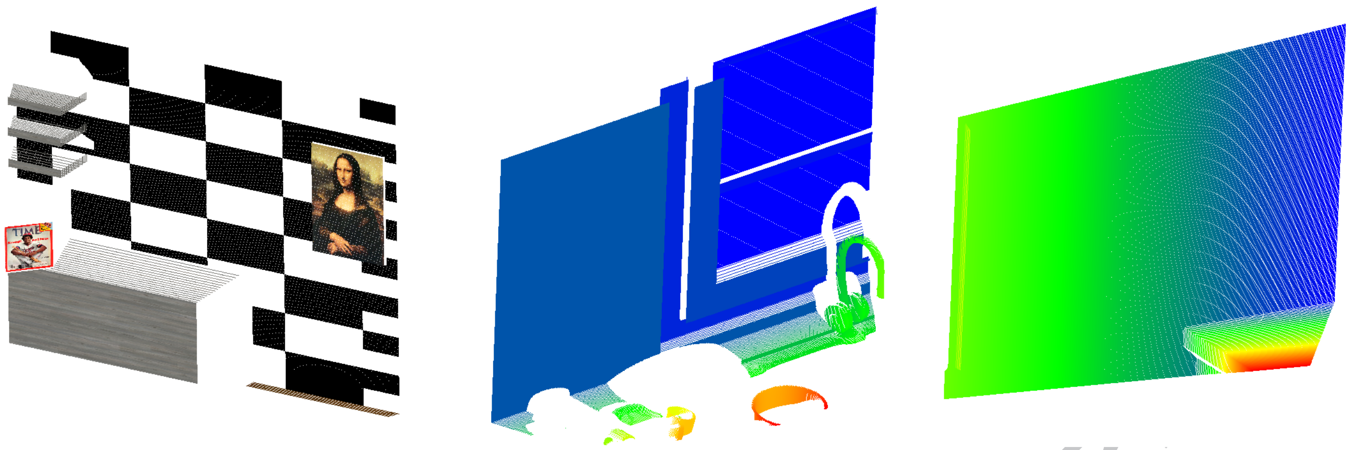

Fig. 3. Point clouds used in the parameter settings experiment.

the reconstructed data file and vice versa. The RMS error measures the quality of the reconstruction which is lower when the reconstruction points are close to the original ones. We compare the results of our algorithm with two state of the art compression methods: Voxel grid and Octree. The Octree implementation uses the center of the occupied nodes as a compression method. The Voxel Grid implementation uses the centroid of the occupied Voxels. Both Octree and Voxel Grid implementations are available in the Point Cloud Library.

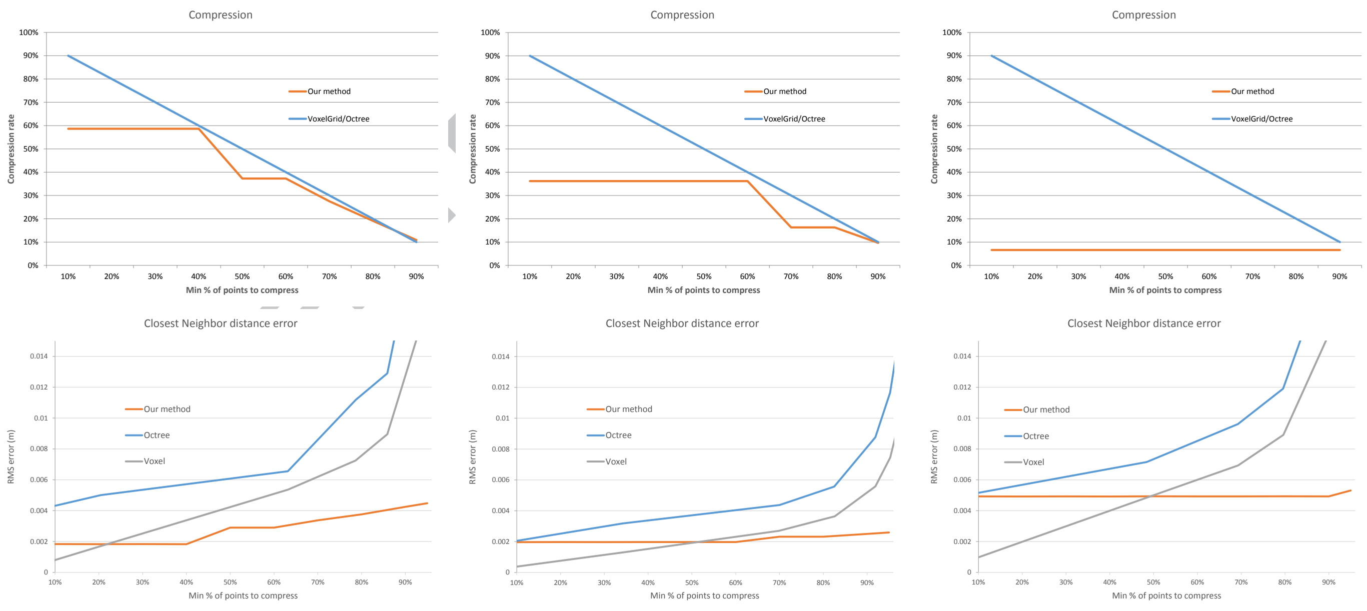

Fig. 4. Results of the compression/reconstruction with different minimum percentages of points to process. Top: compression rate (Octree and Voxelgrid provide the same values). Bottom: RMS error.

Figure 4 shows the comparison of our method with respect to VoxelGrid and Octree methods. We have selected three different scenes, from left with the highest number of planes to right with the lowest number of planes. On the top row,

we present the compression rate relate with the minimum percentage of points to compress. It can be appreciate that as the percentage of points to compress increases (*percProcess*) the compression rate decreases. Voxelgrid and Octree present a linear relationship. This is the expected behavior for these methods. For our method, the behavior is similar to the one obtained with previous methods., but depending of the kind of scene (more or less planes) the obtained compression is different. However, the compression level is lower. In scenes with no planes, the compression rate could be even higher than in scenes with big planes.

Regarding the distance error, in the bottom row of Figure 4, we appreciate how when the *percProcess* increases, the error also increases which is also expected because the fewer points to be compressed, the fewer rate of compression obtained (which means that more points of the original source are saved). Note that although more compression is demanded, the error is almost constant. This is due to the fact that although we ask for more compression, no more planes are founded and then no more compression is done. Therefore, we can conclude that the compression rate and the error are related and we should choose the parameters according to the desired results. Comparing with Octree and Voxelgrid, our method has lower error for compression percentages higher than 50%. This error also depends of the kind of scene to compress (more or less planes). In general, with a maximum error of 5mm in our method, this error is low enough for a real RGB-D camera (more than 5cm at 5 meters).

Both methods, VoxelGrid and Octree, do not have reconstruction phase. Once the point cloud is reduced, they are not able to generate the erased points. They reduce a set of points inside a voxel by a representative point. This point might be the voxel center or the geometric mean of the points inside the voxel. With respect to color, the color of the representative point is the mean of the color of the points inside the voxel. Furthermore, both methods do not store the density of points in the voxel, which is a disadvantage against our method. For example, for those methods, two voxels, one with just one point and other voxel with 100 points, will have only one representative point. Our method stores the density of points inside a plane, so the points generated in the reconstruction phase will have a distribution closest to the original point cloud. Of course, the original point cloud will never be reconstructed, but the reconstructed one will be closer to the original than the

ones in the VoxelGrid and Octree methods.

Figure 5 shows the compression time of our method which is the most time demanding. We observe in Figure 5 (right) that the reconstruction time is small (between 8 and 30 milliseconds) and the main part of the compression time is spent in the file reading part. For the reconstruction phase, Voxelgrid and Octree have a computing time between 10 and 200 milliseconds.

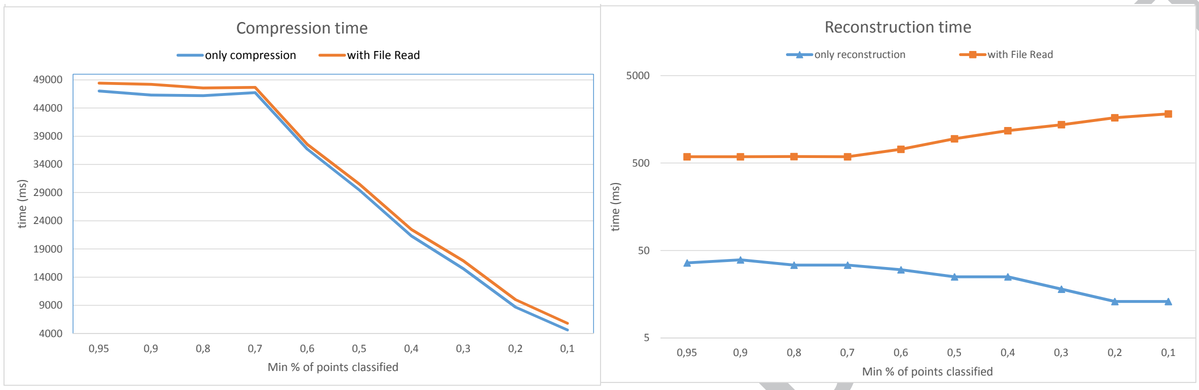

Fig. 5. Compression/reconstruction time. Left: compression time. Right: reconstruction time.

Figure 6 shows different reconstruction results with different ratio of compression. In this case, we have set $K = 1$ for the segmentation step, in order to hide the effects of color segmentation, as we want to check only the effects of ratio compression. When a high compression is demanded (part a), ratio=0.2) the method tries to extract as much planes as possible, resulting in small triangles, since the points supporting the plane are not exactly a plane. For small compression (part c), ratio=0.8) the scene is very similar to the original one.

4.2. Color compression

In this experiment we show the results of color compression using different parameters and how it affects to the compression rate. To measure the color error we compute the average of the euclidean distances of the colors (rgb) of the closest points between the source and generated point clouds similar than the RMS distance error used in experiment 4.1.

Figure 7 shows the color reconstruction using different values for the segmentation process. In the reconstruction process, the polygons are well defined because all the points of the same polygon have the same color. This is expected because the use of the K-means color segmentation discretizes the color into K different colors. The color information reconstructed is enough to easily differentiate the objects in the scene. In applications which need a more accurate color representation (low color error) it should be

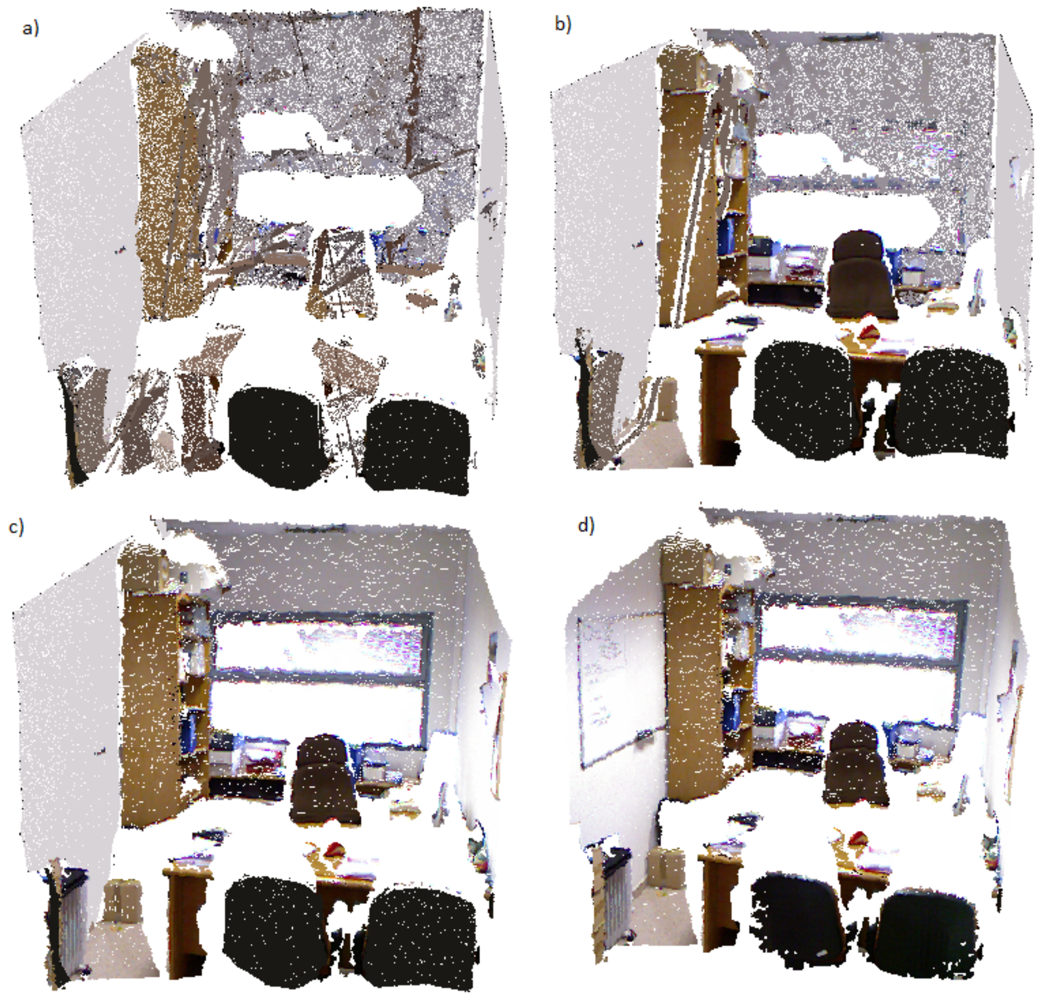

Fig. 6. Different compression of the same scene using a K value of 1 and level compression of 0.2 a), 0.5 b), 0.8 c) and original d).

necessary to apply a more sophisticated and efficient color segmentation and reconstruction methods.

Figure 8 shows and experiment that includes color information. As expected, when increasing the K value of the K-means algorithm, the color error decreases. Besides, the compression is higher when using a lower K, due to with a higher K, a more detailed segmentation is obtained.

4.3. Non frontal point clouds reconstruction

In this last experiment we show how the system works not only on 3D sets generated from a depth image. Due to the design of the system it can be used to compress more complex 3D data sets. In this case, we used a 360 degrees data set. In Figure 9 (left), it can be observed which points are generated inside the polygons and the points that are not compressed. In Figure 9 (right), It can be appreciated the points distributed in line forms while in the reconstructed point cloud, the point are more randomly generated

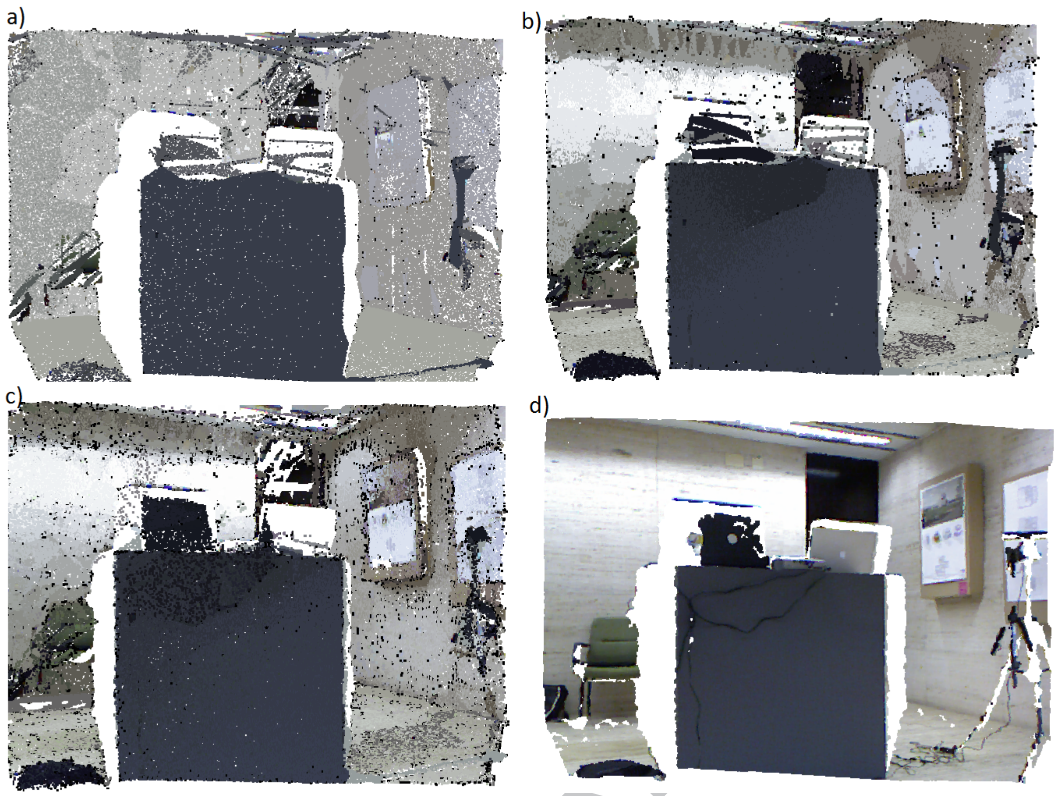

Fig. 7. Reconstruction with several examples using different K values in the segmentation step: a) k=1 b) k= 5 c) k=20 d) original point cloud.

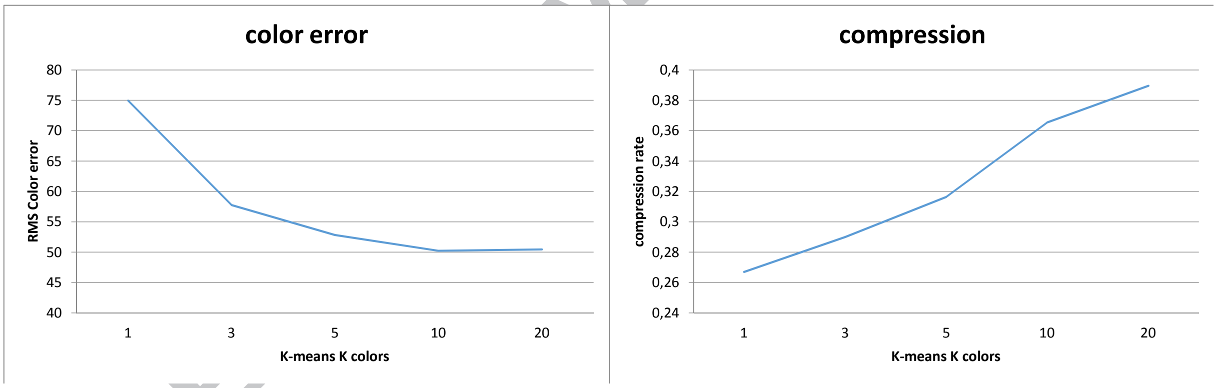

Fig. 8. Experiment using different levels of K in the K-means algorithm.

over its surface.

5. Conclusions

In this paper we have proposed a method of 3D point cloud data compression based on the geometric structure of man-made scenarios which allow the compression of the 3D and color data. The method is able to compress 3D data and reconstruct it keeping the color information provided.

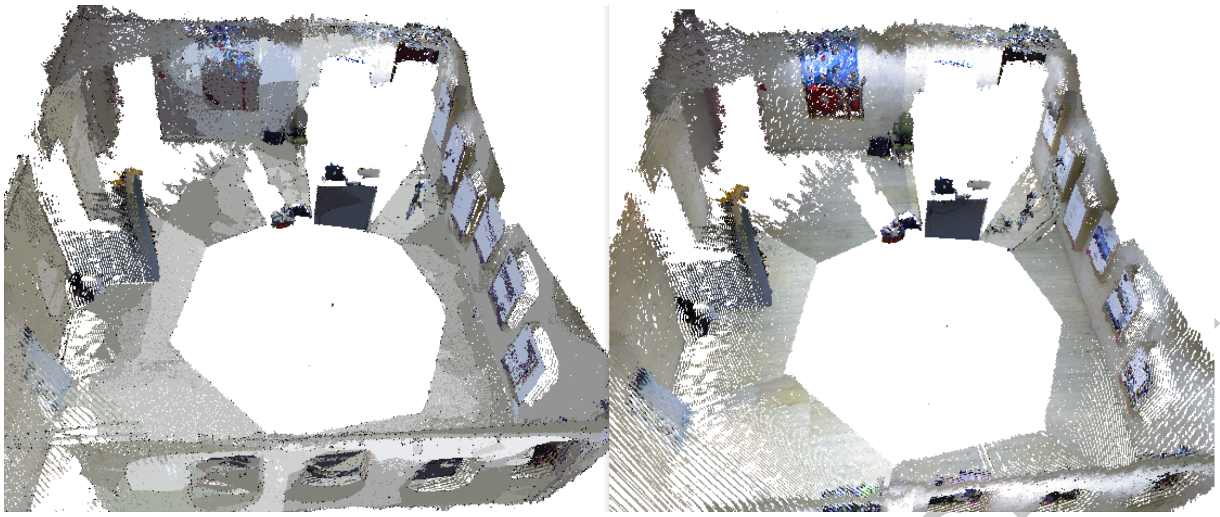

Fig. 9. Reconstruction results of a 360 degrees captured point cloud data. 0.0119886 RMS distance error, 30.1 color error and 0.24 compression ratio (original point set file: 70MB; compressed file: 16,7MB). On the left, the reconstructed point cloud is shown and on the right side the original point cloud.

Experimentation results show that the compression system proposed achieves high compression ratios customized in an interactive way by the users. Due to the plane based compression of the system, the compression rate is better when most of the points of the source data set belong to planes. It is easy to include a color segmentation algorithm to preserve color information. The compressed data composed of a set of planes and other points can be directly used by other applications like 3D registration or 3D data visualization. Our method has a similar behavior compared with Voxelgrid and Octree, with lower error in the reconstruction phase. **In fact, the reconstruction phase is the main contribution of this paper and it is and advantage respect to other lossy methods, cause they are not able to reconstruct an approximated point cloud, like VoxelGrid and Octree.**

As future work, we would like to include another geometric primitives to extract them from the data and not only planes (cylinders, curves, etc.). Another feature to be improved is the plane color segmentation in order to set a globally coherence of colors and/or texture compression information. We also plan to improve the color compression using a color coding similar to the one used by JPEG standard. As our method is time consuming, the implementation of some of the operations on General Purpose Graphic Process Units (GP-GPUs) using technologies as CUDA or openCl should increase the efficiency of the proposed method.

Acknowledgments

This work has been supported by the Spanish Government DPI2013-40534-R grant.

References

Akkiraju, N., Edelsbrunner, H., Facello, M., Fu, P., Mücke, E.P., Varela, C., 1995. Alpha shapes: Definition and software, in: Proceedings of the 1st International Computational Geometry Software Workshop, pp. 63–66.

Bailey, T., Durrant-Whyte, H., Sept. 2006. Simultaneous localization and mapping (slam): part ii. Robotics & Automation Magazine, IEEE 13, 108–117.

Botsch, M., Wiratanaya, A., Kobbelt, L., 2002. Efficient high quality rendering of point sampled geometry, in: EGRW 02: proceedings of the 13th eurographics workshop on rendering, pp. 53–64.

Coleman, G., Andrews, H.C., 1979. Image segmentation by clustering. Proceedings of the IEEE 67, 773–785.

Connolly, C.I., 1984. Cumulative generation of octree models from range data, in: Proceedings of the First International Conference on Robotics and Automation, IEEE. p. 25.

Delaunay, B., 1934. Sur la sphre vide, Izvestia Akademii Nauk SSSR. Otdelenie Matematicheskikh i Estestvennykh Nauk.

Durrant-Whyte, H., Bailey, T., June. 2006. Simultaneous localization and mapping (slam): part i. Robotics & Automation Magazine, IEEE .

Fischler, M.A., Bolles, R.C., 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM 24, 381–395.

Fu, J., Miao, D., Yu, W., Wang, S., Lu, Y., Li, S., July 2012. Kinect-like depth compression with 2d+t prediction, in: Multimedia and Expo Workshops (ICMEW), 2012 IEEE International Conference on, pp. 599–604.

Gumhold, S., 2000. New bounds on the encoding of planar triangulations.

Isenburg, M., Snoeyink, J., 2001. Spirale reversi: Reverse decoding of the edgebreaker encoding.

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

Kobbelt, L., Botsch, M., 2004. A survey of point-based techniques in computer graphics. Computers & Graphics 28, 801–814.

Lowe, D.G., 1987. Three-dimensional object recognition from single two-dimensional images. Artif. Intell. 31, 355–395.

Peng, J., Kuo, C.C.J., 2005. Geometry-guided progressive lossless 3d mesh coding with octree (ot) decomposition. ACM Trans. Graph. 24, 609–616.

Rossignac, J., 1999. Edgebreaker: Connectivity compression for triangle meshes. IEEE Transactions on Visualization and Computer Graphics 5, 47–61.

Rusu, R.B., Cousins, S., 2011. 3D is here: Point Cloud Library (PCL), in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China.

Schnabel, R., Klein, R., 2006. Octree-based point-cloud compression, in: Botsch, M., Chen, B. (Eds.), Symposium on Point-Based Graphics 2006, Eurographics.

Smith, J., Petrova, G., Schaefer, S., 2012. Smi 2012: Full progressive encoding and compression of surfaces generated from point cloud data. Comput. Graph. 36, 341–348.

Szymczak, A., King, D., Rossignac, J., 2000. An edgebreaker-based efficient compression scheme for regular meshes.

Touma, C., Gotsman, C., 1998. Triangle mesh compression, in: Graphics Interface 98 Conference Proceeding.

Wallace, G.K., 1991. The jpeg still picture compression standard. Commun. ACM 34, 30–44.

- Our main goal is to compress and decompress 3D data using geometric methods.
- The proposed method extracts planes and makes color segmentation.
- The result from segmentation is triangulated and triangles stored.
- Thus, we can reach great ratio compression with low color and point loss.
- It's designed to work with man-made scenarios,but can be applied to any general one