Journées Françaises d'Informatique Graphique et de Réalité Virtuelle, Marseille, 12-15 novembre 2019

Room modeling: Encoding Point Cloud into a piecewise planar model and heightmaps

Edmond Boulet-Gilly[1], Géraldine Morin[1], Carsten Griwodz[2], Simone Gasparini[1]

[1]IRIT - University of Toulouse
[2]UiO - University of Oslo

Résumé

Depth scanners enable modeling of an environment with points in space positioned where the laser beam meets a surface. In many cases, those points suffer from a lack of precision, occlusions, heterogeneous density, on top of being part of a large dataset. In this article, we introduce a pipeline able to take as input a point cloud captured in a room and obtain as output a clean, faithful and lightweight 3D mesh that models the structure of the room. Initially we model the environment by a set of planar faces, cycles of a 3D graph made of nodes, edges and faces estimated from the point cloud. Next, by generating heightmaps, we express points of each planar component as vertices displaced from a plane. Those heightmaps are registered as 2D images that can be processed depending on the architecture of the environment. This approach, in addition to proposing a high-definition meshing solution, opens the point cloud processing domain to image processing, thus providing good compression rates and smoothing of imperfections due to depth scanners.

Mots clés : JFIGRV 2019, IRIT, Oslo University, Point Cloud, Lidar, HeightMap, Compression, Meshing

1. Introduction

Indoor scene modeling has been an active research subject in recent years. It finds applications in various domains such as augmented reality, architecture, computer-aided design or interior design. Manual reconstruction by a human operator with generic modeling software can be a very time-consuming and tedious task and thus several approaches have been proposed to either completely automate the process or speed up and ease the manual modeling process.

Image-based reconstruction techniques [FH15] generate 3D models from a set of unordered pictures [SSS06]. Furukawa [FCSS09] showed a technique using stereo-vision to produce depth maps and then provide user exploration of the 3D-reconstructed environment. Unfortunately, the methods have some limitations as they do not support non-axis aligned structures. More in general, stereo-vision techniques [DSEFW11] must solve the correspondences among interest points, and thus are not adapted for textureless objects like, typically, walls.

The use of depth scanners is on the rise in multiple industries, including entertainment, autonomous driving, housing construction and maintenance. As demand grows, so does the need for precision, quality and efficiency. Thus, new techniques are needed to efficiently process the generated point cloud data to produce practical and efficient models as the data often contains redundant information, is affected by measurement noise and by artifacts due to the merging process. Also efficient 3D representations are needed to create 3D visualisation renders to ease the computational and the memory burden.

In indoors applications, the main impact is capturing and validating to verify that interiors conform to construction standards [BCFL12] but also to keep track of deteriorations.

The last generation of depth scanners, such as LIDAR (LIght Detection And Ranging) sensors or even consumer grade phones equipped with SLAM frameworks for augmented reality applications [Goo, App], can generate very dense 3D models.

In this paper, we address the problem of generating a compact representation for a 3D model of a room using heightmaps starting from 3D point clouds recorded with a LIDAR. To represent the room, we

assume that the scene satisfies the Manhattan world hypothesis [CY00]: this enables us to simplify the model and consider only the main planar elements of the room such as the walls, the floor and the ceiling. Then we interpolate the data along those main planar components and compress that interpolation into heightmaps. This allow us to open the issue to image processing and use cutting-edge work to achieve lightweight and clear results by using these heightmaps as displacement maps to compute efficient and coherent 3D models.

The paper is organized as follows: §2 reviews the representation methods for 3D models that can be found in the literature and presents the novelty of the proposed approach. §3 introduces the proposed pipeline and §4 details the implementation of the heightmaps. §5 proposes some experiments that validates the proposed approach and §6 concludes the paper with some perspectives.

2. State of the art

Point clouds collected from sensors in closed rooms usually contain unlabeled data. This is why a segmentation process is usually applied as a first step. As Okorn et al. [OXAH10] showcase, it is possible to detect structure in a room by establishing a Z histogram that helps detecting the walls and an XY histogram that highlights the boundaries on the room.

This feature analysis works fine when the goal is only to draw a map of the room but is limited by the amount of furniture in the room as well as suffering from the impossibility to work on tilted surfaces.

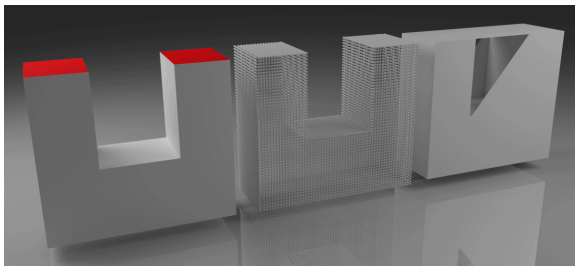
The fact that each indoor structure component is connected to neighbor components leads to the following idea: make a graph that is representative of the structural properties of the room and deduce the exact placement of the walls, the ceiling and the floor, and compute the vertices and edges that connect them.

Chen et al. [CC08] show the benefit of establishing an adjacency graph to ensure the coherence in terms of topology. Each element of the room appears in the graph as a vertex and the edges of the graph indicate which elements are connected.

But this approach is limited because a simple planar segmentation with little sense of coherence will not satisfy the requirement to establish a relevant graph. For instance performing a plane fitting on a structure with discontinuous planar features will return a wrong graph as shown in Fig. 1. In addition, the graph approach is unable to feature small details.

Other work by Xianfeng Gu et al. [XG02] shows us that it is possible to encode detailed geometry into images and, by taking advantages of efficient compression ratio provided by the PNG format for instance, we can reach light and faithful result. But this approach is dedicated to scenarios where we already have access to the mesh topology.

Figure 1 : *U-Shape Structure (Left) transformed in point cloud (Middle) and reconstructed with a graph approach (Right) but the segmentation failed as two planes (in red) were not separated in the graph.*

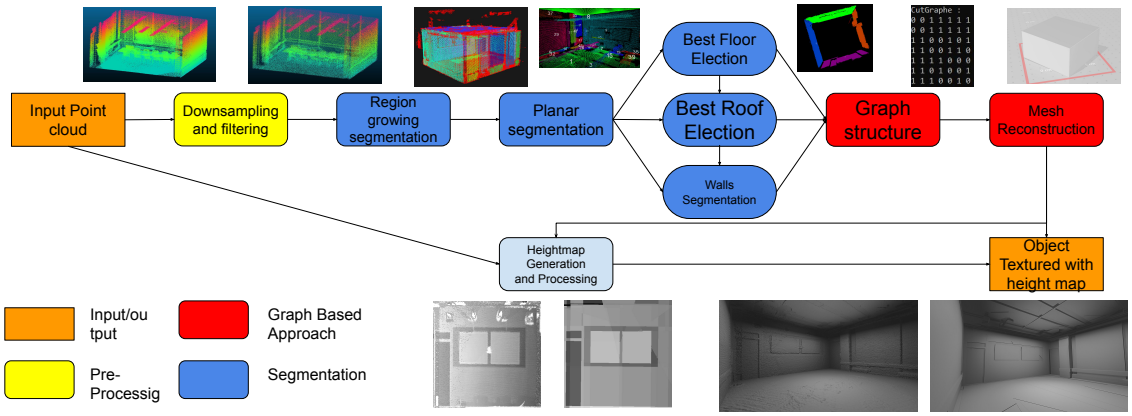


Work made by Charles R. Qi et al. [CSKG17] showcases the PointNet, a technology that can help inject unprocessed point cloud into a pipeline of neural network that will automatically perform the segmentation of an object and even be able to classify a set of some objects. This approach is however limited the amount of different object used during the training and does not tackle the issue of compressing the point cloud or introducing 3D Meshing.

In terms of scene representation, the common way to achieve a simplified representation of an indoor scene is to use higher level geometric primitives such as planes rather the original point cloud. RAPter algorithm [MMBM15], for example, starts from a dense point cloud and provides a final parametric representation of the data by fitting the data with planes and imposing parallelism and perpendicularity constraints to such planes. Similarly, [CF14] obtains an initial floor plan from a set images via SfM, then it generates high quality texture-mapped models for free-viewpoint visualization by re-projecting the images. These representation are compact and can be provide visual information if combined with images, but they lose the details of the initial point cloud.

Our approach aims at providing a compact representation of the scene yet preserving (some of) the geometric details of the initial reconstruction. Our scene representation is inspired to the representations that can be found in “bump mapping” [Bli78] and “displacement maps” [OBM00] techniques in computer graphics. Both techniques are used to improve the photo-realism of the rendering adding fine geometric details to textured surfaces. While the first simulates small geometric details of a flat textured surface by affecting the normals, the latter simulate the details by adding heightmaps (or displacement maps) to the actual geometric position of the points over the textured surface. We use heightmaps to reduce the memory footprint of the 3D representation, thus providing a relatively light model that preserves the details of the initial point cloud.

Figure 2 : *Our Pipeline for Point Cloud Processing focuses on Room Reconstruction*



2.1. Proposed Approach

A closed room's main features are its planar boundaries; thus, we detect (and compress) the walls, floor and ceiling. Walls, floor and ceiling bound the room and have a high probability to represent point sets that have a high density.

In a second step, we replace them with a simpler model, like a surface represented by 4 vertices and 4 edges, potentially replacing thousands of points.

This also allows us to filter the points scanned through transparent windows that are located outside the room.

These closed planar mesh are adjacent to each other and we code this 3D Mesh box as a graph as proposed by Chen et al. [CC08]. Since the point cloud might be not complete and suffer from occlusion, a graph-based architecture makes for a strong starting point to develop other features. But in order to do that, we need to detect all relevant planar components in our point cloud. We initiate the segmentation with a region-growing method, ensuring that each plane resulting from the plane segmentation is linked to exactly one main planar component fed into the graph approach. Our graph structure is kept to the most simplistic entities so that it can be applied to a wide variety of scenarios (faces for the planes, edges are intersections of two planes, and corners/vertices are intersections of 3 planes).

After detecting the walls and replacing them with simple faces, we encode the features lost by approximation using heightmaps.

The operation opens up the domain of image processing and we are able to generate a high resolution version of the room as well low resolution version.

This approach allows us to compute objects that can reconstruct both parts of the point cloud and a 3D mesh, while requiring minimal space on the disk, as we only need the base mesh and PNG heightmaps to compute our end result. The end result provides a bounding box to filter aberrant points, a faithful 3D mesh and changes fundamentally the way our point cloud is encoded on disk, making it an efficient compression method.

3. Creating a base piecewise planar model

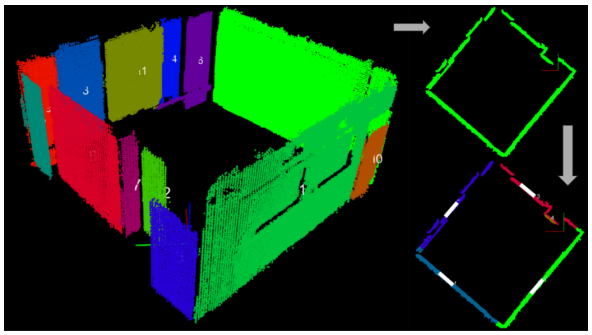
3.1. Planar component Segmentation

The top part of the pipeline in 2 is dedicated to establishing the base 3D Mesh with extended planar faces to apply the heightmap. On top of estimating the most significant planar structures in the point cloud, figuring out the thickness of each planar component represents how much information relative to the points corresponding to this plane will then be coded into the corresponding heightmap.

Our first goal is to figure out the structure of the room, we use the Manhattan world assumption [CY01] to assume that our room is mainly made of planar components and they intersect in right angles. To reduce the study of the room to those components, we perform a region growing segmentation based on normals to regroup points by location and then perform a plane fitting segmentation on each region. The last segmentation returns the plane equation in intercept form's coefficient for each planar component and ensures that each region returned by the first segmentation is made of one or multiple planes.

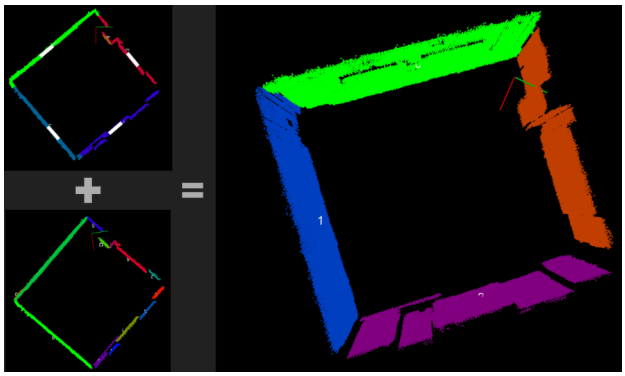
Among the most relevant planar components identified, we then elect floor, ceiling and walls based on size and orientation. Only the floor and the ceiling are considered made of one plane with a given arbitrary thickness. First, we suppose that the point cloud mostly well oriented, so the floor is the biggest plane whose normal is mostly along the Z axis and the ceiling is the biggest plane facing the floor. The walls are the planes orthogonal to the floor and ceiling. We compute the number of walls by performing a rough line segmentation on the projection of those points on the XY plane as proposed by Antonio et al. [AH11] (see Fig. 3) and merge the result of the plane segmentation with

Figure 3 : *Left: Only the biggest planes oriented along the X and Y axis are considered as walls. Top Right: All planes merged into one point cloud and projected along the XY plane. Bottom Right: Line segmentation performed in order to give a rough idea of the wall's location.*



the line segmentation to obtain a final segmentation where every plane is attributed to a single wall based on their location and orientation. This step gives us a good idea of the location and thickness of each wall, as shown in Fig. 4.

Figure 4 : *Top Left: Result of the linear segmentation giving us the number of walls and their orientation. Bottom Left: Result of the planar segmentation giving us the location and orientation of each sub planar component. Right: Result of our Wall segmentation deduced from merging the two previous segmentation.*



Last we perform a gross plane fitting estimation on each wall to compute the final planar coefficient indicating their orientation where extreme accuracy is not required because of the subsequent heightmap computation.

3.2. Connecting the planar components

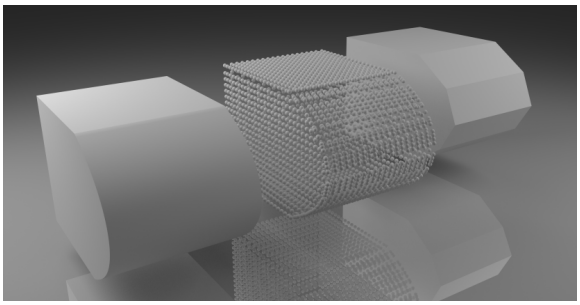
We now have access to the main planar components' orientations and their corresponding points. This section is dedicated to establishing the base, manifold mesh that will support the texture coding of the points. We first establish a graph connecting two intersecting planar components (among the identified main planes). At least, each wall intersects both the floor and the ceiling. Next we establish which two walls intersect: for that, we study both the orientation (using with the plane equation) and request that the intersection is close to the planes, in order to solve occlusion issues but avoid computing intersections that are out of range (like two parallel planes intersecting at infinity).

After determining the pairs of planes that intersect, we define the graph and its geometric embedding as follows:

- Three planes intersecting pairwise intersect at a single vertex,
- Two planes that intersect create an edge linking the two vertices on that edge.

We end up with a graph with vertices and edges. By traversing the graph and grouping vertices located in a common plane, we detect the corresponding face by identifying elementary cycles. We encode the elementary mesh in an OBJ file of our 3D mesh with the UV maps (texture wrapping coordinates indicating how to apply textures to each face) directly embedded so that we are able to apply our heightmaps directly after rendering them. Note that our graph approach creates only flat faces/surfaces. However, it can also handle scenarios with curved geometry, these parts will be approximated by flat surfaces as illustrated in 5.

Figure 5 : *Left: a structure with a curved side. Middle: the same structure sampled into a point cloud. Right: the structure reconstructed with our graph approach.*



3.3. Bounding Box Computation

The graph approach returns a closed, base mesh piecewise linear to model the room delimiting the structure, providing an excellent tool to filter out the points outside the room. In scenes with windows or containing surfaces that do not reflect LIDAR beams completely, it is needed to detect points outside the room and delete them. As illustrated in 7, projecting the points onto the base mesh allows to differentiate the ones that lie outside from the ones that are inside the mesh.

Figure 6 : *Left: a non convex piecewise linear structure. Middle: same structure sampled into a point cloud. Right: the structure reconstructed with our graph approach.*

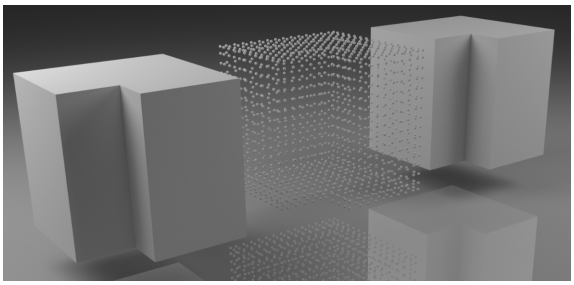
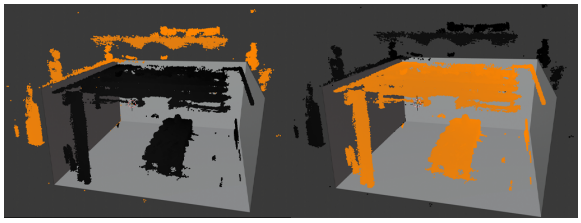


Figure 7 : *Differentiating point outside the closed, base mesh (left) and the points inside the room (right).*



4. Defining Points as Heightmaps

4.1. Heightmap Generation

The previous steps defined the main planar components in the point cloud, their relative position and their thickness. In this section, we propose to encode the point's location relative to a corresponding planar component using heigthmaps.

For a given plane and its corresponding 3D points, we apply a transformation to map the plane to (X, Y) coordinates, and express the point's third coordinate on the z-axis, that is, the displacement relative to the reference plane. When storing the z-coordinates of the points into a heigthmap on the plane, that is, as an image, or texture, we sample (X, Y) on a pixel grid $(\widetilde{X}, \widetilde{Y})$ for a chosen resolution depending on the density of the point cloud. If no point corresponds to a given $(\widetilde{X}, \widetilde{Y})$, the pixel is kept to 0 for now. On the other hand, more than one point may correspond to the same pixel, in which case we have to combine different Z values on this pixel. We have tried setting the displacement to the maximum, the average or the median of the Z values (see Fig. 8). As expected, taking the median leads to the most robust approximation.

In order to code pixels that do not contain any points, we generate an opacity map to help differentiate coordinates with a height of 0 from coordinates that did not have any point. The opacity map allows us to characterize the pixels with no data so they can be deleted as it shows improved results in the performance evaluation section.

Figure 8 : *Heightmaps representing points relative to a wall, where points mapping to the same pixel are combined taking their maximal (left), average (middle), or median (right) Z values.*

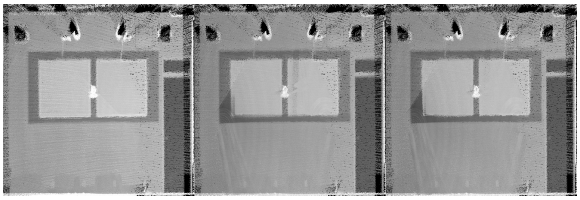
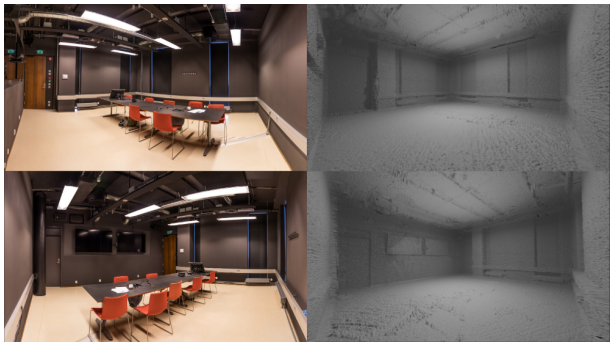


Figure 9 : *An example of a room (left) and its digital model coded by our base 3D mesh and heightmaps applied as displacement, for two different viewpoints (top and bottom).*

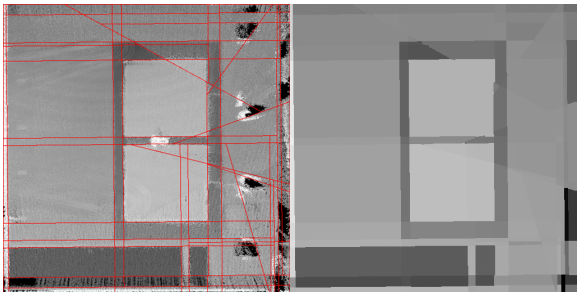


4.2. Post-Processing Heightmaps

A main advantage of our approach is to open the domain of point cloud processing to image processing as it is a widely studied field. Our goal is to both reduce the noise and also reduce the amount of geometry. We achieve that by post-processing the heightmaps.

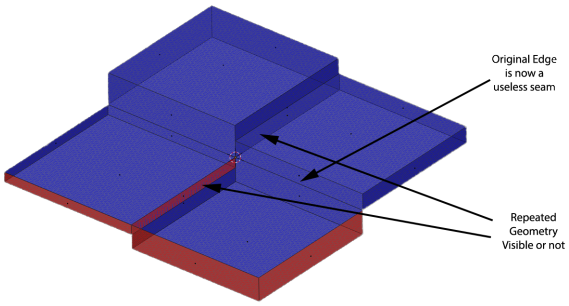
We also propose to compensate for missing data. In order to fill in the black pixels due to missing data, performing a convolution based on the neighbor pixels allows us to smooth areas that were not well sampled. This approach works well when the missing pixels are scattered and not grouped; it does not smooth the noise.

Figure 10 : *Left: a heightmap and the Kippi partitioning overlaid. Right: each part of the heightmap is assigned its the median value.*



Regarding the noise, we implemented multiple approaches like applying a Bi-Linear filter [Inc19], or smoothing based on superpixels [ASS*12], in order to smooth uniform areas and keep sharp seams between two different height. However, none of the mentioned method can reach the level of simplicity and fidelity we are looking for. We turned our choice to the work of Bauchet et al. and their implementation of Kippi [BL18]. The Kippi (KInetic Polygonal Partitioning of Images) [BL18] approach consists of first performing a segmentation based on segments on each heightmap and then we use it, not as a displacement map, but directly as a geometry where each area has a different height. Because the features of the average room can be broken down into lines, it is relevant to use it here.

Figure 11 : *Issues appearing when extruding each area to the corresponding height.*



The process is a bit more complex than just applying the heightmaps as displacement. We need to extrude each face to the corresponding height, but doing so with an inadequate implementation of the extrude operation creates an undesirable geometry as seen in 11. This is why we implemented our own extrude operation followed by a planar decimation to reach the most simple and clean result.

Listing 1 : *Extrude Geometry while keeping clean tolopology.*

```
Select whole plane
Move to zero height
While max height not reached
        Extrude to current min
           height
        De-Select current plane
           min height
        Delete current Min
           Height
Perform planar Decimation
```

5. Implementation and Experiments

In this section, we evaluate the results of our approach compared to classical methods for the point cloud compression and for surface reconstruction. We analyze the compressing performances and the data fidelity of our proposed method. Finally, we sum up the limitations and robustness of our approach.

Because our approach only tackles the shape of the room and the features of the wall, our reference point cloud for comparing the fidelity and compression is only made of points that constitute the main planar components of the room (floor, ceiling and walls) not the filtered points, that are neither the objects in the room (chairs and table) nor the points outside the room (outliers).

Our algorithm performance is compared to voxel downsampling, a method that creates a 3D grid of voxel and keeps only one point by voxel, if there is one in the reference point cloud. This method is more or less aggressive depending of the size of each voxel, we tested it on voxel of size 0.16 (smaller voxel, more points kept, label *Downsampled016*), 0.30 (label *Downsampled030*) and 0.72 (bigger voxel, less points kept, label *Downsampled072*).

Our final 3D mesh is compared to Poisson surface approximation [KBH06] for continuous surface reconstruction. We computed different approach of Poisson surface approximation, one based on the full reference point cloud (label *Poisson* in the figures 14 and 16) and one based on a voxel downsampled by 0.8 version of the reference point cloud (label *PoissonDownsampled*).

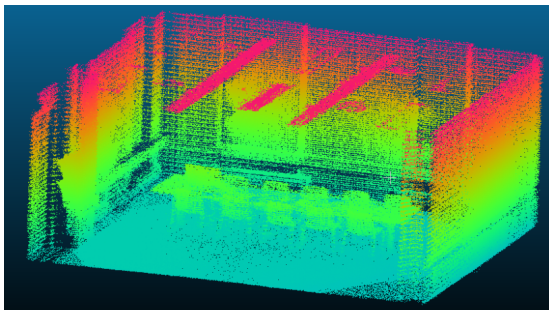
We choose to show different results for our approach, based on heightmaps and reconstructed 3D mesh. For our heightmap implementation we included the 3 possibilities for the interpolation of the Z-value (label *Median*, *Average* and *Max*) with and without being processed with a hole filling convolution detailed in Section 4.2 (label *Median Processed*, *Average Processed* and *Max Processed*) or using the opacity map to filter out missing pixel (label *Median Opacity*). In the 3D mesh reconstruction evaluation we show the performance of the kippi implementation (label *Kippi*).

5.1. Experimental setting

The pipeline is tested on real-world data collected by Sillerud and Johanssen [SJ19] with a LIDAR Scanner. They scan a single room with several scans. By registering these different views they managed to get a complete scan of a room at UiO called Black Box. The point cloud weighs 800 MB and is made of 25.41 Millions points.

The room holds 6 planar components: 4 walls, 1 ceiling and 1 floor. The floor is flat and has few features but needs to be separated from the chairs. The walls are mostly flat but feature minor seams and major beams. The ceiling is made of a fake roof occluding

Figure 12 : *The point cloud we experimented on with the points corresponding to the top plane removed for visibility.*



most of it and there are also some minor features like pipes.

In [SJ19], the authors analyze the error of the LIDAR and concluded that the noise on points increases with distance to the scanner. Because the point cloud is made out of different scans, the capture of a 3D point in a given scan is more or less accurate, but in the other sub-scans the same point is more than likely at a different location. The resulting scan contains multiple layers of each wall at different locations due to this specific issue.

We choose to encode our heightmaps at the resolution 512x512, considering the density of our point cloud, and on 8-bit values, meaning they only support between 0 and 255 different height between the minimum and maximum depth of the main planar component. It is a relevant range of value for the floor that is thin but it might be relevant to switch to 16 or 32 bit encoding when the wall features more extremes and different values.

The heightmap and the opacity map could be generated in only one image with different channels but we choose to render them separately as some software we choose to use did not support multi channel images.

5.2. Application to Compression

By expressing our points as being part of a plane, we manage to obtain good compression rates as we now need minimal information to establish their location. For instance, as shown in Fig 13 storing a point in a PCD (point cloud data) file requires to store the three coordinates X, Y and Z, whereas our approach requires to store data of each plane for a large set of points and only one coordinate (on 8 bits) per point for the displacement value. Thus, the efficiency of the compression depends on the numbers of planes and points per plane stored. In our experimental setting, storing the points to compress the point cloud is extremely efficient.

As we can see in 14, Poisson surface approximation average a byte per vertices is 96 meanwhile we have a byte per vertices of less that one for the displacement

Figure 13 : *Size of points cloud (no geometry) with voxel downsampling and our proposed approach with an original point cloud of 25 millions points. Left column and axis is size in KB, Right column and axis is the amount of points.*

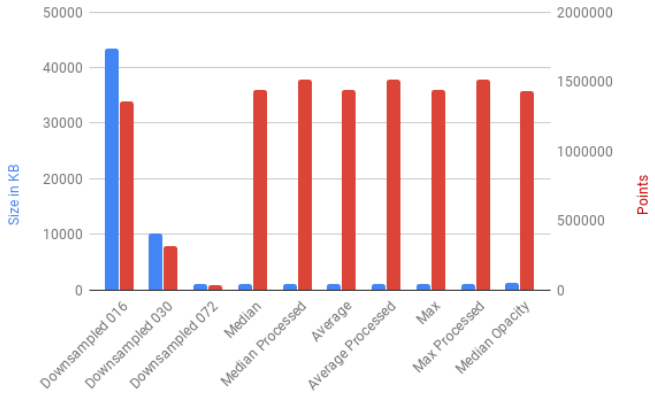
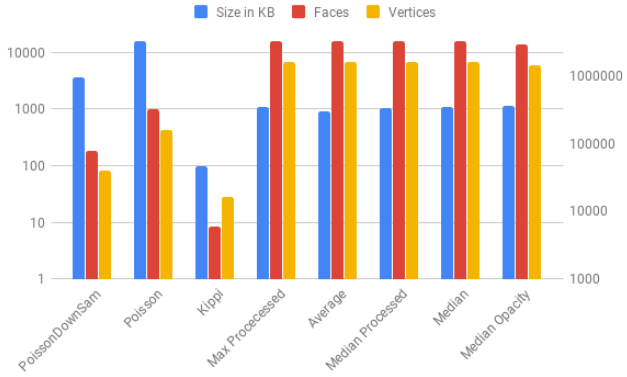


Figure 14 : *Size in KB (log scale on the left axis, blue column) of reconstructed surface with Poisson surface approximation and the mesh resulting from our approach and geometry amount (right axis, red and yellow column).*



map implementation and 9 for the Kippi implementation.

Overall our approach allow us to store much more 3D information, than the traditional PLY file who stores edges, faces and vertices separately.

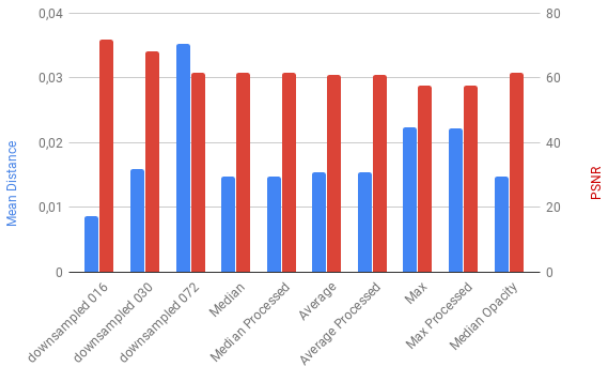
To conclude regarding the compression, by using the heightmaps to store both points and geometry we reach detailed point cloud or 3D mesh while taking minimum disk space much more efficiently than traditional PCD or PLy format.

5.3. Approximation of the original point cloud

Regarding the approximation, we used Cloud Compare to compute standard deviation PNSR and Mean

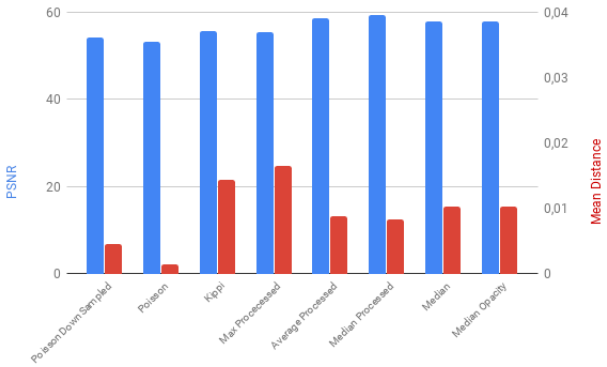
distance between both point cloud to point cloud, by comparing the original to the one obtained by our approach and voxel downsampling, but also point cloud to 3D Mesh, by comparing the original to 3D mesh obtained by our approach and Poisson surface approximation.

Figure 15 : *PSNR and Mean Distance of points cloud with Voxel downsampling and our approach.*



As Fig. 15 shows, our approach reaches the same PSNR as the most aggressive Voxel downsampling while performing much better in terms of mean distance.

Figure 16 : *PSNR and Mean Distance of Mesh with Poisson surface approximation and our approach.*



As 16 shows, we manage to reach higher PSNR than Poisson surface approximation but score a bit higher in mean distance while staying in a low range of value.

5.4. Visual Performance

The most impressive aspect of our approach relies on the Kippi implementation that allows us to compute a model with extremely light geometry but high fidelity.

Figure 17 : *Left: The room where the point cloud was taken. Right: Our 3D Mesh extruded following the Kippi Segmentation.*

5.5. Robustness

The robustness of our approach is limited by the amount of noise in the input point cloud. Here we manage to get relevant result without post-processing the point cloud where each LIDAR scan had an accuracy of ±3 cm according to [SJ19].

The heightmap implementation does not alter the data, it merely changes the way the points are encoded. The points that are reconstructed after the displacement are shifted only due to the sampling resolution in the 1 byte encoding of the displacement and the interpolation along the depth axis. Either way, if we fail to detect the main planar component orientation when we perform the rough plane fitting segmentation, the 3D mesh appears at the right location and in the right orientation as the displacement adapts; 18 shows how tilting the heightmap might result in missing some points (in red) but the points that are still in range are encoded and decoded at the same location.

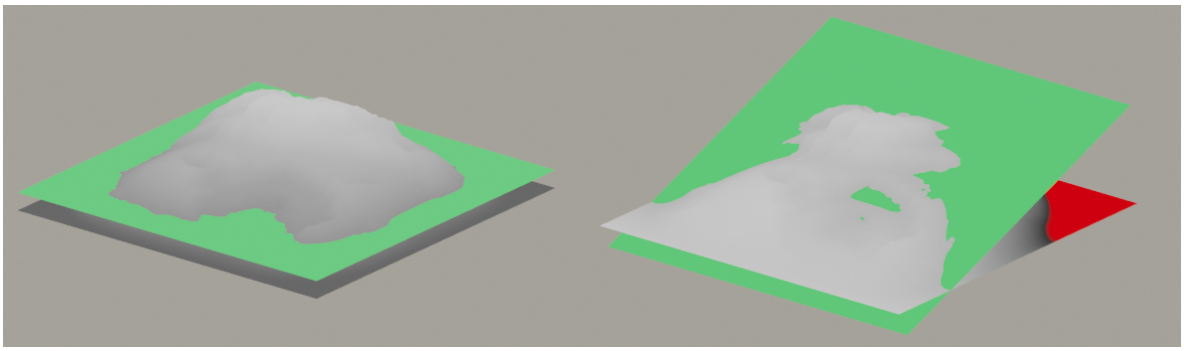
5.6. Limitations

The limitation of our approach lies in the quality of the point cloud and our ability to separate main planar component point. If the walls are not straight for instance, the projection on the plane XY returns points scattered on the the plane and the linear segmentation will fail, making us unable to detect the walls and deduce the 3D mesh. In some scenario it might be better to feed a basic plane segmentation directly into the graph approach if the planes left are only the main planar components, like in Fig. 5.

Our current implementation of the graph approach does not support triangular rooms as 3 walls might intersect pairwise but not in a single point. There are also many scenarios where if the graph is not accurate, vertices, edges and faces will be created unnecessarily as seen in Fig. 1.

We would also like to point out that main planar components of the room are treated separately but

Figure 18 : *Left: Plane Oriented along the structure, taking every point. Right: Plane Tilted, some point are out of the thickness reach but the point position relative to the plane did not changed and the color changed to balance the displacement*



share a lot of common points where they collide, which leaves a lot of room for optimization to merge the edges of each planar component properly to create a manifold model and close the room.

Another issue is that each wall has a thickness that expresses every point within an estimated distance determined by the segmentation. This implies that if there is a chair positioned close to the wall, but a beam on the other extremity of that wall that maximizes the evaluated thickness, the chair will probably be absorbed into a heightmap as a wall feature. This might be fixed by generating a thickness mask instead of a fixed thickness that stays constant along the entire wall.

The issue of the thickness gets more problematic when considering that some features might not be planar and they might not be taken into account when computing the thickness estimation. Past a certain point, geometric criteria can separate what point constitutes a wall feature from an object too close to the wall, indicating that only a semantic approach will solve the issue.

The Kippi segmentation works great in our scenario as the wall features are planar and can be split in multiple lines, but if the walls have features that cannot be simplified into areas, the Kippi segmentation will just lose all the fine detail and only smoothing the heightmap will return satisfying result. In such case, a second pass can help taking care of non linear parts.

6. Conclusions

In the end, we proposed and implemented complete pipeline for creating an alternative, light model from a large, noisy point cloud. The result shows good compression rate, a robust and performing approximation as well as providing both low- and high-resolution solutions. We acknowledge that the technology has some limitations regarding the room configuration, the amount of noise and the feature segmentation but we remain optimistic that further work can build upon and improve the robustness of our approach.

We are confident that as LIDAR technology improves, our approach will stay relevant and return even better results, as our current output is already usable for visual interpretation and rendering of 3D rooms.

Références

[AH11] Adan A., Huber D.: 3D reconstruction of interior wall surfaces under occlusion and clutter. In Proceedings of the 2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission, 3DIMPVT 2011 (2011).

[App] Apple: Apple ARKit: SDK for augmented reality. https://developer.apple.com/augmented-reality/.

[ASS*12] Achanta R., Shaji A., Smith K., Lucchi A., Fua P., Süsstrunk S.: Slic superpixels compared to state-of-the-art superpixel methods. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 34, Num. 11 (Nov 2012), 2274–2282.

[BCFL12] Bhatla A., Choe S. Y., Fierro O., Leite F.: Evaluation of accuracy of as-built 3d modeling from photos taken by handheld digital cameras. Automation in Construction. Vol. 28 (décembre 2012), 116–127.

[BL18] Bauchet J. P., Lafarge F.: KIPPI: KInetic Polygonal Partitioning of Images. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2018).

[Bli78] Blinn J. F.: Simulation of wrinkled surfaces. SIGGRAPH Comput. Graph.. Vol. 12, Num. 3 (août 1978), 286–292.

[CC08] Chen J., Chen B.: Architectural modeling from sparsely scanned range data. International Journal of Computer Vision. Vol. 78 (07 2008), 223–236.

[CF14] Cabral R., Furukawa Y.: Piecewise planar and compact floorplan reconstruction from images. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2014).

[CSKG17] Charles R., Su H., Kaichun M., Guibas L.: Pointnet: Deep learning on point sets for 3d classification and segmentation. pp. 77–85.

[CY00] Coughlan J. M., Yuille A. L.: The manhattan world assumption: Regularities in scene statistics which enable bayesian inference. In Proceedings of the 13th International Conference on Neural Information Processing Systems (Cambridge, MA, USA, 2000), NIPS'00, MIT Press, pp. 809–815.

[CY01] Coughlan J. M., Yuille A. L.: The manhattan world assumption: Regularities in scene statistics which enable Bayesian inference. In Advances in Neural Information Processing Systems (2001).

[DSEFW11] De Silva D., Ekmekcioglu E., Fernando W., Worrall S.: Display dependent preprocessing of depth maps based on just noticeable depth difference modeling. Selected Topics in Signal Processing, IEEE Journal of. Vol. 5 (05 2011), 335 – 351.

[FCSS09] Furukawa Y., Curless B., Seitz S. M., Szeliski R.: Reconstructing building interiors from images. Proceedings of the 2009 IEEE 12th International Conference on Computer Vision (sep 2009), 80–87.

[FH15] Furukawa Y., Hernández C.: Multi-view stereo: A tutorial. Found. Trends. Comput. Graph. Vis.. Vol. 9, Num. 1-2 (juin 2015), 1–148.

[Goo] Google: ARCore: SDK for augmented reality. https://developers.google.com/ar/discover/.

[Inc19] Inc. T. C. L. C.: Computer Desktop Encyclopedia. 1981-2019. 10 2019.

[KBH06] Kazhdan M., Bolitho M., Hoppe H.: Poisson Surface Reconstruction. In Proceedings of the Eurographics Symposium on Geometry Processing (2006) (2006).

[MMBM15] Monszpart A., Mellado N., Brostow G. J., Mitra N. J.: RAPter: Rebuilding Man-made Scenes with Regular Arrangements of Planes. ACM Trans. Graph.. Vol. 34, Num. 4 (2015), 103:1–103:12.

[OBM00] Oliveira M. M., Bishop G., McAllister D.: Relief texture mapping. In Proceedings of SIGGRAPH (July 2000), pp. 359–368.

[OXAH10] Okorn B., Xiong X., Akinci B., Huber D.: Toward automated modeling of floor plans. In Proceedings of the Symposium on 3D data processing, visualization and transmission (2010), vol. 2.

[SJ19] Sillerud V. S., Johanssen F. K.: Reconstruction of Indoor Environments Using LiDAR and IMU. Master's thesis, University of Oslo, Department of Physics, Faculty of mathematics and natural sciences, 2019.

[SSS06] Snavely N., Seitz S. M., Szeliski R.: Photo tourism: Exploring photo collections in 3d. In SIGGRAPH Conference Proceedings (New York, NY, USA, 2006), ACM Press, pp. 835–846.

[XG02] Xianfeng Gu Steven Gortler H. H.: Connectivity-free resampling of an arbitrary shape into a regular 2d grid. ACM Trans. Graphics (SIGGRAPH), 21(3), 2002 (3 2002).