Acta Polytechnica CTU Proceedings **30**:126–130, 2021
DOI:10.14311/APP.2021.30.0126

available online at http://ojs.cvut.cz/ojs/index.php/app

AUTOMATED BIM ENTITY RECONSTRUCTION FROM UNSTRUCTURED 3D POINTCLOUDS

Jan Voříšek*, Bořek Patzák, Edita Dvořáková, Daniel Rypl

Czech Technical University in Prague, Faculty of Civil Engineering, Department of Mechanics, Thákurova 7, 166 29 Prague 6, Czech Republic

* corresponding author: jan.vorisek@fsv.cvut.cz

Abstract. Laser scanning is used widely in architecture and construction to document existing buildings by providing accurate data for creating a 3D model. The output is a set of data points in space, so-called point cloud. While point clouds can be directly rendered and inspected, they do not hold any semantics. Typically, engineers manually obtain floor plans, structural models, or the whole BIM model, which is a very time-consuming task for large building projects.

In this contribution, we present the design and concept of a *PointCloud2BIM* library [1]. It provides a set of algorithms for automated or user assisted detection of fundamental entities from scanned point cloud data sets, such as floors, rooms, walls, and openings, and identification of the mutual relationships between them. The entity detection is based on a reasonable degree of human interaction (i.e., expected wall thickness). The results reside in a platform-agnostic JSON database allowing future integration into any existing BIM software.

Keywords: 3d model, BIM, laser scanning, point cloud.

1. Introduction

The Building Information Modelling (BIM) brings excellent opportunities for the efficient and more automated design process of building structures and maintenance and reconstruction planning of existing structures. Obtaining a digital BIM representation of the latter is often a challenging task, namely when no up-to-date (or none at all) floor plans are available.

Laser scanning is a convenient and accessible technology to document the existing infrastructure. The output is a dense set of data points in space, so-called point cloud. See Figure 1 for an example of a laser-scanned building. Each point carries its spatial coordinates, and some scanners also capture the RGB color value and intensity of the scanned point. While point clouds can be directly rendered and inspected, they carry no additional information about the scanned objects. There are no available tools to convert the raw point data into a BIM representation with a reasonable degree of automation, although research in this field is highly emerging [2, 3].

Currently, the civil engineers have to manually identify the BIM entities from graphical visualization of point cloud and create their digital representation in BIM model. The long-term goal of the project is to develop an efficient and user friendly tool for semi-automated conversion of point clouds to BIM entities while maintaining user interaction. The idea is based on automated identification of entities, where each entity is labeled with estimated degree of identification reliability. The individual identified entities are presented to the user who can accept all suggested entities recognized with reliability higher than defined threshold and manually accepting/modifying remaining suggestions. The presented paper describes the concept of *PointCloud2BIM* library [1] for point cloud manipulation and BIM entity identification.

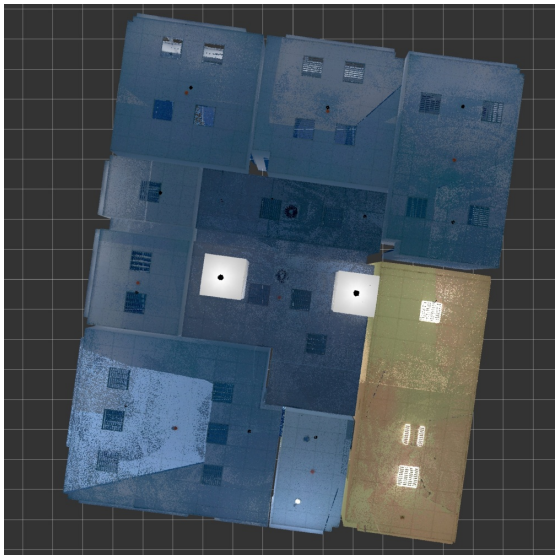

Figure 1. Laser scan of an unfurnished office building, consisting of 85M points, top view.

2. Pre-processing of the point clouds

Laser scanners produce very dense point clouds containing tens or hundreds of millions of points. The distance between the scanned points often ranges in millimeters. Therefore the point cloud data processing must be handled carefully in order to achieve competitive computation times and memory efficiency.

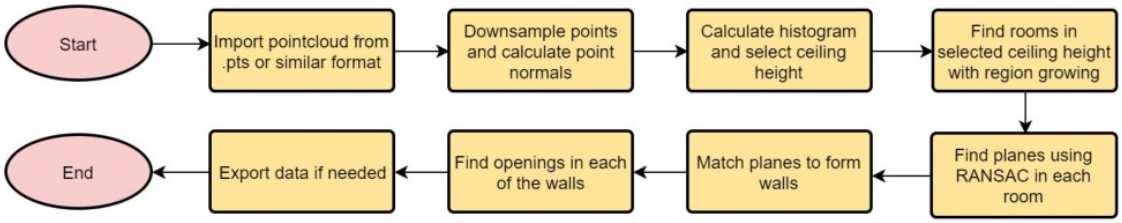

FIGURE 2. Wide figure.

2.1. STORAGE

Point information such as the XYZ coordinates, RGB color, or intensity value is stored in an uncompressed binary file. The main advantage is that binary data can be directly read into C++ structures instead of the costly parsing of text files line-by-line.

All the other project data reside in the project's JSON file, which serves as a database of point cloud metadata and other entities. The JSON format is widely supported across platforms and programming languages and was selected to allow smooth integration into existing software. An empty project file is created automatically before the first point cloud import.

2.2. DOWNSAMPLING

While storing the points as binary data offers high reading speeds, with hundreds of millions of points, reading all the values into memory might not be possible on a standard computer with limited memory. For example, storing 100 million XYZ coordinates (3x64 bit float) occupies 2,4 GB of memory, which is more than half of a commonly available 4GB.

Reducing the number of points to work with is inevitable. In our case, the points were downsampled in a 3D voxel grid using the efficient implementation of a C++ hash map [4]. Reasonable selection of a grid step of say $1 \times 1 \times 1$ cm reduces all the points within the 1 cm^3 cube to a single point with weighted coordinates.

Result of downsampling to a $1 \times 1 \times 1$ grid is summarized in the Table 1.

#	Before	After
1	85 316 410	12 358 178
2	62 947 576	20 213 992
3	108 200 686	34 672 667

TABLE 1. The effect of downsampling on three typical point clouds — the number of original points and the number of points after downsampling.

One can see that just by downsampling to a 1 cm^3 grid, the size of points goes down to at least 30%.

2.3. THE NORMAL VECTOR OF DOWNSAMPLED POINTS

Further processing can take advantage of determining the normal vector at each downsampled point (voxel). This is done in the three following steps.

A single point represents each voxel by averaging the coordinates of the inner points. There are up to 26 ($3 \times 3 \times 3 - 1$) voxels surrounding each voxel if all the neighbors are activated. Therefore, the total number of points n for the normal evaluation is at most 27.

The centroid of the n points is calculated as a weighted average of the XYZ coordinates.

$$\mu_x = \frac{\sum_{i=1}^{n} x_i}{n}, \mu_y = \frac{\sum_{i=1}^{n} y_i}{n}, \mu_z = \frac{\sum_{i=1}^{n} z_i}{n}, \quad (1)$$

The 3×3 covariance matrix of the point XYZ coordinates relative to the centroid is calculated using the formula for the covariance of A and B below.

$$\mathrm{cov}(A, B) = \frac{\sum_{i=1}^{n}(a_i - \mu_a)(b_i - \mu_b)}{n - 1} \quad (2)$$

The smallest Eigenvector of the covariance matrix is the plane normal [5]. The direct solver of eigenvalues was implemented for the 3×3 matrix using the arccos function, as described in [6].

3. BIM ENTITY EXTRACTION

The overall strategy of point cloud transformation into the digital BIM representation of the building is based on a gradual segmentation of floors, rooms, walls, and openings. The overview of this complex process is demonstrated in Figure 2.

3.1. FLOOR SEGMENTATION

The floor segmentation is based on the histogram of vertical coordinates. From the histogram, the ceiling and floor levels are extracted, corresponding to locations with significant location frequency. More accurate results can be obtained by only considering points with vertical normal in the calculation. The ceiling level can also be input manually by inspecting the point cloud in a suitable tool like Autodesk ReCap.

Figure 3. Results of the room segmentation of the point cloud shown in Figure 1.

Figure 4. Vertical plane segmentation (top view). Door planes are segmented as they cover sufficient area, also notice the noise around outer windows and in the center room.

3.2. Room segmentation

The ceiling level of the floor is known either from the histogram calculation or from user input. All the ceiling plane points and points within 30 cm below (as suggested in [3]) are projected onto a 2D grid (floor plan) resulting in a pixel-like image. The step of the grid should be at least twice the step used for the initial downsampling, or the grid will contain missing points as some points may not be appropriately captured.

Region growing [7] is applied to join adjacent pixels into separate regions. Backfilling of each of the found regions is performed to include columns and skylights within the room. Knowing the pixel count and pixel area (step x step), area of the subject rooms or whole floor can be estimated.

Finally, the original point cloud is iterated, and points inside of the found regions (in terms of X, Y coordinates) and floor (given Z coordinate range) are selected. Separate point clouds of each room are stored in the project file. An example of room segmentation results can be seen in Figure 3.

3.3. Vertical plane segmentation

The wall planes are identified in each room using the modified Random sample consensus (RANSAC) [8]. Only points with horizontal normal are considered for plane finding, a slight deviation of 5 degrees is allowed to account for deviations in construction.

Each wall finding starts with hypothesis generation. Three non-collinear points A, B, C are randomly selected from the point cloud for which the four plane coefficients a, b, c, and d are calculated using the two vectors $\vec{AB}$ and $\vec{AC}$

$$\vec{AB} = (B_x - A_x, B_y - A_y, B_z - A_z) \quad (3)$$

$$\vec{AC} = (C_x - A_x, C_y - A_y, C_z - A_z) \quad (4)$$

To satisfy the plane equation $ax + by + cz + d = 0$, plane coefficients are calculated as follows

$$a = (B_y - A_y)(C_z - A_z) - (C_y - A_y)(B_z - A_z) \quad (5)$$

$$b = (B_z - A_z)(C_x - A_x) - (C_z - A_z)(B_x - A_x) \quad (6)$$

$$c = (B_x - A_x)(C_y - A_y) - (C_x - A_x)(B_y - A_y) \quad (7)$$

$$d = -(aA_x + bA_y + cA_z) \quad (8)$$

After establishing the hypothesis, more points are selected randomly, and the distance from an arbitrary point to the plane is calculated. Points within sufficient distance are considered in the support set. For the first 100 accepted points, the plane coefficients are refined using the algorithm described in Section 2.3.

3.4. Wall segmentation

Wall segmentation is performed on each of the vertical planes found in the previous step. Only walls of constant thickness are supported. Therefore, comparing the angle between every two planes and the distance between the two subject planes is sufficient to mark wall segment candidates.

For each wall candidate, its points are projected to a single 2D plane. Such a transformation can be done simply without the transformation matrix. The normal vector is taken from plane segmentation by calculating the average value of all wall planes. Local X-axis is selected by picking two plane points A, B, and clipping their vector 's Z coordinate. By knowing the local X-axis and the normal vector, the Y-axis vector can be calculated using the cross product. This is sumarized below.

$$X_{loc} = (B_x - A_x, B_y - A_y, 0) \quad (9)$$

$$Y_{loc} = n \times X_{loc} \quad (10)$$

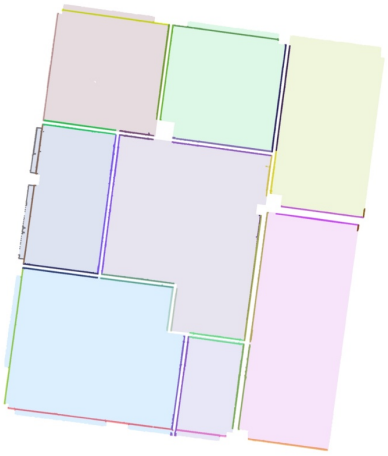

FIGURE 5. Wall segmentation (top view).

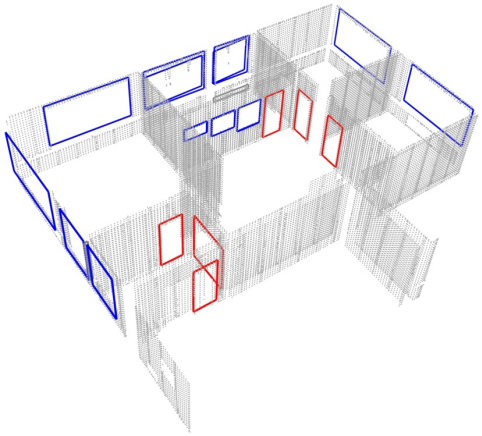

FIGURE 6. Opening detection — doors (red), windows (blue).

The wall local axes X_{loc} and Y_{loc} are then normalized, and for each wall point P, the local coordinates are calculated using the dot product.

$$P_{x,loc} = (P - A) \cdot X_{loc} \tag{11}$$
$$P_{y,loc} = (P - A) \cdot Y_{loc} \tag{12}$$

Plane points lie in two intervals $[a, b]$ and $[c, d]$ in the local X-axis direction. The distance o in the wall direction in which the planes are overlapping is calculated

$$o = [\max(a, c), \min(b, d)]$$

Wall candidates are filtered based on the user's overlap requirements, as partially overlapping planes are likely to belong to a common wall.

3.5. Opening segmentation

A naive way of finding openings in walls (on two parallel planes) is to find overlap of missing points on both planes. However, such treatment produces a lot of false results. For example, two cabinets on each side of the wall in the same place do not form an opening.

Points present between the walls can be taken into consideration to improve the accuracy of opening detection. By a combination of inverse region growing and boundary detection, openings with a sufficient density of intermediate points can be identified.

By calculating the bounding box of an opening in the local plane coordinates, dimensions can be calculated for rectangular openings (i.e., doors and some windows). Doors are recognized by comparing their bottom coordinates to the corresponding floor level. Opening visualized by their bounding box can be observed in Figure 6.

4. Demonstration on a sample point cloud

The example of a simple office building with just a single floor with eight rooms is selected to demonstrate the capabilities of the presented approach. As the whole floor was unfurnished, the overall floor envelope has little to none missing points except for the areas behind doors and heating elements. The original data set consists of 85 316 410 points is shown in Fig. 1.

At first, the original point cloud was downsampled into 12 358 178 using a $1 \times 1 \times 1$ cm grid. By inspecting the point cloud, we identify the floor and ceiling levels, and we use all the points in between ceiling level and 30 cm below for room finding. The room finding performed on a 2D floor plan grid with a 2×2 cm step yields all the eight identified rooms, visualized in Figure 3. In the same step, we also identify all the 93 vertical planes, as can be seen in Fig. 4. The planes assigned to their source rooms are then passed to the next step for wall matching even if some of the identified planes might be out of our interest (i.e., doors or noise).

In Figure 5, we can observe all the matched walls on the subject floor. While all the inner walls were identified, the exterior walls are missing as the exterior scan of the office building was not available. However, some of the external walls are obtained thanks to the scan capturing the window frames' internal and external planes. These reveal the estimated window positions as well as exterior wall location, excluding its wall thickness.

5. Conclusions

The library was implemented in C++ based on the previously formulated methodology. It can be inte-

grated into any BIM software to assist the conversion of point clouds into digital BIM representation. The library API is documented in the Reference Manual [9]. The library also works as a standalone toolkit without any dependency on BIM software. A set of command-line executables is available to perform the individual steps introduced above. Each of the tools operates on top of a common database-like file (JSON) with point data stored separately in binary files.

A single floor of an unfurnished office building was successfully segmented into separate rooms (compare Figures 1 and 3). Plane matching results can be seen in Figure 4 with the 25 distinct wall segments considering the exterior windows as walls. The success of opening detection can be examined in Figure 6 where six doors were identified and eight exterior windows (two of which are found twice).

6. Future work

Our main goal is to create a robust and easy-to-integrate solution of converting the raw point cloud files into the corresponding BIM entities. While our library provides enough functionality to process, analyze, and visualize the point clouds and calculated results, more research is desired, especially in the area of opening detection and overall confidence rate evaluation of the reconstructed BIM entities.

Further research is needed to integrate the library into some of the industry-leading BIM applications (i.e. Autodesk Revit). Such integration is required to validate and refine our methodology using a variety of point clouds from different environments.

Acknowledgements

The development of this software was supported by the Technology Agency of the Czech Republic under the program of the National Competence Center 1 as a project "Center for Advanced Materials and Efficient Buildings" (CAMEB) - project registration No. TN01000056 and by the Grant Agency of the Czech Technical University in Prague (SGS project No. SGS20/038/OHK1/1T/11), both gratefully acknowledged.

References

[1] PointCloud2BIM library for automated identification of bim entities from point clouds. `http://mech2018.fsv.cvut.cz/pc2bim`. Accessed: 2020-08-06.

[2] C. Wang, Y. Cho, C. Kim. Automatic bim component extraction from point clouds of existing buildings for sustainability applications. *Automation in Construction* **56**, 2015. doi:10.1016/j.autcon.2015.04.001.

[3] H. Macher, T. Landes, P. Grussenmeyer. From point clouds to building information models: 3d semi-automatic reconstruction of indoors of existing buildings. *Applied Sciences* **7**:1030, 2017. doi:10.3390/app7101030.

[4] G. Popovitch. The parallel hashmap or abseiling from the shoulders of giants. `https://greg7mdp.github.io/parallel-hashmap/`. Accessed: 2020-04-14.

[5] E. Ernerfeldt. Fitting a plane to noisy points in 3d. `https://www.ilikebigbits.com/2017_09_25_plane_from_points_2.html`, 2017. Accessed: 2020-04-14.

[6] S. Hartmann. Computational aspects of the symmetric eigenvalue problem of second order tensors. *Technische Mechanik* **23**:283–294, 2003.

[7] M. Montoya, C. Gil, I. García Fernandez. Implementation of a region growing algorithm on multicomputers: Analysis of the work load balance., 2000.

[8] M. A. Fischler, R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM* **24**(6):381–395, 1981. doi:10.1145/358669.358692.

[9] J. Voříšek, D. Rypl, B. Patzák, E. Dvořáková. Pointcloud2bim library: Pointcloud2bim library reference manual. `http://mech2018.fsv.cvut.cz/pc2bim/www/refman/html/index.html`.