2021 European Conference on Computing in Construction
Ixia, Rhodes, Greece
July 25-27, 2021

VOID-GROWING: A NOVEL SCAN-TO-BIM METHOD FOR MANHATTAN WORLD BUILDINGS FROM POINT CLOUD

ABSTRACT

The automated generation of 3D models of buildings from point clouds is still under heavy research. Currently, this Scan-to-BIM process requires high manual effort, and the previous research focuses on simple room structure under low occlusion level. We propose a novel "void-growing" approach that extracts walls, floors, and ceilings automatically. Different from the majority of current approaches starting with detecting surfaces of elements in point clouds, our approach grows the void volume space inside a room first. Our approach performs well in occluded environments. It can reconstruct simple cuboid rooms as well as complex rooms like L-shape and U-shape rooms. Different ceiling heights caused by suspended ceilings can also be represented properly.

INTRODUCTION

The popularity of Building Information Modeling (BIM) is penetrating the construction sector. However, only few existing buildings have reliable BIM models. There are mainly two reasons for this situation. The first one is that many facilities have no pre-existing digital models from when they were constructed. The other one is that the digital design model was not updated when the asset was modified through the lifecycle of the asset. Existing capturing technologies such as laser scanning allow us to automate point cloud acquisition for geometric information in the built environment. We can use the collected point cloud to extract and reconstruct BIM models for the components like columns, slabs, pipes, and walls. These generated BIM models can represent the current condition of facilities.

However, this manual modeling process is quite time-consuming and labour-intensive. Lu et al. (2019) state that it takes several weeks to complete a bridge model on average, while a complicated industrial plant often means over 6 months of work for a team of 10 or more modelers. Agapaki et al. (2018) measure the man-hours required for modelling pipes, and it takes around 5,200 labour hours to model a facility with 53,834 pipes. That means the cost and effort to generate 3D models manually exceed its benefits. Therefore, researchers are trying to automate the Scan-to-BIM process in order to reduce the human effort.

In this paper, we propose a novel method named "void-growing" that can automatically reconstruct structural elements of Manhattan-world buildings from the point cloud and test and evaluate the performance in our dataset of offices. The Manhattan-world assumption states that there is a predominance of three mutually orthogonal directions of building elements Vanegas et al. (2010). This is a valid assumption for most office buildings in Europe and the US. In our proposed approach, as the name implies, instead of detecting surfaces of elements first, we grow the void space until getting the full void volume inside a room.

RESEARCH BACKGROUND

In recent years, several research groups focused their work on Scan-to-BIM approaches. However, this topic is still solved only partially or for specific geometric elements or types of buildings.

Single room reconstruction

Some approaches are proposed to reconstruct individual rooms. Budroni & Boehm (2010) use plane sweeping to segment horizontal surfaces and vertical structures. Positions of the floor, ceilings, and walls are automatically detected. Then the 3D model can be generated from the detected ground contours. Adan & Huber (2011) use a histogram to determine surfaces of ceilings and floors and then use Hough transform to detect walls surfaces. Xiong et al. (2013) use the region growing method to form different patches and implement a stacked learning approach to classify the detected patches.

Multiple rooms reconstruction

More approaches aim to reconstruct multiple rooms. Sanchez & Zakhor (2012) propose an approach that employs principle component analysis and Random sample consensus (RANSAC) to detect large-scale architectural structures, such as ceilings and floors, as well as small-scale architectural structures like staircases. In this approach, all points are classified into doors, ceilings, walls, and remaining points. Monszpart et al. (2015) extract planar structures in a point cloud that follows regularity constraints and then optimise the plane arrangement. Authors use this approach to extract planes in many scenes such as urban scenes, exterior, and interior of buildings. Oesau et al. (2013) use horizontal slicing and volumetric-cell labeling is proposed to reconstruct watertight surface meshes. The binary labeling of the volumetric cells is formulated as energy minimization and solved by the

graph-cut method.

Xiao & Furukawa (2014) propose a method named "inverse constructive solid geometry (CSG)" that detects planar surfaces and then fits the cuboid primitives to the point cloud. Mura et al. (2014) use an approach based on the diffusion process of space partitioning. They extract patches using a simple region growing process based on normal deviation and plane offset. Wang et al. (2017) use Principal Component Analysis (PCA) to estimate the normal for each point, RANSAC to fit linear primitives, and graph-cut to identify walls. The proposed hierarchical clustering method can identify each room without knowing the number of rooms. Murali et al. (2017) use the RANSAC-based method to detect vertical and horizontal planes. Then they create a wall graph and fit cuboids to rooms. Ochmann et al. (2016) propose a method that explicitly represents buildings as interconnected volumetric wall elements. They determine an optimal room and wall layout by graph-cut based multi-label energy minimization.

Anagnostopoulos et al. (2016) use contextual to classify the points belonging to floors, walls and ceilings reasoning. Macher et al. (2017) propose a semi-automatic reconstruction approach for multi-storey buildings. The automatic part of their work is to segment the input data into sub-spaces (rooms) and planes. After segmentation, the 3D geometry of the elements is translated BIM file manually. Ochmann et al. (2019) reconstruct volumetric models of walls and slabs in multi-storey buildings. They detect planes employing RANSAC first. Then the detected planes are classified as horizontal slab surfaces and vertical wall surfaces. A 3D plane arrangement is constructed by intersecting all planes, yielding a cell complex. To find an optimal label of all cells, an integer linear programming approach in which binary variables for each cell are interpreted as room, outside, and wall cell is used.

Reconstruction using prior knowledge

Some approaches use prior knowledge explicitly to reconstruct walls and rooms. Stambler & Huber (2014) propose the concept of enclosure reasoning that premises rooms are cycles of walls enclosing free interior space. They use the region growing method to segment the point clouds and used simulated annealing to optimize rooms and walls. Tran et al. (2019) use the shape grammar approach to model indoor environments. They generate 3D parametric models by placing cuboids into point clouds, classifying them into elements and spaces. The wall candidates are obtained from pairs of adjacent peaks in the histogram of point coordinates.

Gaps in knowledge

In indoor environments, occlusion caused by furniture occurs quite often. The majority of methods are sensitive to the occlusion level. They first detect surfaces of elements in point cloud utilizing Hough transform (Adan & Huber 2011), RANSAC (Wang et al. 2017), (Ochmann et al. 2019), (Murali et al. 2017), or region growing (Xiong et al. 2013), (Mura et al. 2014), (Stambler & Huber 2014). When occlusions occur in the environment, the performance of these methods declines significantly, especially when there are complex rooms (like L-shape rooms and U-shape rooms) in the point cloud. If starting from detecting surfaces first, it is hard to distinguish a large surface of furniture like a cupboard from a relatively small wall surface.

METHODOLOGY

Overview

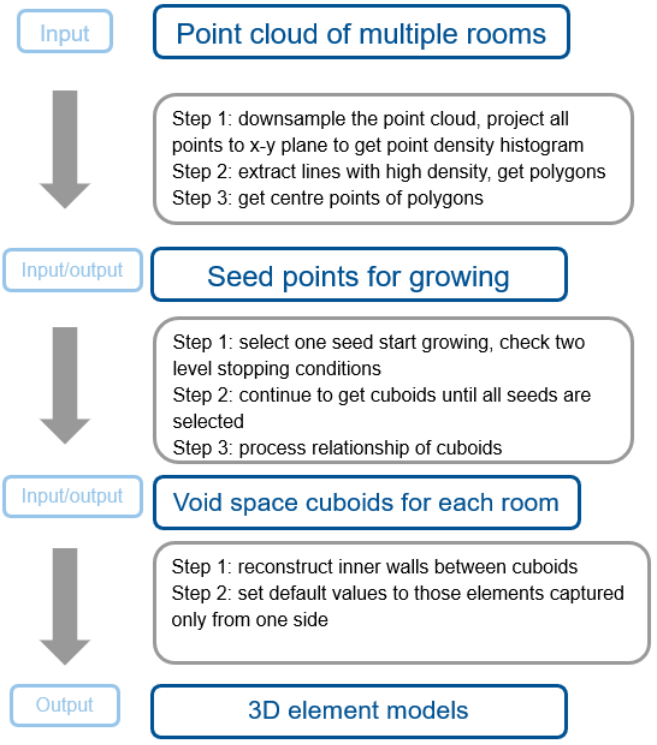

Figure 1: The workflow of the proposed approach.

In our proposed approach, we aim to detect furthest surface in all directions and get the largest void volume space inside a room. As long as the wall surface is not fully occluded by furniture, the surface can be found by the algorithm. Furthermore, although suspended ceilings are quite common in the building industry, most previous approaches do not consider different ceiling heights for different rooms in the point cloud. Instead of detecting surfaces in the point cloud first, the proposed void-growing approach aims to find the void space volume inside each room. Based on the void volumes we find, we can reconstruct the structural elements later.

By considering the void volumes at room level, it is easier to distinguish the surfaces of building elements from the surfaces of furniture. Our approach starts

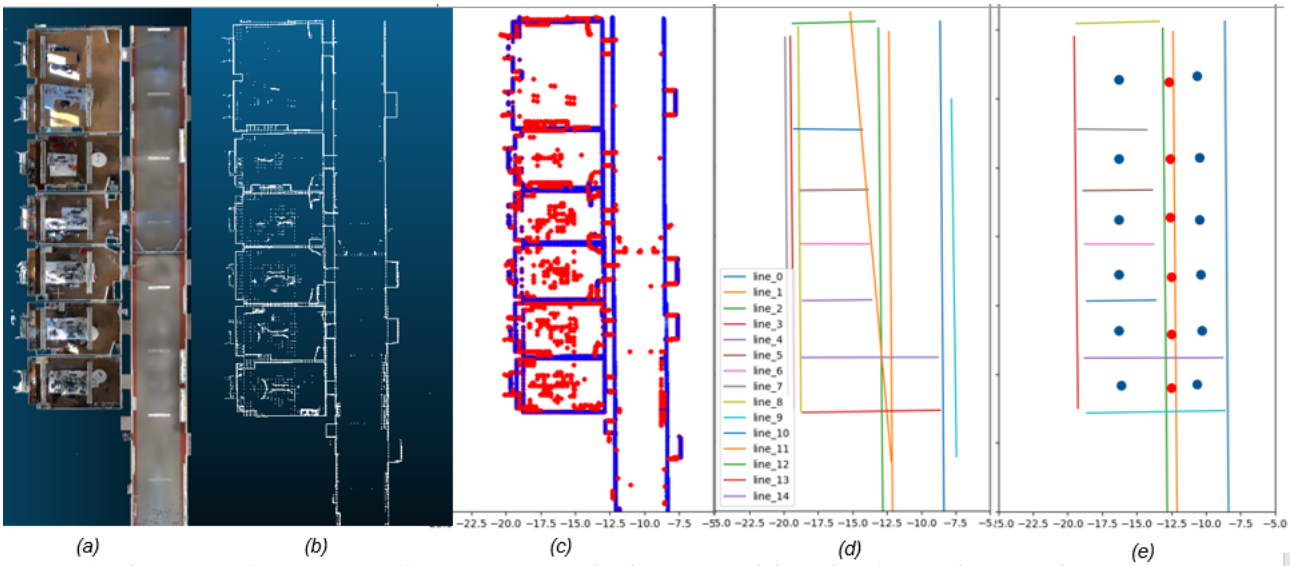

Figure 2: The process of generating seeds: (a) Input point cloud (we removed the ceiling for visualization); (b) project points to X-Y plane; (c) use RANSAC to extract lines; (d) remove lines that do not fulfil the requirements; (e) select center points as the seeds

.

with finding seed points in the point cloud of multiple rooms in one storey. Then we start the void growing to get void volumes from the seed points we found. After considering the relationship among all void cuboids, we generate one void volume for one room. At last, we create 3D models based on all volumes we found. The overall workflow of the proposed approach is illustrated in Figure 1.

Generating seeds for growing

The process of generating seeds is illustrated in Figure 2. In this step, our approach is based on several assumptions: a) the input point cloud is a Manhattan-world dataset; b) the room width and length can not be smaller than a threshold value (in our case we set the value to 30cm). Initially, given a Manhattan world point cloud of multiple rooms, what we want to achieve, is to generate seed points for growing space in further steps. We aim to find at least one seed inside a room so that we do not miss any room space in the process of creating void volumes.

In our approach, we downsample the input point cloud by voxelization to get the voxelized point cloud. Smaller voxel size in voxelization benefits precision of results as well as burdens the calculation in furthur steps. In our experiment we use 5cm voxels to compromise the precision and computational expense. Apart from that, we also use the voxels in the voxelization process. That means all voxels can be divided into two categories: void voxels and non-void voxels. Non-void voxels represent the elements in the point cloud, while void voxels represent the empty space in the point cloud. Our void growing approach is mainly working on the void voxels.

In order to find seed points for void growing, we project the voxelized point cloud after downsampling to the XY plane. In the XY plane, we get the diagram that shows the point density. In Figure 2b), we remove the points in low ponit density area to get better visualisation. The remaining points are located where point density is high and high-density areas represent a large number of points at this position in Z-direction. They usually occur in the place where vertical surfaces are present, like the vertical surfaces of walls, furniture, etc.

In the density diagram, we use RANSAC to extract lines. As our proposed approach is designed for Manhattan world, we remove the lines that are not parallel or perpendicular to X or Y coordinates. Apart from that, if a set of parallel lines are near to each other, we only select one line from them. The reason why we remove those parallel lines is that these lines with small distances cannot be two boundaries of one room (see assumption *b*).

Our next step is to calculate the intersecting points of these lines to get polygons. These polygons are potential floor plan representations, not real the real floor plan. The geometry center points of these polygons are then selected as our potential seed points in the 2D plane. We need to set a default value to our seeds to get the seeds in 3D space (we use 1.5m height in our experiment to make our seed relatively middle inside a room). As our method is to grow a void space inside a room, the voxels where our seed points are located are supposed to be void voxels. If not, we just select their nearest void voxel as our seeds.

We only need to select all these seed points roughly. The accuracy of our approach depends on further steps.

Growing from each seed

This part is the core of our approach. As we want to find the largest void space volume inside a room, the final void volume is supposed to meet two requirements: a) it should enclose the points of furniture inside the room; b) it should not expand to the outside through window and door openings.

In our proposed approach, we grow our void space in a cuboid way. That means the void space is grown from one seed (void voxel) in six directions (top, bottom, left, right, front, back). Similar to the region growing approach, our approach checks the neighbors of the seed whether they belong to the volume space at each step. In general, the growing process and the 26 neighbors of one voxel in 6 directions are illustrated in Figure 3.

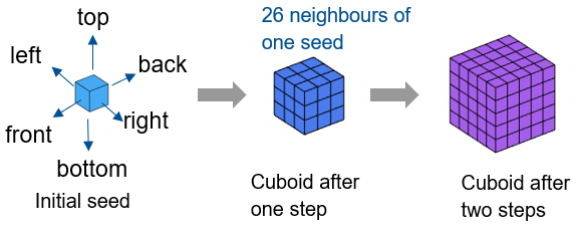

Figure 3: The general void growing process.

It is vital to determine when to stop the growing process. One reason why we grow the void space in a cuboid way is that we can store and check the frontiers in six directions and use the information on frontiers to determine whether we should stop growing in each direction. In our approach, we use two-level stopping conditions: a) the first level is used to check whether we should stop growing in one direction in first-level growth; b) if the first stopping condition is fulfilled in all six directions, the algorithm would check whether the second level stopping condition is fulfilled in all six directions. If it is not fulfilled in any direction, it continues growing in this direction until the condition is fulfilled. How the two stopping conditions are defined and checked will be introduced later in this section.

In the previous step, we have downsampled the input point cloud and store all the void and non-void voxels. Until there are no non-used seeds, the algorithm picks up a seed we have found and starts the growth of the volume space. We use S to denote the set that contains all seeds we found. This process is as follows:

1) the algorithm checks whether the selected seed is used before. If so, it would delete this seed and pick another seed.

2) the picked seed voxel is added to another set which is denoted by N_t. It represents the seed set at step t. N_0 denotes the initial set that contains only one seed from S.

3) for every seed voxel in N_t at step t, the algorithm finds its 26 neighbors. The set that contains all neighbor voxels is denoted by P. It represents the potential seeds for the next step. And we use N to represent the union of all previous seed sets, from time step 0 to time step t.

4) we remove voxels in P which are already shown in N.

5) the algorithm checks whether it fulfills the first-level stopping condition (stopping conditions will be introduced later in this section). If it should stop in any direction, we select all voxels in the frontiers of these directions. The set of these voxels on the frontier is denoted by E. If it stops in all six directions, go to step 7.

6) we remove the voxels in E from potential seed set P as it would not grow in the corresponding directions. Moreover, we also need to add the seed points from the previous step in this direction to the new seed set of the next step. Otherwise, it cannot grow in this direction without the corresponding seeds. The newly generated seed set is denoted by N_{t+1}. Then go back to step 2).

Algorithm 1 The void growing algorithm.

Input:
void voxels and non-void voxels of the point cloud;
initial seed, S;
first-level stopping condition, $F()$;
second-level stopping condition, $G()$;
functions to find all neighbors, $\alpha()$;
functions to get voxles on the frontier of any direction, $\beta()$;
functions to find neighbors in specific directions, $\gamma()$;
Initialize:
voxel list of void volume space in point cloud, $O \leftarrow \varnothing$
Algorithm:
while S is not empty **do**
  select one initial seed N_0 from S
  if $N_0 \in O$ **then**
    while $F(N_t)$ is not fulfilled in six directions **do**
      find seeds' 26 neighbours $P \leftarrow \alpha(N_t)$
      $P \leftarrow P \setminus (P \cap O)$
      if $F(P)$ is fulfilled in any direction **then**
        find voxels in that direction $E \leftarrow \beta(N_t)$
        get seeds for next round $N_{t+1} \leftarrow P \setminus E$
        $N_{t+1} \leftarrow N_{t+1} \cup \beta(Nt-1)$
        $O \leftarrow O \cup N_t$
      end if
    end while
    seeds for second-level growth $M \leftarrow N_t$
    if $G(M_t)$ is not fulfilled in any direction **then**
      find voxels in that direction $L_t \leftarrow \beta(M_t)$
      while $G(L_t)$ is not fulfilled **do**
        growing in that direction $L_{t+1} \leftarrow \gamma(M_t)$
      end while
    end if
  end if
end while

7) it checks the second-level stopping conditions for each direction. If it stops in any direction, the voxel set in the corresponding directions, denoted by M, is taken out from seed set N_t. This set is considered as our new seed set for the second-level growth. Then by using the similar method described in step 2) to step 5), it continues growing only in the specific directions. It continues growing until it fulfills the second-level stopping conditions in all six directions. The only difference is that we only use neighbors in the corresponding directions of a voxel, instead of using 26 neighbors in the first-level growth.

The pseudocode for the algorithm is shown in Algorithm 1.

The two-level stopping conditions

In our approach, we define two-level stopping conditions to determine when to stop the growth. The goal of applying these conditions is to grow the void space until wall surfaces but ignore furniture surfaces. Both stopping conditions consider the stopping in six directions separately and they both use the information on the frontier during the growing process. If there is large furniture in office rooms and a large part of a wall is occluded, we don't want the growing process stops at these furniture surfaces. The algorithm will check all small surfaces which could be a small part of a wall.

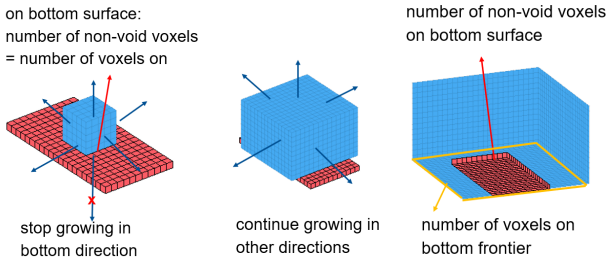

Figure 4: The first-level stopping condition: (a) it stops growing in bottom direction because it fulfils the stopping condition; (b) it continues growing in other directions except the bottom direction; (c) it keeps growing in other direction, enlarging the bottom surface, and continue growing in bottom direction when the stopping condition is not fulfilled.

We define a ratio that is used in the first-level stopping condition as follows:

$$r = P/Q, \tag{1}$$

where P denotes the number of non-void voxels in one direction, and Q is the number of void voxels in the same direction, r is the ratio used to determine whether it should stop growing in this direction. If $r \geq r_0$, it would stop growing in this direction; if $r < r_0$, it would growing in this direction. In Figure 4a), because the bottom surface of void space is fully covered by the non-void voxels at this step, the value of P is identical to the value of Q. In contrast, in Figure 4c), P is the number of red voxels and Q is the voxel number on bottom surface of the blue cuboid. The basic idea is that the algorithm compares r values of all directions at every step with a predefined threshold value (after testing different values we set it to 0.1 in our experiment). The reason to choose a small value is that we want to make sure that our algorithm would not overlook surfaces in the growing process. If r is larger than the threshold, as there is a relatively large number of non-void voxels and the algorithm would stop growing in this direction in this step. Whether it stops growing in this direction does not influence the growth of other directions. If it stops growing in one direction and continues growing in other directions, the frontier of the stopped direction is also enlarged. That means in further steps, if r is smaller than the threshold again, it can continue growing in this direction.

This process is illustrated in Figure 4. The red voxels here represent the non-void voxels, while the blue voxels are the void voxels. We show one simple example here to explain how the first-level stopping condition works. In Figure 4a), when it grows to a relatively large surface in the bottom direction (like the top surface of a desk), at this step the algorithm is not expected to decide whether it grows to a desk surface or to a floor surface. So, it pauses the growth in the bottom direction and continues growing in the other five directions as $r \geq r_0$. It will stop growing in the bottom direction until it does not meet the stopping condition in this direction (for example in Figure 4c). That means, there is only a relatively small number of non-void voxels here so that this surface cannot be a floor surface.

If we only apply the first stopping condition, our proposed algorithm could stop at walls, ceilings, and floors as well as some large furniture surfaces. But we want to make it be able to identify surfaces of furniture from structural elements. So, we propose the second-level stopping condition to check the frontiers in all directions after the first-level stopping condition.

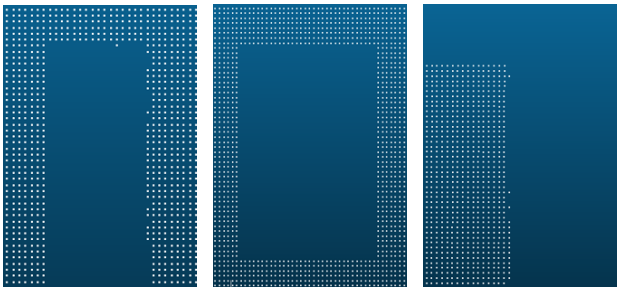

Figure 5: The pattern on growing frontier in second-level stopping condition: (a) a door opening; (b) a window opening; (c) a furniture pattern (a bookshelf)

The second-level stopping condition is based on prior knowledge. We check the patterns of void and non-void voxels on the frontier. We use the following prior knowledge to define the condition:

a) if it grows to a place in one direction where almost all voxels are non-void, it should stop the growth in this direction. We consider the surface here is the surface of floors, ceilings, or walls without openings.

b) we predefine door modules to find wall surfaces with door and window openings. Because windows are transparent, the laser scanner cannot collect any data on its surface. In our experiment, we define the door openings should have 1m-3m width and 2m-4m height. The predefined dimension of window openings we use is 0.5m-3m width and 0.5m-3m height. Apart from different dimensions of doors and windows, in our dataset window openings are usually not connected to floors. The pattern of predefined openings in vertical surfaces is shown in Figure 5. In Figure 5, each white point represents the center point of non-void voxel. If the algorithm can fit a rectangular pattern that consists of void voxels when growing in one direction, it stops and considers it as the wall surface with a door or window opening.

c) if it stops at a surface and its frontier has a rectangular pattern that consists of non-void voxels, this kind of surface is considered as the surface of furniture (as shown in Figure 5c). This pattern does not fulfill the second-level stopping condition. So, it continues growing in this direction until fulfilling it.

By applying the two-level stopping conditions, our proposed approach is expected to find void volume inside rooms with high performance, especially when there is high occlusion caused by furniture. Then we only use the points on each frontier to extract planes and consider these planes as the faces of room cuboids..

Store and project pattern on frontiers

In point clouds of the indoor environment, if part of a wall surface is occluded by furniture, only the furniture surface and the other part of the wall surface that is not covered by furniture can be scanned. That means there would be no points at the wall surface behind the furniture. If we find large amount of non-void voxels caused by furniture on the frontier, the algorithm would continue growing as it does not find any pattern of door or window opening. But in further steps, if it grows to a wall surface, it cannot stop because of the "hole" on this surface.

In order to solve this problem, we combine the patterns we found in previous steps with that of current step together. In the growing process, the furniture pattern we found would be stored. And in further steps, we project the non-void voxels of the furniture surface to the frontier of the current step. After filling the "hole", the algorithm then checks the second stopping condition on the surface. This process is illustrated in Figure 6. It grows to the furniture surface first and then stores the found pattern of non-void voxels. After that in further growth, we project the found pattern to the new frontier to get a new pattern on the surface. The algorithm continues growing until the stopping condition is fulfilled.

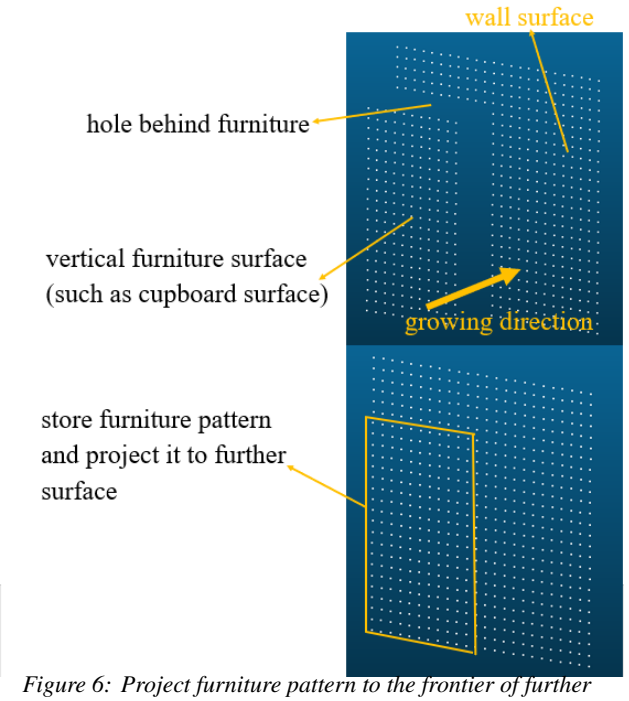

Figure 6: Project furniture pattern to the frontier of further steps.

Merge the connected cuboids

From the previous steps, we have gotten cuboids for void volume inside rooms. As we have at least one seed inside one room, one room could have multiple cuboids from different seeds. This happens usually in some complex rooms, like U-shape rooms, L-shape rooms, rooms with different ceiling heights, etc. There are two circumstances when two cuboids should be merged in one: a) one surface of a cuboid touches a surface of another cuboid: b) two cuboids have overlapping space.

Extract walls, floors, and ceilings

In this step, we extract elements we want to reconstruct from the cuboids generated from the last step.

We use different strategies for different kinds of structural elements. For ceilings, floors, and outer walls, we usually only collect data inside the room. As a result, we do not have data from the other side. For these elements, we use the surface where void volumes stop growing as the inner surfaces of these elements. We can leave it as a plane or set a default value.

In contrast, both sides of the inner walls can be scanned when collecting data inside a building. That means we grow void volume space inside adjacent rooms. The space between to void cuboid is considered as the inner wall (as shown in Figure 7). The thickness of the inner wall is the distance between these two surfaces.

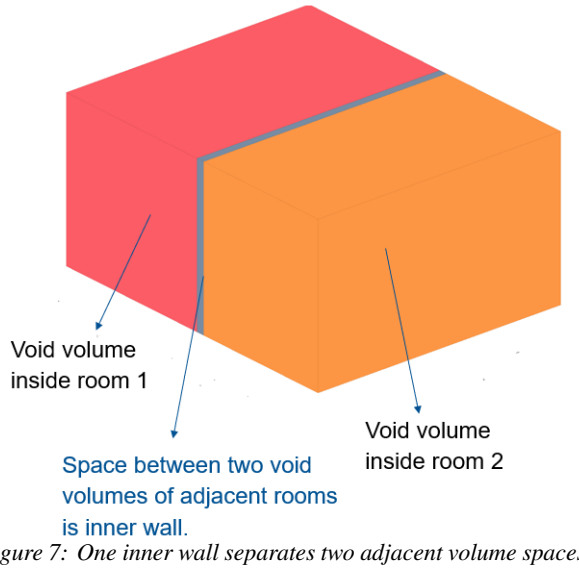

Figure 7: One inner wall separates two adjacent volume spaces.

Experiment and result

The code of void-growing approach is written in C++ by using Point Cloud Library (PCL) 1.9.1 (Rusu & Cousins 2011) and the Computational Geometry Algorithms Library (CGAL) 5.1 (The CGAL Project 2020). The input dataset was collected in the office area of the Chair of Computational Modeling and Simulation at the Technical University of Munich.

The input point cloud and the cuboids we get inside each room are shown in Figure 8. Each color represents a void volume inside a room. As we can see, not only typical cuboid rooms but also complex rooms can be detected. The hallway can be seen as an L-Shape room. Furthermore, if we focus on the ceiling of the input cloud, the ceiling heights of some rooms and the hallway are not identical because of suspended ceilings which are quite common in the building industry nowadays. The information of different ceiling heights in the input point cloud can be identified clearly in our grown void volumes.

In Figure 9, we remove the points representing the ceiling to depict rooms of the input point cloud more clearly. It is evident to see that a gap separates two adjacent room volumes. Moreover, the gaps between two adjacent rooms are supposed to be the wall that separates these rooms.

As shown in Figure 8 and Figure 9, the void growing algorithm performs qualitatively will in our dataset, a typical indoor environment of Manhattan-world offices with strong occlusion. We state that our algorithm could apply in other Manhattan-world point clouds without modifications or with small modifications. In most cases, if the buildings have similar door and window openings, we can apply our approach directly to new datasets. If the door and window openings have different shapes (like circles), it would give us different patterns on frontiers during the growing process. The current second-level stopping conditions would not work. To fit these patterns and make the algorithm stop at the opening surface, we could add other predefined opening shapes and consider those patterns as our new stopping conditions.

Based on the volume spaces we have found in previous steps, we can extract walls, floors, and ceilings. In our experiment, if only one surface of elements is scanned, we set the default thickness of the element to 30cm. The reconstructed 3D model created by our proposed approach and the BIM model created manually are shown in Figure 10. We evaluate the two models quantitatively by comparing the areas of offices in Table 1 and the thicknesses of walls in Table 2.

Table 1: Area comparison between the void-growing model and BIM model: (m^2)

office No.	void-growing	BIM	deviation (abs.)	deviation (rel.%)
1	46.62	49.22	2.60	5.28
2	26.15	26.82	0.67	2.50
3	23.31	24.11	0.80	3.32

Table 2: Wall thickness comparison between the void-growing model and BIM model: (m)

wall No.	void-growing	BIM	deviation (abs.)	deviation (rel.%)
1	0.200	0.173	0.027	15.6
2	0.200	0.164	0.036	22.0
3	0.150	0.144	0.006	4.2

As shown in Table 2, the thickness of the detected wall is almost identical. The reason is that we use a very simple downsampling strategy: the voxel center points are picked as the new points in point cloud. The voxel size limits the performance of our approach. When detecting objects with large dimensions, the impact is insignificant. However, it becomes very important to find the thickness of elements. The results can be improved by reducing the voxel size, but the computational effort would be increased as well. We need to make a compromise between precision and computational burden.

CONCLUSIONS

This paper presents a void growing approach for the Scan-to-BIM process for buildings with a Manhattan world structure. Instead of starting with detecting surfaces in the point cloud, the void growing approach detects void space inside rooms first. It can handle both, simple cuboid rooms, and rooms with complex structures, like L-shape rooms. Room volumes with

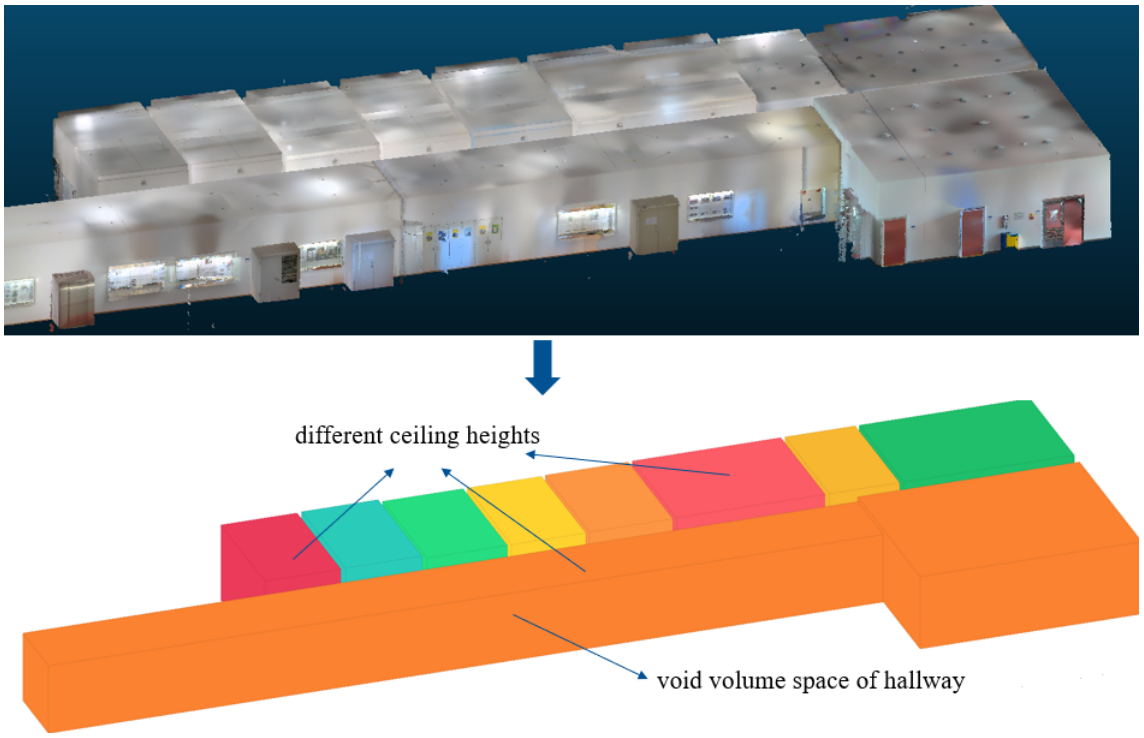

Figure 8: Input point cloud and void volumes inside rooms with different ceiling heights.

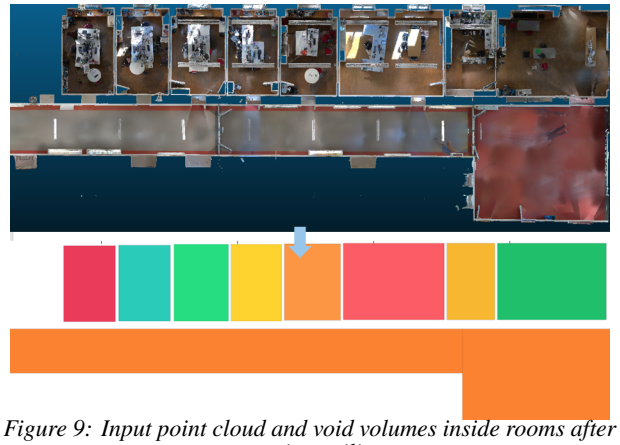

Figure 9: Input point cloud and void volumes inside rooms after removing ceilings

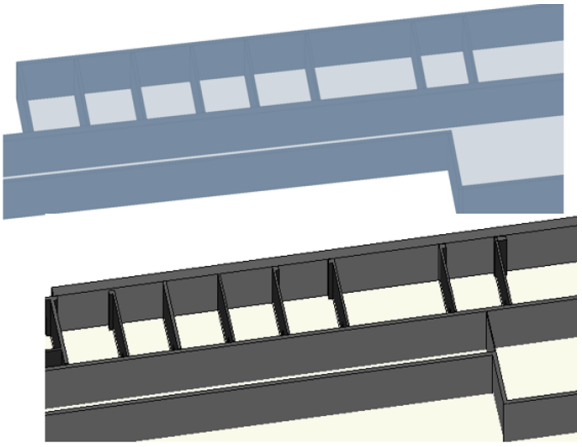

Figure 10: The model generated by void-growing approach and the BIM model: a) reconstructed model b) BIM model

different suspended ceiling heights can also be represented. As it aims to detect the furthest surface inside a room in all directions, it can find surfaces of structural elements even when the occlusion level is relatively high. However, if a surface of a wall is fully occluded by large furniture, the void growing would stop at the surface of the large furniture.

In the future, we plan to use deep learning techniques to train neural networks to segment point clouds by various categories. By using semantic, as well as geometric information, we can define new stopping conditions for the void-growing approach. Furthermore, apart from ceilings, floors, and walls, we also want to extend our current approach to detect more elements like columns, beams, staircases, etc.

ACKNOWLEDGMENTS

The work in this paper is funded by the Institute for Advanced Study (IAS) at the Technical University of Munich under Hans Fischer Senior Fellowship. The dataset we use in this paper is collected on the main campus of the Technical University of Munich with the help from NavVis (https://www.navvis.com/).

REFERENCES

References

Adan, A. & Huber, D. (2011), 3d reconstruction of interior wall surfaces under occlusion and clutter, in '2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission', IEEE, pp. 275–281.

Agapaki, E., Miatt, G. & Brilakis, I. (2018), 'Prioritizing object types for modelling existing industrial facilities', Automation in Construction **96**, 211–223.

Anagnostopoulos, I., Pătrăucean, V., Brilakis, I. & Vela, P. (2016), Detection of walls, floors, and ceilings in point cloud data, in 'Construction Research Congress 2016', pp. 2302–2311.

Budroni, A. & Boehm, J. (2010), 'Automated 3d reconstruction of interiors from point clouds', International Journal of Architectural Computing **8**(1), 55–73.

Lu, R., Brilakis, I. & Middleton, C. R. (2019), 'Detection of structural components in point clouds of existing rc bridges', Computer-Aided Civil and Infrastructure Engineering **34**(3), 191–212.

Macher, H., Landes, T. & Grussenmeyer, P. (2017), 'From point clouds to building information models: 3d semi-automatic reconstruction of indoors of existing buildings', Applied Sciences **7**(10), 1030.

Monszpart, A., Mellado, N., Brostow, G. J. & Mitra, N. J. (2015), 'Rapter: rebuilding man-made scenes with regular arrangements of planes.', ACM Trans. Graph. **34**(4), 103–1.

Mura, C., Mattausch, O., Villanueva, A. J., Gobbetti, E. & Pajarola, R. (2014), 'Automatic room detection and reconstruction in cluttered indoor environments with complex room layouts', Computers & Graphics **44**, 20–32.

Murali, S., Speciale, P., Oswald, M. R. & Pollefeys, M. (2017), Indoor scan2bim: Building information models of house interiors, in '2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, pp. 6126–6133.

Ochmann, S., Vock, R. & Klein, R. (2019), 'Automatic reconstruction of fully volumetric 3d building models from oriented point clouds', ISPRS journal of photogrammetry and remote sensing **151**, 251–262.

Ochmann, S., Vock, R., Wessel, R. & Klein, R. (2016), 'Automatic reconstruction of parametric building models from indoor point clouds', Computers & Graphics **54**, 94–103.

Oesau, S., Lafarge, F. & Alliez, P. (2013), Indoor scene reconstruction using primitive-driven space partitioning and graph-cut.

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

Sanchez, V. & Zakhor, A. (2012), Planar 3d modeling of building interiors from point cloud data, in '2012 19th IEEE International Conference on Image Processing', IEEE, pp. 1777–1780.

Stambler, A. & Huber, D. (2014), Building modeling through enclosure reasoning, in '2014 2nd International Conference on 3D Vision', Vol. 2, IEEE, pp. 118–125.

The CGAL Project (2020), CGAL User and Reference Manual, 5.1.1 edn, CGAL Editorial Board. URL: https://doc.cgal.org/5.1.1/Manual/packages.html

Tran, H., Khoshelham, K., Kealy, A. & Díaz-Vilariño, L. (2019), 'Shape grammar approach to 3d modeling of indoor environments using point clouds', Journal of Computing in Civil Engineering **33**(1), 04018055.

Vanegas, C. A., Aliaga, D. G. & Benes, B. (2010), Building reconstruction using manhattan-world grammars, in '2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition', IEEE, pp. 358–365.

Wang, R., Xie, L. & Chen, D. (2017), 'Modeling indoor spaces using decomposition and reconstruction of structural elements', Photogrammetric Engineering & Remote Sensing **83**(12), 827–841.

Xiao, J. & Furukawa, Y. (2014), 'Reconstructing the world's museums', International journal of computer vision **110**(3), 243–258.

Xiong, X., Adan, A., Akinci, B. & Huber, D. (2013), 'Automatic creation of semantically rich 3d building models from laser scanner data', Automation in construction **31**, 325–337.