Generation of 3D Building Models from Archived City Area Maps by Fusing Information from Multiple Sources

Chaoqun Dong[1,*] · **Roman Martel**[1,*] · **Kan Chen**[1] · **Henry Johan**[2] · **Marius Erdt**[1,2]

Abstract This paper proposes a pipeline for the automatic generation of 3D building models for city areas based on information from archived city area maps. These maps are typically created and archived by city authorities for several decades. As such, they exhibit several challenging properties like a mixture of handwritings and typewriter font styles, varying layouts and notation standards, low contrast and physical damages. To tackle these challenges, we propose to extract and fuse information from multiple sources. In the proposed pipeline, we firstly locate and extract text content within the city area maps to obtain the essential information to identify described buildings. Secondly, based on this information, we retrieve the building height information and addresses from a public housing database. Then, we extract the building shape and size information on the basis of the obtained addresses through an online map API. Lastly, utilizing all the acquired building information, we generate 3D models of the buildings and their neighborhoods in the CityGML LOD1 format. The whole pipeline and its individual components are tested on a dataset of city area maps provided by city authorities.

Keywords Computer Vision · 3D Building Models · Archived City Area Maps · Text Recognition · Deep Learning

1 Introduction

In the past, city planning using hand drawings and typewriters was the common practice. City authorities can, therefore, provide plenty of archived non-digital maps and documents. These documents contain detailed information about many older city areas and their historical development. Making this information available to modern 3D modeling and simulation tools used by city authorities nowadays is a challenging task. The documents are often archived over decades, which can lead to damages and low contrast. Scanning of the documents may introduce additional artifacts. Furthermore, over the years, the standard formats, typewriter fonts and handwritings in the documents change.

In this paper, we address these challenges by proposing a pipeline which leverages the resource of archived city area documents. It extracts building information from the city area documents and fuses it with other data sources to generate a 3D model of the city area. The main features of the proposed pipeline are as follows:

- We apply a deep learning based method to locate and extract information from city area documents to identify target buildings.
- Based on this information, we find the target buildings in other databases from which the building shape, size and height information is extracted.
- Based on the obtained building heights and shapes, after coordinate system conversion, we can generate a list of 3D models for all target buildings, which have correct aspect ratios and relative locations.

There are many applications and use cases which benefit from 3D models of city areas like urban planning, visualization, traffic and wind simulations, estimation of population, energy demand or solar irradiation and many more. A structured overview of use cases can be found in [6]. For many of these use cases a low accuracy estimate of building models is already sufficient. We, therefore, use the CityGML representation [16] with level of detail one (LOD1) to store the output models. LOD1 is used for simple building models with no details as shown in Figure 17.

This research was funded by the National Research Foundation, Prime Ministers Office, Singapore under the Virtual Singapore (VSG) Programme First R&D grant number NRF2015VSG-AA3DCM001-004.

* These authors contributed equally to this work.
1 Fraunhofer Singapore, Singapore
2 Nanyang Technological University, Fraunhofer IDM@NTU, Singapore

To get access to authentic data, we collaborate with city authorities in Singapore. They provided us with a dataset of archived city area documents in the form of maps for several areas in Singapore. A representative example for such a map can be seen in Figure 2a. They mostly contain a detailed map area with streets and building outlines. A smaller overview map in a box at one of the corners shows the area in a larger context. Several text boxes, distributed around the map, provide various additional information for a couple of the buildings identified by a block number. The city area itself is identified by a city area name located in one of the text boxes. There is a large variation of general layout, handwritings and typewriters fonts. The drawings displaying building shapes are in many cases degraded over time and often show ambiguities among lines for representing building shapes and lines for measurement or description. As a result, the city area maps contain unreliable and limited building shape information. These challenges make the dataset a good candidate to demonstrate the general applicability of the pipeline in a real-world scenario.

In summary, the main contribution of this paper is to make the vast amount of information stored in documents archived by city authorities accessible for modern 3D modeling and simulation tools. To achieve this, we propose a pipeline which is able to deal with the challenges of such archived documents by fusing them with additional data sources. We further test the pipeline on a real dataset of city area maps and report the results for the whole pipeline and its components.

This paper is structured as follows. Section 2 summarizes related work. Sections 3, 4, 5, 6, 7, 8 and 9 describe the proposed pipeline. Our results are summarized in Section 10 and subsequently discussed in Section 11. Finally, Section 12 concludes the paper.

2 Related Work

2.1 Single Building 3D Model Reconstruction

Several approaches have been developed for the reconstruction of single building models. Approaches based on scanned 2D floor plans are summarized in [13]. [11] gives an overview focusing on detailed building models based on photo and laser scanning data. In [1], a pipeline for the analysis of floor plans using classical image analysis is proposed. The inputs are high quality floor plans of single buildings which contain a standardized set of text labels, drawings and symbols indicating doors, walls, windows and other entities. Using these standards, the task is separated into smaller steps such as separation of text and graphics, wall detection and automatic detection of rooms. A similar approach was not applicable to our dataset of city area maps because of the lack of standards and the lower quality. For example, the text recognition in [1] worked well because a single digital font was used across the whole dataset. Our dataset contains a mixture of several typewriter and handwritten fonts which are partly damaged due to large storage and scanning artifacts. Several papers focus only on a single step of the above mentioned pipeline using the same or similar datasets. Segmenting text elements from graphics in floor plans is described in [3] and [29]. Automatic wall detection is the topic in [8] and [2]. [22] deals with the automatic detection of rooms in floor plans. A similar approach as in [1] to automatically generate a 3D model from scanned 2D floor plans is described in [14]. In [12], 3D building models are generated based on 2D floor plans in computer-aided design (CAD) format.

2.2 City Area 3D Model Reconstruction

The final target of this paper is the generation of 3D building models for whole city areas. A variety of approaches have been developed to address this task. A dataset of 2D digital maps manually annotated with polygons is used in [28] to automatically generate 3D building models. The resulting models have a higher level of detail than in our approach, however, the dependance on manual polygon annotations limits its applicability. Stereo pairs of satellite images are used in [9] to reconstruct 3D building models on a city scale. In [5], a dataset of already prepared building footprints is utilized to deduce the shape of buildings. To obtain an accurate and robust estimate of the building heights, additional information about the buildings and statistical data of their local neighborhood is used. Airborne and terrestrial LiDAR data is combined with aerial photographs in [30] to automatically generate 3D building models for a whole city area. The acquisition of the necessary data for this approach on a city scale is, however, costly in terms of time, effort and money. In [23], the outlines of a pipeline based on the same city area maps as ours is proposed. Compared to our current approach, there are several important differences. The method for initial map segmentation based on calculating image statistics was replaced with a more robust one using a rank filter and Hough transform (see Section 4). For the text detection, the previous method based on extremal regions was replaced with a CNN approach (Section 5). As [23] describes only an outline of a pipeline, no solution for the text recognition and information extraction is provided. These steps are added in the current pipeline (Sections 6 and 7). In the shape extraction part, image tilling and text removal were added to handle the cases in which the target building is large or contains text overlappings at the building boundaries (Section 8). In addition, a new method to generate 3D building models in a large area with correct aspect ratios and relative locations is proposed (Section 9). Furthermore, in [23] no experimental results are presented and discussed because the paper only proposed the outline of a pipeline without a complete implementation.

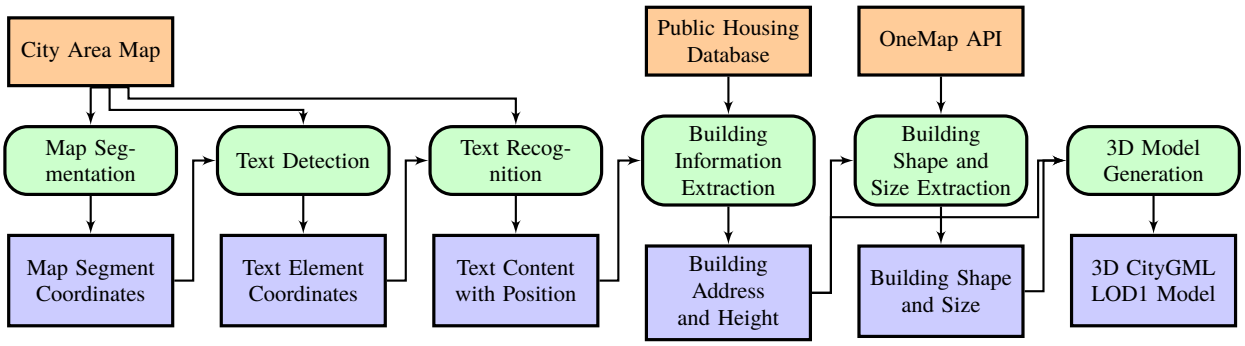


Fig. 1 Overview of the proposed pipeline. All input sources are shown in orange on the top. Every step of the pipeline is displayed in green. The output data appear on the bottom in blue

In [15] and [25], further approaches are described which make use of data from the OpenStreetMap project [17]. The data from the OpenStreetMap project can be very well combined with our approach, making it applicable to many regions for which this data is present. In our concrete test example of city area maps in Singapore, we used data from the new OneMap API [24]. That is because the new OneMap API is the official reference data source for building locations and shapes in Singapore while OpenStreetMap relies on crowdsourced data.
In [4], improvements to the CityGML standard itself [16] are proposed which is used to store the resulting 3D models of our pipeline.

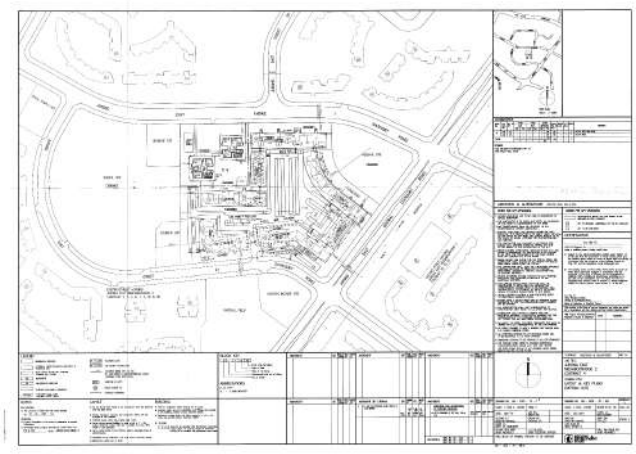
3 Overview of the Proposed Pipeline

Given a document describing a city area, we propose to find a set of buildings in the document and combine several data sources to identify them and generate their 3D building models in the CityGML LOD1 representation. This idea could work with various kinds of documents. The main requirement is that they contain the location information in text for all buildings which should be generated. This location information could be the building address, the geographical location in longitude and latitude or a building identifier for which the address can be determined in a separate database. Assuming the building location has been found, several other data sources can be used to obtain the necessary information to generate a 3D model for the building. For instance, the OpenStreetMap project [17] maintains an API which provides the building shape and size information for a given building address. Furthermore, some governments maintain housing databases or online map APIs for which building shape and size can be extracted.

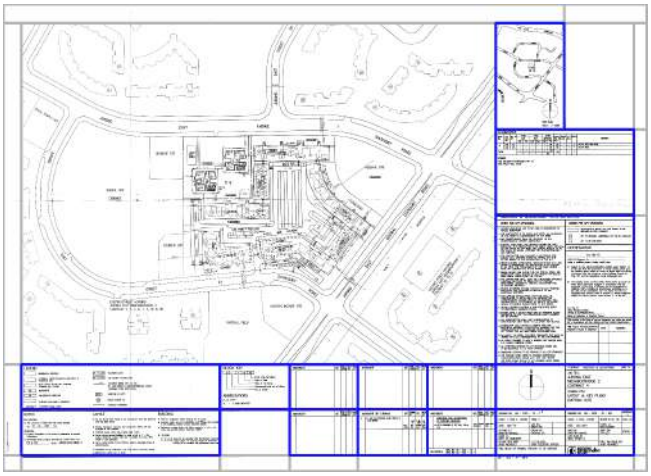
As a case study to demonstrate the feasibility of this idea, we propose a concrete pipeline which takes real city area maps archived by city authorities in Singapore as input documents. The overview of this pipeline is illustrated in Figure 1. The inputs are scanned versions of city area maps (see Figure 2a). A city area map describes the surrounding area for a few target buildings. These buildings are listed in a table somewhere in the map using building block numbers to identify them (see e.g. Figure 3 left). Furthermore, the name of the city area is mentioned in another location in the map (see e.g. Figure 3 right). Combined, the city area name and block number serves as an identifier for a building. The city authorities in Singapore maintain a public housing database[1] which contains the address and further information like number of storeys for each building identifier. Using the addresses from the database, we could use the OpenStreetMap project [17] to get the building shape and size information. On one side this approach would work in general for most regions worldwide, on the other side we would rely on crowdsourced data which is not guaranteed to be accurate. In case of Singapore, the city authorities also provide an online map API (OneMap API [24]) which can be used for the same purpose. This API serves as the official reference for building shapes and locations in Singapore. For this case study, we therefore use the OneMap API to locate the buildings and get images of their footprints. With shape and size information extracted from the footprints and height information computed from the housing database, 3D building models can be generated in the CityGML LOD1 format. Our approach in total is divided into six steps.

1. **Map Segmentation**: segment the city area map into its main parts and filter out the irrelevant ones (Section 4).
2. **Text Detection**: detect all text elements in the relevant parts (Section 5).
3. **Text Recognition**: recognize all detected text elements and classify them to be relevant or not (Section 6).
4. **Building Address and Height Extraction**: find the building block number and city area name of the target buildings amongst the recognized text and identify the build-

[1] https://data.gov.sg/dataset/hdb-property-information



(a) Input city area map before any processing



(b) City area map after segmentation. Irrelevant segments are marked with gray boxes while relevant ones are blue

Fig. 2 Example for a city area map segmentation result. The segments mainly align with the separating lines of the original map boxes/segments

ing in the public housing database to obtain the address and height (Section 7).

5. **Building Shape and Size Extraction**: find the footprints images of the identified buildings using the OneMap API and extract their shape and size (Section 8).
6. **3D Model Generation**: generate 3D building models by combining the previously gathered building height and shape information and converting to the appropriate coordinate systems (Section 9).

Besides the city area maps, we are also provided with several additional documents showing detailed building floor plans and building height profiles. This additional data is not processed by the pipeline directly, however, it is used to get training data for the text detection in Section 5 and to derive the building height formula in Section 7.3.

4 City Area Map Segmentation

We segment the input city area map into smaller parts alongside the boundaries of the map areas and text boxes. In this step, we make an assumption that these boundaries are indicated by separating lines, which are fairly horizontal or vertical (see e.g., Figure 2a). The provided maps have a high resolution, which is usually around 13200×9600 pixels. This level of detail is not necessary for a rough segmentation of the main areas and would slow down most algorithms. Furthermore, we want to use the same parameter in the x and y directions for the following steps. We therefore first reduce the input city area image to a square resolution of 500×500 pixels.

Afterwards, adaptive thresholding is used to transform the grayscale image to a binary one. A rank filter [27] with long horizontal and vertical lines as kernel is applied to the binary image. This results in an image containing mainly the separating horizonal and vertical lines of the map segments plus some unwanted remnants from other structures. To obtain a parameterized form of the remaining lines (e.g. slope and y-intercept), a classical Hough transform [10] is applied to this image. Because the separating lines of interest are either horizontal or vertical, we now filter out all lines which deviate more than 5 degree from being horizontal or vertical. The remaining lines are mapped to the closest horizontal or vertical line. After that, all lines are described only by their position on the x axis (for vertical lines) or y axis (for horizontal lines). We go through the image from top left to bottom right combining the lines to rectangular boxes. A set of four lines will be combined to a rectangular box if each line covers more than 50% of a corresponding side of the box. The image boundaries are thereby considered as full lines. Therefore, if no lines are detected by Hough transform, the algorithm will simply return a single segment containing the whole image.

The resulting segments with extreme aspect ratios, nearly empty or completely filled content and covering a large part of the image (usually map area) are filtered out. The result of this method applied on a typical city area map is shown in Figure 2b. The segments which are filtered out are shown in gray while the remaining segments after filtering are highlighted in blue. The remaining segments are the output of this step of the pipeline.

5 Text Detection

The aim of this step is to find all text elements in the original city area maps. There are publicly available OCR engines like Tesseract [26] or OCRopus [7] to solve this task. These engines provide many convenience functions such as grouping detected words to lines or blocks and choosing languages for the recognition. However, in our case, they proved to be not

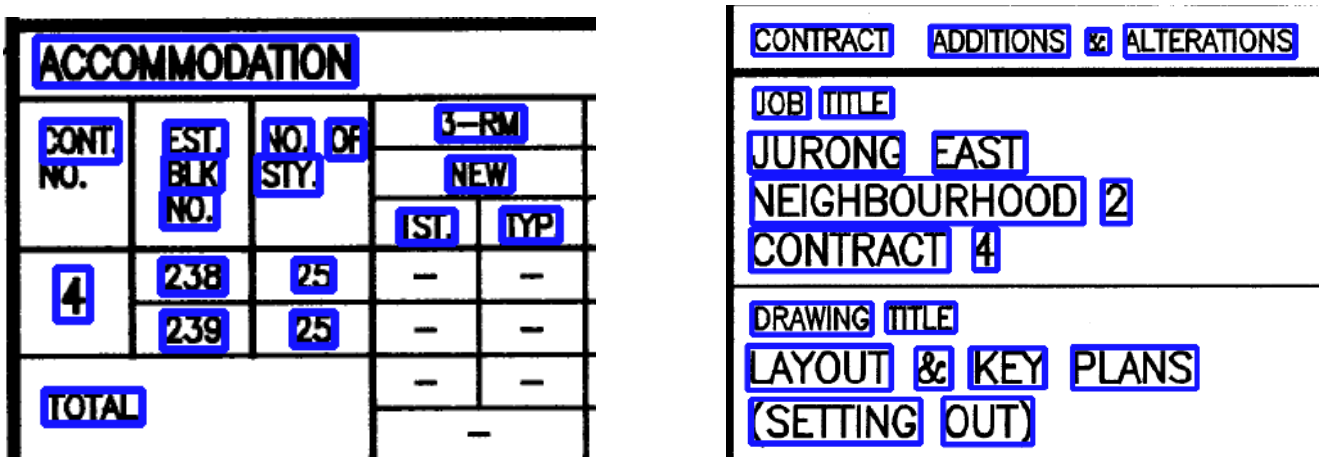


Fig. 3 Example results for the text detection using the EAST text detector [31] with a model pre-trained on ICDAR 2013 and ICDAR 2015 and fine-tuned on our own dataset of city area map segments. The detected text boxes are marked in blue

applicable because of the poor quality of the maps and the challenging mixture of hand-written and typewriter fonts. Our choice fell therefore on the EAST text detector [31] which is reported to be fast and robust to noise and variation of fonts.

Our solution builds on an existing implementation with a trained model[2]. This model is pre-trained on the training sets of the ICDAR 2013 and ICDAR 2015 competitions [20], [19]. These datasets are very different from our city area maps. They contain color images of random scenes taken by a Google Glass in a city area. Therefore, we fine tune the model on a training dataset collected from other city area documents. These documents are different from the original city area maps but were provided by the same city authorities and therefore exhibit a similar layout and style of fonts. The training dataset consists of 104 segments, for which we manually labeled the bounding box of each text element. The 104 segments were determined by applying the method from Section 4 to the other city area documents. Fine tuning is performed for 50K steps using a batch size of 12 images per step and random data augmentation operations such as cropping, resizing and slight rotations. Loss functions, optimization protocol and regularization are the same as in [31]. Examples for the detection results using the final fine-tuned model can be seen in Figure 3.

6 Text Recognition

In this step, we have the detected text boxes from the previous step as input and have to recognize their text content. However, we are only interested in a small part of the total text content, namely the city area name and the block numbers of the target buildings. It is therefore not necessary to recognize all text elements but only a predefined subset. In [18], the text recognition task is treated as a classification problem, where each class represents a possible word and the total amount of words is fixed and given by a large dictionary. In the case of [18], this dictionary can contain up to 90K words. We, however, need to classify only a much smaller set of about 3K words, which makes the training faster and requires less training data. Besides that, the approach in [18] expects the same input data as ours, text boxes cropped around single words. Furthermore, [18] proposes to train the model on synthetic data and reports to have robust results using this approach. For these reasons we decided to follow the approach in [18] for the text recognition step.

The model in [18] is based on a CNN for feature extraction and a classifier using two fully-connected layers. We provided our own implementation of the model based on the original paper. Due to our lack of labeled training data for text recognition, we also adopt the idea of training on synthetic data as presented in [18]. The details of the creation for the synthetic dataset are described in paragraph *Synthetic Dataset*.

For the word class dictionary, we decided to add classes from four categories:

1. **City Area**: we compiled a list of all possible words occurring in all areas of the city.
2. **City Area Keyword**: keywords indicating the position of city area name in the city area maps.
3. **Building Block Number**: a list of all numbers which are allowed to be used for building blocks.
4. **Building Block Number Keyword**: keywords indicating the positions of the list of building block numbers in the city area maps.

All words which do not belong to any of these categories will be considered as irrelevant. Example classes for the categories can be seen in Figure 5. Combining all classes

[2] https://github.com/argman/EAST

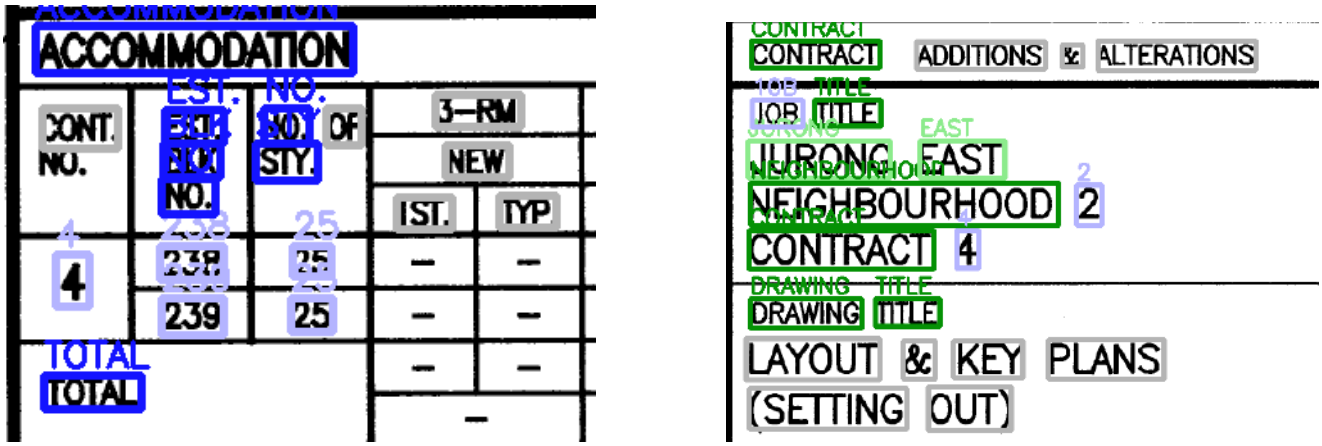


Fig. 4 Example results for the text recognition using the approach described in [18]. The model is trained from scratch for 10 epochs on a specifically created synthetic data set.
Color coding of the text boxes: light green: city area names, dark green: city area name keywords, light blue: building block numbers, dark blue: building block number keywords, gray: irrelevant

and one class for irrelevant words (residual class), we end up with a dictionary containing 3182 classes. The recognition model is trained from scratch on our own synthetic training dataset. We train for 10 epochs and follow the procedure in [18] regarding loss function, optimization, regularization and choice of batch size. Examples for the results when applying the trained recognition model on the detected text boxes (Figure 3) from the previous step can be seen in Figure 4.

Synthetic Dataset We cannot use the same dataset as in [18] because our objective is different. Instead of recognizing words in colored random scene images, we try to recognize words in grayscale city area maps. The challenge is hereby the robust recognition of words typed with different typewriter fonts or written with different hand writings.

We, therefore, gathered 10 different publicly available hand writing font sets and 17 different typewriter fonts. For each word, we randomly sample the font, font size, padding or cropping amount, rotation in the range -4 to 4 degree and whether salt-and-pepper noise should be added to the word image. In this fashion, we generate 100 word image samples for each of the 3181 target word classes (residual class excluded). The residual class represents a potentially large number of words and appears more often in the city area maps than the other classes. Therefore, we set the number of residual class samples to be 20% of the total amount of the other word class samples. The sampling process for the residual class is identical to the other classes except that we also have to randomly sample one of the potential words it can represent. These potential words are chosen from a pool of words collected from the city area maps, which do not belong to any of the four categories mentioned in Section 6. To account for possible empty detections in the previous step, we also set 1% of the residual class images to be empty with random noise only added. This leads in total to a training and test dataset containing 381840 samples. 10 % of the samples are chosen to be the test set, and are therefore used only for evaluating the final text recognition performance after training. Figure 5 displays a couple of samples from the synthetic data set.

(a) Class: "Woodlands", Category: City Area

(b) Class: "Geylang", Category: City Area

(c) Class: "Contract", Category: City Area Keyword

(d) Class: "688E", Category: Building Block Number

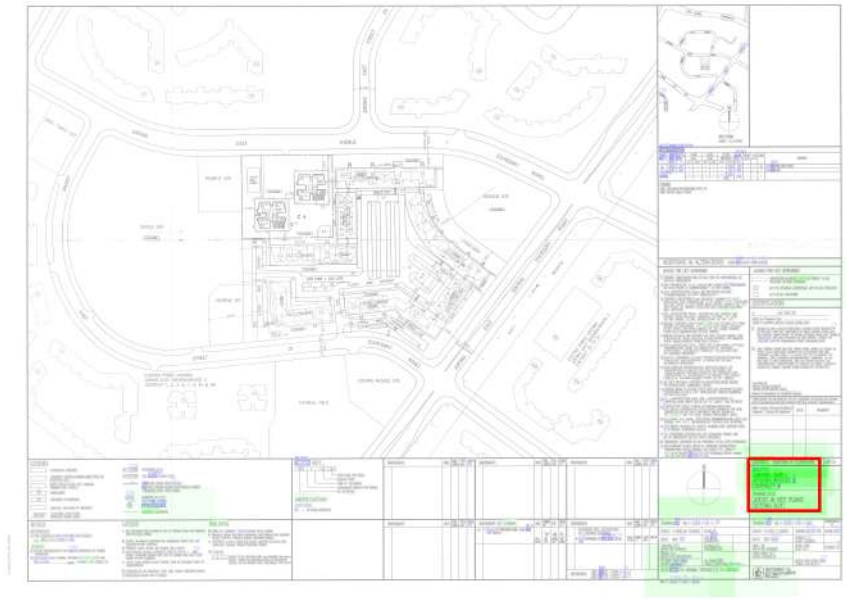
(e) Class: "210C", Category: Building Block Number

(f) Class: "BLK.", Category: Building Block Number Keyword

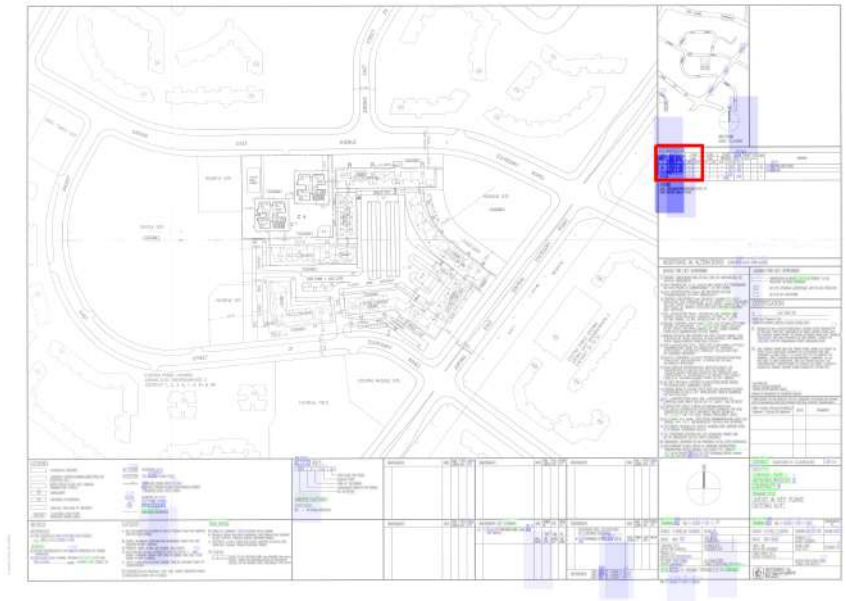
(g) Class: "-", Category: irrelevant

(h) Class: "-", Category: irrelevant

Fig. 5 Example images of the synthetic dataset with class name and category



(a) Example for the voting mechanism to find the position of the city area name



(b) Example for the voting mechanism to find the position of the building block numbers

Fig. 6 Each keyword votes for a box to indicate the location of the target information. The boxes for the city area name are displayed in green, the ones for the building block numbers in blue. The color becomes more opaque with increasing votes in a region. The red rectangles show the regions containing the target information. These regions are displayed in Figures 3 and 4. One can see that they correspond to the regions with most votes as indicated by the most opaque colors

7 Building Address and Height Extraction

7.1 Keyword Voting Mechanism

The input in this step is the list of all words with their positions in the city area map, which have not been classified as irrelevant. All these words have a label attached identifying the class and category. The target words have the category "City Area" or "Building Block Number". Typically many instances of these categories are found across the whole city area map. However, the city area maps follow some standards. The city area name and block number list are accompanied by a couple of keywords. For example, in Figure 3 on the right, the city area "JURONG EAST" is surrounded

by the keywords "JOB", "TITLE", "NEIGHBOURHOOD" and "CONTRACT". These keywords are the same across the whole dataset. We therefore identified and included them in the training data set in the previous Section 6. For each keyword, we also know whether it should appear above, below, left or right of the target information. To find the position of the target words, we therefore propose a voting mechanism in which each keyword votes for a box containing the target word. For example, in Figure 3 on the right, the keyword "NEIGHBOURHOOD" will vote for a box above its location to contain the city area "JURONG EAST". The sizes of the boxes are chosen to be generous to allow for some variation of the concrete relative position of keyword to target word in each map. The voting for the "City Area" and "Building Block Number" categories are done separately. Figure 6a demonstrates an example result for the city area voting. The corresponding result for the building block number is shown in Figure 6b.

7.2 Building Address Extraction

After voting, the proposed regions are processed starting with the region with the most votes. An example vote region for the city area is shown in Figure 4 on the right. We start by listing all words with label category "City Area" inside the vote region. In the example in Figure 4 on the right, these are "JURONG" and "EAST". These words are processed line by line from top left to bottom right. The first word is compared to the list of all possible city area names. If it uniquely matches a single city area name, then this one is chosen. Otherwise the next word is also taken into account. This is repeated until we reach a unique match. In case no match has been found, the region is discarded and the process is repeated for the region with the next highest number.

For the building block numbers an example vote region is shown in Figure 4 on the left. Here, the target block numbers "238" and "239" are listed vertically in a table. We therefore use a similar procedure to determine the building block numbers except that the region is processed column-wise from top to bottom. All words with category "Building Block Number" inside the region are selected. If one column of building block numbers has been detected, the procedure aborts and the numbers inside this column are chosen.

The result after this is a city area name and a list of building block numbers detected for the city area map. Each combination of building block number and the city area name is compared to a publicly available dataset containing information about all public housing complexes in the city [3]. If the combination can be found, then the corresponding official address from the dataset is stored. Otherwise the combination is discarded.

[3] https://data.gov.sg/dataset/hdb-property-information

7.3 Building Height Estimation

For the building height estimation, we employ a simple approach of inferring the height based on the number of storeys for each building. This number can be extracted from the same public housing dataset as in the previous section. A couple of the documents provided by the city authorities in Singapore mention the heights of the storeys for several buildings. Based on this information, we determine the following formula to estimate the building height h from the number of storeys n:

$$h = h_{\text{first}} + (n-1)\, h_{\text{other}} + h_{\text{roof}} \tag{1}$$

where $h_{\text{first}} = 3.6\,\text{m}$ is the estimated height for first floor, $h_{\text{other}} = 2.7\,\text{m}$ the estimated height for every other floor and $h_{\text{roof}} = 1.35\,\text{m}$ the estimated height added by the roof.

The applicability of this formula is limited to the dataset of city area maps in Singapore only. There are a couple of options to get a more general estimation of building heights. In [5], statistics of additional building information taken from a database are used to improve the building height estimation. Stereo pairs of satellite images as in [9] or point clouds could be utilized, as well. These methods would, however, increase the overall complexity of the pipeline and getting the additional data on a city scale is not always feasible.

8 Building Shape and Size Extraction

With the location information (block number and street name) of the target buildings obtained from previous steps, here we describe how to extract the shape and size of each building based on it. Instead of directly processing the city area maps, one can make use of some open source online maps such as OpenStreetMap®. In the case of Singapore, we choose the new OneMap API [24]. It is developed by Singapore Land Authority and it is an official reference for building location and shape.

8.1 Motivation of Using the New OneMap API

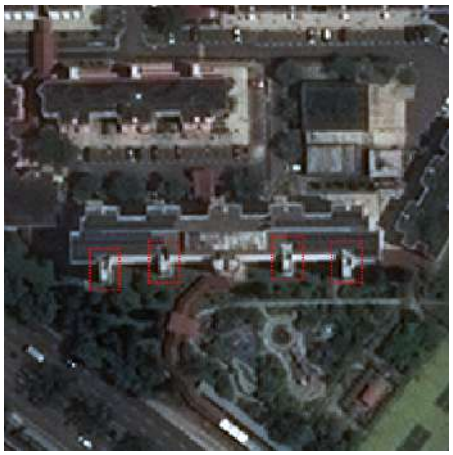
Processing an archived city area map as shown in Figure 2a may lead to inaccurate building shape and very long computational time. This is because most of the old maps were drawn by hand and afterwards digitalized by scanning.

During the scanning process, some undesired noise such as smudges and dots may be introduced, which makes the shape segmentation more complicated. Additionally, these maps were archived in the storage for years making some of

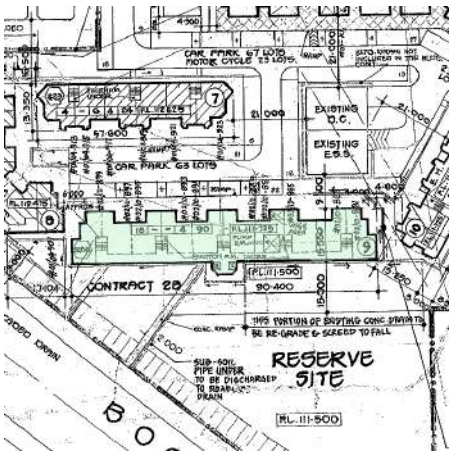
them have damages even before scanning. All these different kinds of noise and damages will not only introduce inaccuracies in the segmented building shape but also demand preprocessing to clean up the image. More sophisticated image processing techniques should be applied accordingly, which, of course, require longer processing time and more computational power.

Furthermore, the majority of the archived city area maps were drawn decades ago. Some newly constructed building parts are not shown in these old city area maps. For example, Figure 7a shows a capture of a building (Block No. 221A) in satellite view from OneMap and Figure 7b is its correspondence in the archived city area map. We highlight the target building in Figure 7b by marking it green. The shapes of the building in these two images are not identical because some extensions (highlighted with red boxes in Figure 7a) on one side of the facade is not shown in the old city area map. Hence, with such information unavailable, of course, we cannot get an accurate building shape from a city area map.

Lastly, one major limitation of archived city area maps is the lack of information about the precise location of buildings on the surface of Earth. That is, there is no information with regards to the latitude and longitude of buildings. This means it is impossible to correctly place the generated 3D models of buildings solely based on city area maps.



(a) Satellite image of Block No. 221A [24]



(b) Block No. 221A in the archived city area map (highlighted in green)

Fig. 7 Example of city area map not showing new constructions of a building

8.2 Usage of the New OneMap API

Due to the aforementioned reasons, instead of using the original archived city area map, our method extracts a more accurate building shape in a faster manner by leveraging the new OneMap API. Specifically, we use the new OneMap Search API and Static Map API [24].

The Search API returns the latitude and longitude of the target building center based on its block number and street name. The Static Map API returns an image of a map section based on some pre-defined parameters such as image center coordinates, zoom level, image size and so on. In our implementation, the image center coordinates are set as the latitude and longitude of the target building center, which are the outputs of the Search API. In order to get as many details as possible, we use the maximum zoom level and the biggest returned image size (512×512 pixels) provided by the Static Map API for image retrieval. By using the maximum zoom level, we have a fixed scale of the map which provides us with information about the actual building size.

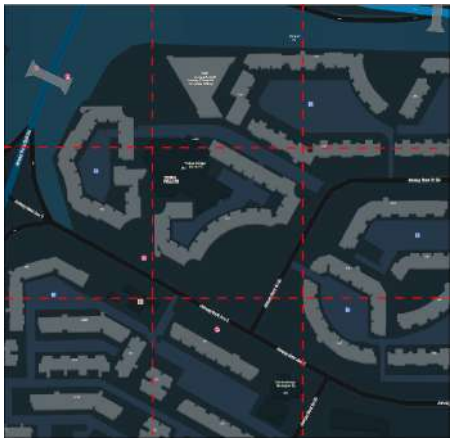
Figure 8a shows an example image retrieved by the Static Map API based on the center coordinates of a building Block No. 252, which is the target building in this case. In Section 8.3, we explain the necessity of getting Figure 8b. Only a grayscale version of this image will be processed for extracting the building shape. One can easily see that dealing with such a computer generated image will lead to a much more accurate segmentation result as well as a much shorter processing time, compared with working on a noisy and possibly damaged archived city area map.

8.3 Steps of Building Shape Extraction

Image Acquisition The Static Map API provides a maximum returned image size as 512×512 pixels, which might not be big enough to contain some very large buildings. Figure 8a shows a returned image of the Static Map API which has a building with its upper part cropped out. Such image will result in an incomplete building shape after segmentation.

In order to handle this situation, instead of retrieving only one 512×512 pixels image, we retrieve a 3-by-3 neighborhood of the target image. The tiled image center will still correspond to the latitude and longitude of the target building's center. The latitude and longitude of the surrounding

(a) Image retrieved for Block No. 252 without image tiling



(b) Image retrieved for Block No. 252 with image tiling

Fig. 8 Example of image retrieved with and without image tiling for a large building[24]

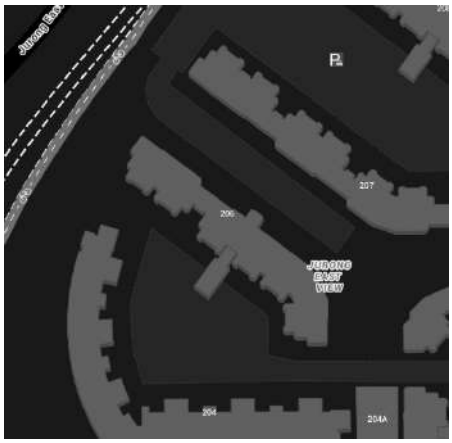


Fig. 9 Grayscale version of the image retrieved from the new OneMap Static Map API using the center coordinates of Block No. 206[24]

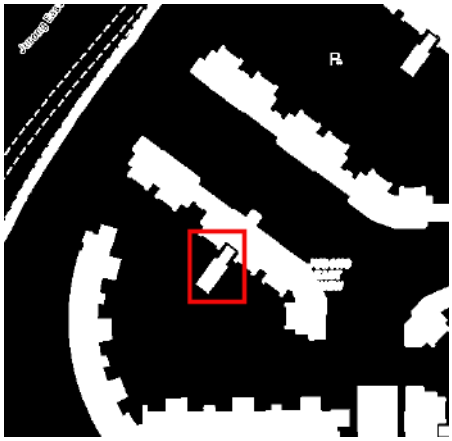


Fig. 10 Thresholding result with a contiguous noisy object indicated in red box

eight images' centers can be calculated from the center ones for image retrieval using the Static Map API.

Eventually we end up with an image having a size of 1536×1536 pixels. This size is big enough to hold an entire building and it does not add noticeable extra computational time. Figure 8b demonstrates such a tiled bigger image. For the purpose of illustration, it is resized to the same size as Figure 8a. The red dash lines indicate the boundaries of the neighborhood images.

Segmentation As mentioned before, only a grayscale version of the API returned images will be processed. In the retrieved images, all public housing buildings have the same grayscale value (confirmed with the map legend). Hence, one simple thresholding operation is able to segment all the target buildings successfully. However, here we do not use an optimal threshold to segment all the public housing buildings in one shot. Instead, we choose a slightly higher threshold to keep some annotations at the same time. This makes the extracted building shape complete for those cases where the annotations are overlapping with the building boundary. Then, we determine the target building shape based on the contour characteristics of all the objects remained after thresholding. We explain how to remove those overlapped texts to get the correct building shape in Section 8.4.

From now on, we use a different example building, Block No. 206, and set the retrieved image size as 512×512 pixels instead of the tiled one for illustration purpose. Figure 9 is the grayscale version of the API returned image and Figure 10 shows the result after thresholding.

Contour Processing Figure 10 shows that all the target building candidates and some noise are left after the thresholding operation. We conduct contour processing on such results for extracting the target building shape.

Fig. 11 Result of removing small objects and noise. Red dots are the centroids of contours and green dot is the image center

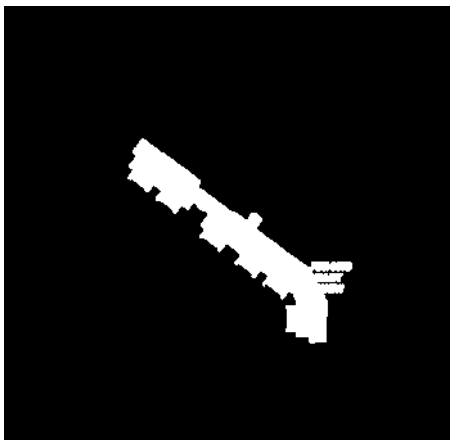


Fig. 12 Extracted shape without removing overlapped block name

Firstly, we remove all the objects whose contour areas are small. The main purpose of doing this is to get rid of contiguous objects around the target building, which, according to the original city area map, are not part of the target building. Figure 10 highlights such an object by drawing a red box around it. Additionally, some other noise sharing the same grayscale value are eliminated as well. The result image is shown in Figure 11.

Secondly, we pick out the target building among all the building candidates by choosing the contour whose centroid has the shortest distance to the image center. As explained in Section 8.2, the Static Map API uses the coordinates (in our case, latitude and longitude) of the target building's center as the retrieved image center coordinates, which means these two centers are overlapped. Therefore, the target building's contour centroid should be the closest one to the image center. This is shown in Figure 11 with all the centroids of the contours marked by red dots and the image center marked in green. This method works for tiled image too because the tiled image is centered at the target building's center as well.

Finally, the shape of the target building is extracted based on the selected contour. The result is shown in Figure 12 and we explain how to get the correct building shape without overlapped texts in the next section.

8.4 Overlapping Text Removal

As mentioned before, there are texts, for instance building block number or building name, overlapping with the target building boundary, such as Block No. 206 shown in Figure 9. Following the shape extraction steps described in Section 8.3, Figure 12 shows that the segmentation result of Block No. 206 includes both the target building and the overlapped texts on top of it. This distorts the extracted shape of the building and will eventually affect the shape of the extruded 3D model, which is shown in Figure 17a.

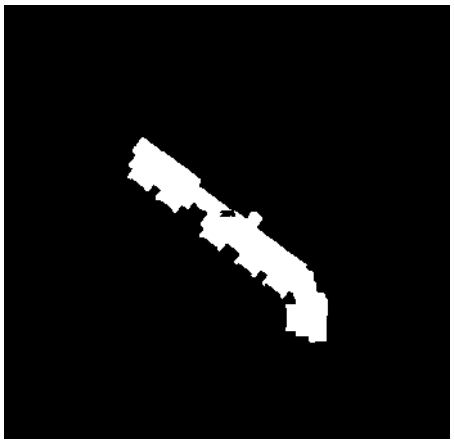
We solve this problem by processing the detected contour points. After extracting the target building with overlapped texts, we separate the texts apart from the building with a simple thresholding operation based on their different grayscale values. The shapes of the separated two parts are shown in Figure 13. Then, we remove the contour points of the text shape from the contour points of the target building shape. In the implementation, it is not that straightforward because there is usually no intersection of the two sets of contour points. So we remove all the contour points of the target building which are adjacent to the texts' contour points. The remaining points are used to generate the 3D model and the two ending points of the removed contour part are connected. Using this simple connection, the distorted part of the building shape is corrected. An illustration of the final extracted shape of Block No. 206 is shown in Figure 14.

9 3D Model Generation

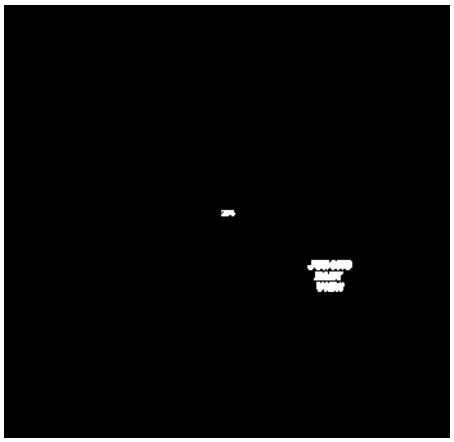
The goal of our pipeline is to not only be capable of generating 3D model of a single building but also to be able to generate 3D models of all the target buildings in a large neighborhood. More importantly, each 3D building model should have the correct aspect ratio and all the buildings in a neighborhood should have the correct relative location with each other, as in real world. We fulfill these goals simultaneously by converting the coordinate systems to a unified one.

9.1 Unified Coordinate System Selection

In our pipeline, at this stage, there are three different important parameters used to define a 3D model. However, they



(a) Target building shape



(b) Overlapped text shape

Fig. 13 Separation of target building and overlapped texts

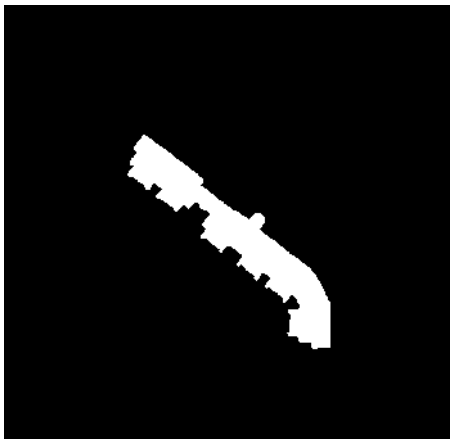


Fig. 14 Extracted shape of Block No. 206 after text removal

are in three different coordinate systems with three different units. First, for the height information of a building, the unit is meter. Second, for the location of a building, the Search API returns the latitude and longitude in decimal form. Third, for the retrieved shape of a building, the unit is pixel since it is segmented from a 2D image. Here, we choose the world coordinate system with latitude and longitude in decimal form to represent all the contour points. It is because among these three candidates it is the only one providing us with the information about how each building located with respect to each other. Also, although the buildings are extracted from separate retrieved images, their location information is well maintained within this coordinate system.

9.2 Coordinate System Conversion

We first do the conversion between building height and building shape. We accomplish it by converting the unit in meter to pixel based on the fixed map scale. Then we convert both height and shape with the unit of pixels into the unit of latitude and longitude in decimal format. It is achieved by measuring how much change in latitude and longitude with respect to one pixel. We call this amount of change *step*. Additionally, we need a reference point in each retrieved image, which we should know its exact position in the image and its latitude and longitude. The center pixel of each retrieved image is a good choice since the Static Map API uses the latitude and longitude of the target building center as the retrieved image center. We then compute how many pixels of shift in both directions for each contour point with reference to the image center pixel. Multiplying the shifts with *step* and adding the results to the latitude and longitude of the image center pixel will give us the contour points in the desired form.

9.3 3D Model Extrusion

With all the detected contour points converted, we generate the 3D model of a building. As a matter of fact, the building shapes extracted from the images returned by the Search API are not in a very high resolution, hence the extruded 3D models have zig zag edges wherever the edge is not vertical or horizontal. We smooth those edges by doing interpolation on all the contour points. After this, we extrude the 3D models by duplicating the contour points and lifting them according to the building height. Finally, we represent the 3D model in the CityGML LOD1 format.

10 Results

In this section, the pipeline is tested on real archived city area maps provided by city authorities in Singapore. Since

the pipeline is separated into six distinct steps (see Figure 1), we conduct tests for each of the steps individually. Whenever reasonable, we also report the results for several steps combined as in Section 10.4.

10.1 City Area Map Segmentation

The city area map segmentation described in Section 4 segments the input maps into smaller parts and filters out the irrelevant ones. The purpose of this is to increase the accuracy of the whole pipeline by reducing the chance of false detections originated from irrelevant parts of the map. Furthermore, it should increase the performance by reducing the amount of data which has to be processed by later steps. A way of evaluating its effectiveness is therefore to calculate the average fraction of the city area map which remains after filtering. Our dataset of city area maps contains eight maps with resolutions ranging from 13248×9222 pixels to 15433×13244 pixels. The result for this dataset is shown in Table 1. To measure the performance, we repeat the segmentation 10 times for each map. The average processing time per map and iteration is shown in Table 1, as well.

Table 1 Results for the city area map segmentation and filtering. The remaining area after filtering is averaged over all eight maps. The processing time was measured on a machine with an Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz. It is averaged over all maps and over 10 runs per map

Avg. Fraction of Remaining Area	Avg. Processing Time [s]
0.40	0.714

10.2 Text Detection

To evaluate the text detection approach described in Section 5, we created a test dataset. This dataset is different to the training dataset mentioned in Section 5 and is only used to evaluate the final performance of the text detection models after training. For this dataset we randomly sampled 52 cropped images from the eight city area maps. All images have a fixed size of 1280×720 pixels. For these images, we manually annotated the bounding boxes of all text elements. In total, the test dataset contains 378 ground truth bounding boxes.

As a baseline, we evaluate the performance of the pretrained model which we use as the starting point for training. This result is compared with the final model which we obtained after fine-tuning on our own training dataset as described in Section 5. The results are summarized in Table 2.

10.3 Text Recognition

The text recognition results are evaluated on the held-out test set as mentioned in Section 6. The final model was trained from scratch on the synthetic training dataset. It is therefore not possible to provide baseline results for it. Its results on the test dataset are shown in Table 3.

10.4 Building Information Extraction

Our dataset of city area maps consists of 10 maps which contain lists of target building blocks. From these, eight contain a city area name which we consider as human readable. These readable city area maps contain in total 27 target building block numbers. To evaluate this step of the pipeline, we count how many of these buildings can be successfully found in the public housing database, as described in Section 7. For this task, all previous steps of the pipeline contribute to the results. The results for the whole dataset and for each city area map individually are summarized in Table 4. To estimate the total runtime of all steps necessary to find the building information, we repeat them 10 times for each map. This yields the average runtime for each map: 8.497 s (averaged over all 10 runs). The measurement was conducted on a machine with Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz and GeForce RTX 2080 Ti GPU.

In Table 4, we can see that for two city area maps (No. 3 and 7), no buildings can be detected. These two cases are caused by failures in previous steps of the pipeline as can be seen in Figure 15. The actual keyword voting mechanism described in Section 7.1 is not responsible for any of the failure cases. For four of the eight maps, all target buildings could be successfully detected. For the two remaining maps which are partly correctly processed (No.5 and 6 in Table 4), the errors are caused by either the text detection step (block numbers are not or only partly detected) or the text recognition step (fully detected block numbers are wrongly classified).

10.5 Building Shape and Size Extraction

We first test the building shape and size extraction with all the 17 correctly detected blocks mentioned in Table 4. All blocks are correctly extracted. Then, for a better evaluation, another 83 public housing buildings are randomly selected from the public housing database. Combining with the aforementioned 17 blocks, we have a test set with 100 buildings in total. Since we perform shape extraction based on OneMap, we use the building shapes from it as the ground truth for evaluation. If an extraction has the same shape as shown in OneMap we treat it as correct, otherwise it is incorrect. The extraction results are shown in Table 5.

Table 2 Results for the pre-trained model compared to the final text detection model after fine tuning. The average precision (AP) metric metric is calculated according to the evaluation protocol from the COCO object detection challenge [21]. AP_{50} refers to the average precision calculated using an intersection over union (IoU) threshold of 50%. AP_{75} uses an IoU threshold of 75% and mAP is the mean average precision for all IoU thresholds from 50% to 95% in 5% steps

Model	mAP	AP_{50}	AP_{75}
Pre-trained on ICDAR 2013 and 2015	0.351	0.552	0.440
Fine-tuned on training data set	0.500	0.653	0.580

Table 3 Results for the final text recognition model. The metrics are calculated across all 3182 classes. To get single values for the metrics, we performed a weighted average where the weights reflect the number of instances per class. This is relevant because the residual class contains significantly more examples than the other classes (see Section 6). F_1 score is the unweighted harmonic mean between precision and recall

Model	Precision	Recall	F_1 score
Trained from scratch on synth. data	0.98	0.97	0.97

All of the four incorrect extractions are caused by overlapped texts covering a large portion of the building boundary. One incorrect case is shown in Figure 16. Figure 16a is the retrieved image from OneMap and cropped to show the details more clearly because Block No. 10C is a small building. One can see that the texts cover about one third of the target building and the covered building part contains critical information about the building shape. Thus, after text removal, the extracted building shape is incorrect and results in generating an incorrect 3D model as in Figure 16b. The generated 3D model has a triangle shape because our text removal method simply connects the two ending points in the set of the remained contour points of the building. The average shape extraction time for these 100 buildings is 0.072s, which was measured on a machine with an Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz. The time for retrieving an image from the Static Map API may vary according to the internet connection. For the 100 buildings in the test set, the average image retrieving time is 0.301s measured with the same machine as for shape extraction.

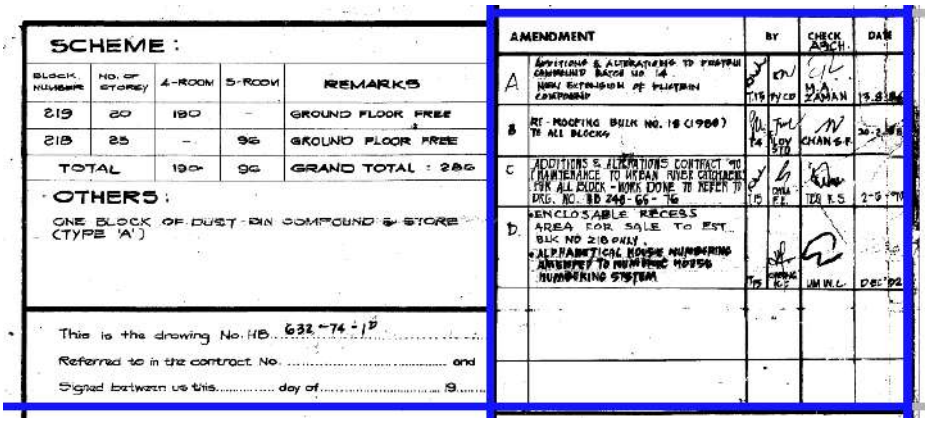
10.6 3D Model Generation

Single Building We first show the result of generated 3D model for a single building, which is the target building in Figure 9. Having the overlapping texts removed as described in Section 8.4, we get the generated 3D model in Figure 17b, which is the final result. We can see the differences between with and without text removal operation from Figure 17. The average time of generating the 3D model for a single building is 0.011s measured with all 100 buildings in the test set mentioned in Section 10.5 and on a machine with an Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz.

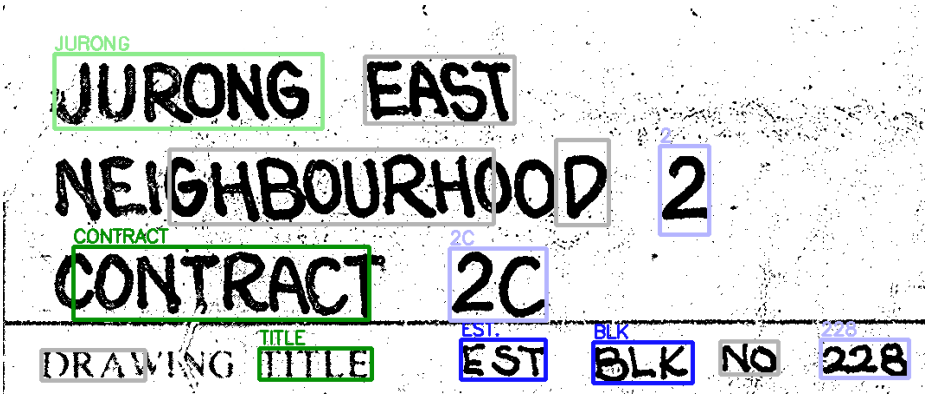
Buildings in a Large Neighborhood As we claimed before, our method not only can generate 3D building models with correct aspect ratios but also can generate buildings in a large neighborhood with correct relative locations. Figure 18 shows our generated 3D building models in an area from two different views. We evaluate our results by superimposing 3D building models on a 2D map. The 2D map in Figure

Table 4 Results of the first four steps in the pipeline to extract building information from city area maps

Map	Target Blocks	Detected Blocks	Correct Detections	False Detections	Precision	Recall
Map 1	3	3	3	0	1.00	1.00
Map 2	4	4	4	0	1.00	1.00
Map 3	2	0	0	0	0.00	0.00
Map 4	2	2	2	0	1.00	1.00
Map 5	8	4	2	2	0.50	0.25
Map 6	5	4	4	0	1.00	0.80
Map 7	1	0	0	0	0.00	0.00
Map 8	2	2	2	0	1.00	1.00
All	27	19	17	2	0.89	0.63



(a) Excerpt of the city area map No. 3 in Table 4 after the segmentation step. The blue boxes show the remaining segments after filtering. The list of building blocks on the left (under the keyword 'SCHEME') is not contained in a box and will be ignored. Therefore no building blocks for this map can be detected in later steps



(b) Excerpt of the city area map No. 7 in Table 4 after the text recognition step. The city area name was detected by the text detection step. However, the text recognition could only detect the first word 'Jurong' (indicated by the light green box). The second word 'East' is wrongly marked as irrelevant (indicated by the gray box). The word 'Jurong' appears several times in the list of possible city area names. Therefore no city area can be uniquely assigned to this map and hence no building blocks identified

Fig. 15 Illustration of the failure cases encountered in Table 4

Table 5 Results of the shape extraction

Total Blocks	Extracted Blocks	Correct Extraction	Incorrect Extraction
100	100	96	4

18 is extracted from OpenStreetMap® [4] using the latitude and longitude values of this neighborhood. Figure 18 shows qualitatively the 3D building models are placed at correct locations on top of this map. Therefore, we can conclude that having the coordinate systems converted according to Section 9.1 and Section 9.2, our generated CityGML models are placed at the correct locations.

[4] ©OpenStreetMap contributors, the data is available under the Open Database License, https://www.openstreetmap.org/copyright.

11 Discussion

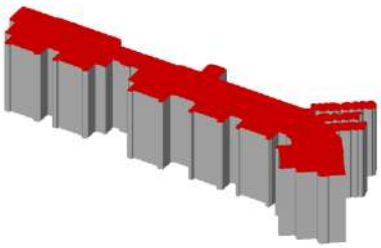
11.1 City Area Map Segmentation

In Table 1, we can see that, after filtering out the irrelevant segments, we are left with only 40 % of the original city area map on average. This has two positive effects on the overall pipeline. Firstly, it reduces possible errors, which might be originated in the areas of the map which are filtered out. It could for example be that a wrongly classified city area name is by chance surrounded with several keywords for the city area name. This might cause the whole map to be assigned to this wrong city area. By filtering out large parts of irrelevant data, the chances for such errors are reduced. Sec-

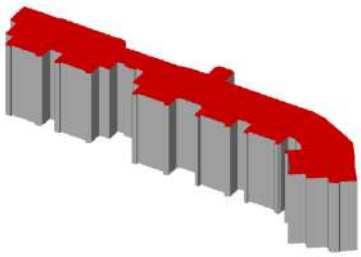
(a) Cropped retrieved image for Block No.10C [24]

(b) 3D model generated based on incorrect shape extraction

Fig. 16 One incorrect shape extraction case



(a) Extruded LOD1 model without text removal



(b) Extruded LOD1 model with text removal

Fig. 17 Generated 3D model for a single building

ondly, because only a fraction of the original data is further processed, the runtime and memory footprint of the whole pipeline is reduced. However, a drawback of this step is that, it adds additional complexity and therefore another possible error source. This is exemplified by Figure 15a, where the segmentation step filtered out a relevant part of the city area map. To reduce the probability for cases like this, the hyper-parameters can be adjusted to filter out a smaller fraction of the original map which would, however, also reduce the positive effects. It therefore boils down to choosing a reasonable balance which depends on the application.

In general, this map segmentation step could be useful for similar scenarios in which an input document is divided by horizontal an vertical lines into smaller segments.

11.2 Text Detection

The results in Table 2 are based on our own dataset and hence cannot be compared to other works. Nevertheless, it can be seen that fine tuning the model on the training dataset improves the average precision on the test dataset significantly. As the test dataset was generated from the real city area maps, it can be assumed that the fine tuning improves the overall performance of the pipeline. As described in Section 5, our text detection method is based on an existing method and implementation. This existing method was originally developed in the context of scene text detection based on color pictures taken with the Google Glass. Our results demonstrate its applicability in the domain of grayscale document text detection after fine tuning only on a small dataset of 104 city area segments.

11.3 Text Recognition

The results in Table 3 look promising, but they are difficult to compare with other approaches because they were obtained on our own synthetic dataset. However, the overall results of the pipeline indicate that it translates to the real data as well. Together, this confirms that the approach from [18] to obtain a working text recognition model by training on a synthetic dataset works. This is in general useful because it removes the dependency on large labeled training datasets, which is often a limiting factor in the application of deep learning methods in practice. We could obtain our results without having to manually annotate a single word image. Nevertheless, the failure case in Figure 15b shows that a better text recognition performance is still desirable to improve the overall performance of the pipeline. For example, more

(a) A view of 3D building models in a large neighborhood

(b) Another view of 3D building models in a large neighborhood

Fig. 18 Generated 3D models for a large neighborhood

model architectures than the one described in [18] can be tested for a better performance on our dataset. Another option is to create a small dataset based on our data and use it for fine tuning similar to Section 5.

11.4 Building Information Extraction

The results in Table 4 and the failure cases in Figure 15 demonstrate the complexity of this task. There are several possible error sources which might result in a single building block or the whole list of building blocks to be not correctly identified. Nevertheless, for four out of eight city area maps, all buildings could be successfully detected. In the remaining maps, none of the errors can be traced to the keyword voting

mechanism step described in Section 7.1. This indicates that it works quite robustly on our dataset.

The results demonstrate only one possible application of the building information extraction steps. In general, the same approach can be used when other information indicated by previously-known keywords has to be extracted from similar map documents.

11.5 Building Shape and Size Extraction

Table 5 shows that our shape extraction method is able to segment the shapes of all the public housing buildings. The failure cases of the method, regarding the correctness of the extracted shapes, are coming from texts or other annotations overlapping a large portion of the buildings. In these cases, the lost of information is too much. Therefore, our method actually can achieve a good result in a simple and fast manner.

11.6 3D Model Generation

Figures 17 and 18 show that we accomplish the goal of generating 3D models with both correct aspect ratios and relative locations. It is difficult to measure how accurate the generated models are because otherwise we need the ground truth data of all the generated buildings. The results shown in Figure 18 look quite convincing and can already be used for visualisation and simulation in urban planning. Furthermore, our CityGML models are with the correct world coordinates, which can be easily mapped to a virtual map provider or extended to an actual terrain data for a more sophisticated visualisation.

12 Conclusions and Future Work

In this paper, we have presented a pipeline to automatically generate 3D building models in CityGML LOD1 format from archived city area maps. Our proposed method tackles the challenges of handling low quality scanned archived maps by fusing information from multiple sources. We proposed to extract the essential building block numbers and city area name information from an input archived map. Based on this information, in order to generate 3D building models which are described in the input map, we proposed to integrate and utilize the number of storeys and address information from a public housing database as well as retrieving and extracting the shape, size and actual location of the buildings from an online city map API. Our experimental results demonstrated that the proposed pipeline is able to generate correct 3D building models from archived city area maps provided by a city authority. As such, the proposed pipeline as a whole is the main contribution of this paper. In addition, we showed that several of the individual steps of the pipeline solve tasks which can be useful in other contexts, as well (Section 11.1 and 11.4). For the text detection and recognition steps we used already existing deep learning methods. Our contribution here is to demonstrate their application in a different context by leveraging only a little amount of labeled data (Section 11.2) or pure synthetic data (Section 11.3).

In the future, we plan to generate 3D building models in higher LOD, for example, adding roofs, windows, doors and room details. We also plan to fuse other information to enhance the 3D building models such as adding textures based on online photos. Furthermore, by using this information we plan to improve the automatic building height estimation, similar to the work in [5].

Acknowledgements We gratefully thank the Housing & Development Board (HDB) for providing us the archived city area maps. We also want to express our appreciation to the new OneMap, which developed by the Singapore Land Authority, for providing us their APIs.

References

1. Ahmed, S., Liwicki, M., Weber, M., Dengel, A.: Improved automatic analysis of architectural floor plans. In: 2011 International Conference on Document Analysis and Recognition, pp. 864–869 (2011)
2. Ahmed, S., Liwicki, M., Weber, M., Dengel, A.: Automatic room detection and room labeling from architectural floor plans. In: 2012 10th IAPR International Workshop on Document Analysis Systems, pp. 339–343 (2012)
3. Ahmed, S., Weber, M., Liwicki, M., Dengel, A.: Text/graphics segmentation in architectural floor plans. In: 2011 International Conference on Document Analysis and Recognition, pp. 734–738 (2011)
4. Biljecki, F., Ledoux, H., Stoter, J.: An improved LOD specification for 3D building models. Computers, Environment and Urban Systems **59**, 25–37 (2016)
5. Biljecki, F., Ledoux, H., Stoter, J.: Generating 3D city models without elevation data. Computers, Environment and Urban Systems **64**, 1–18 (2017)
6. Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., ÃĞÃűltekin, A.: Applications of 3d city models: State of the art review. ISPRS International Journal of Geo-Information **4**(4), 2842–2889 (2015)
7. Breuel, T.M.: The ocropus open source ocr system (2008)
8. de las Heras, L., FernÃąndez, D., Valveny, E., LladÃşs, J., SÃąnchez, G.: Unsupervised wall detector in architectural floor plans. In: 2013 12th International Conference on Document Analysis and Recognition, pp. 1245–1249 (2013)
9. Duan, L., Lafarge, F.: Towards large-scale city reconstruction from satellites. In: B. Leibe, J. Matas, N. Sebe, M. Welling (eds.) Computer Vision – ECCV 2016, pp. 89–104. Springer International Publishing, Cham (2016)
10. Duda, R.O., Hart, P.E.: Use of the hough transformation to detect lines and curves in pictures. Commun. ACM **15**(1), 11–15 (1972)
11. Fritsch, D., Klein, M.: 3D preservation of buildings – reconstructing the past. Multimedia Tools and Applications **77**(7), 9153–9170 (2018)
12. Gerstweiler, G., Furlan, L., Timofeev, M., Kaufmann, H.: Extraction of structural and semantic data from 2d floor plans for interactive and immersive vr real estate exploration. Technologies **6**(4) (2018)

13. Gimenez, L., Hippolyte, J.L., Robert, S., Suard, F., Zreik, K.: Review: reconstruction of 3d building information models from 2d scanned plans. Journal of Building Engineering **2**, 24 – 35 (2015)
14. Gimenez, L., Robert, S., Suard, F., Zreik, K.: Automatic reconstruction of 3d building models from scanned 2d floor plans. Automation in Construction **63**, 48 – 56 (2016)
15. Goetz, M.: Towards generating highly detailed 3d citygml models from openstreetmap. International Journal of Geographical Information Science **27**(5), 845–865 (2013)
16. Gröger, G., Kolbe, T., Nagel, C., Häfele, K.: OGC city geography markup language (CityGML) encoding standard, version 2.0, ogc doc no. 12-019. Open Geospatial Consortium (2012)
17. Haklay, M., Weber, P.: Openstreetmap: User-generated street maps. IEEE Pervasive Computing **7**(4), 12–18 (2008)
18. Jaderberg, M., Simonyan, K., Vedaldi, A., Zisserman, A.: Reading text in the wild with convolutional neural networks. International Journal of Computer Vision **116**(1), 1–20 (2016)
19. Karatzas, D., Gomez-Bigorda, L., Nicolaou, A., Ghosh, S., Bagdanov, A., Iwamura, M., Matas, J., Neumann, L., Chandrasekhar, V.R., Lu, S., Shafait, F., Uchida, S., Valveny, E.: Icdar 2015 competition on robust reading. In: 2015 13th International Conference on Document Analysis and Recognition (ICDAR), pp. 1156–1160 (2015)
20. Karatzas, D., Shafait, F., Uchida, S., Iwamura, M., i. Bigorda, L.G., Mestre, S.R., Mas, J., Mota, D.F., AlmazÃăn, J.A., de las Heras, L.P.: Icdar 2013 robust reading competition. In: 2013 12th International Conference on Document Analysis and Recognition, pp. 1484–1493 (2013)
21. Lin, T., Maire, M., Belongie, S.J., Bourdev, L.D., Girshick, R.B., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: common objects in context. CoRR **abs/1405.0312** (2014)
22. Macé, S., Locteau, H., Valveny, E., Tabbone, S.: A system to detect rooms in architectural floor plan images. In: Proceedings of the 9th IAPR International Workshop on Document Analysis Systems, DAS '10, pp. 167–174. ACM, New York, NY, USA (2010)
23. Martel, R., Dong, C., Chen, K., Johan, H., Erdt, M.: Generation of 3d building models from city area maps. In: Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 4: VISAPP,, pp. 569–576. INSTICC, SciTePress (2019)
24. NewOneMap: New OneMap Search API, https://docs.onemap.sg/#search. New OneMap Static Map API, https://docs.onemap.sg/#static-map Contains information from new OneMap accessed on 13th November 2018 from new OneMap Static Map API, which is made available under the terms of the Singapore Open Data Licence version 1.0: https://www.onemap.sg/legal/opendatalicence.html. (2018)
25. Over, M., Schilling, A., Neubauer, S., Zipf, A.: Generating web-based 3d city models from openstreetmap: The current situation in germany. Computers, Environment and Urban Systems **34**(6), 496 – 507 (2010)
26. Smith, R.: An overview of the tesseract OCR engine. In: Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02, ICDAR '07, pp. 629–633. IEEE Computer Society, Washington, DC, USA (2007)
27. Soille, P.: On morphological operators based on rank filters. Pattern Recognition **35**(2), 527 – 535 (2002)
28. Sugihara, K., Murase, T., Zhou, X.: Automatic generation of 3D building models from building polygons on digital maps. In: 2015 International Conference on 3D Imaging (IC3D), pp. 1–9 (2015)
29. Tombre, K., Tabbone, S., Pélissier, L., Lamiroy, B., Dosch, P.: Text/graphics separation revisited. In: D. Lopresti, J. Hu, R. Kashi (eds.) Document Analysis Systems V, pp. 200–211. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
30. Yang, B., Lee, J.: Improving accuracy of automated 3-d building models for smart cities. International Journal of Digital Earth **12**(2), 209–227 (2019)
31. Zhou, X., Yao, C., Wen, H., Wang, Y., Zhou, S., He, W., Liang, J.: EAST: an efficient and accurate scene text detector. CoRR **abs/1704.03155** (2017)