# Automatic Virtual 3D City Generation for Synthetic Data Collection

Bingyu Shen[1], Boyang Li[1], Walter J. Scheirer[1]
[1]University of Notre Dame, Notre Dame, Indiana
{bshen@nd.edu, bli1@nd.edu, walter.scheirer@nd.edu}

## Abstract

*Computer vision has achieved superior results with the rapid development of new techniques in deep neural networks. Object detection in the wild is a core task in computer vision, and already has many successful applications in the real world. However, deep neural networks for object detection usually consist of hundreds, and sometimes even thousands, of layers. Training such networks is challenging, and training data has a fundamental impact on model performance. Because data collection and annotation are expensive and labor-intensive, lots of data augmentation methods have been proposed to generate synthetic data for neural network training. Most of those methods focus on manipulating 2D images. In contrast to that, in this paper, we leverage the realistic visual effects of 3D environments and propose a new way of generating synthetic data for computer vision tasks related to city scenes. Specifically, we describe a pipeline that can generate a 3D city model from an input of a 2D image that portrays the layout design of a city. This pipeline also takes optional parameters to further customize the output 3D city model. Using our pipeline, a virtual 3D city model with high-quality textures can be generated within seconds, and the output is an object ready to render. The model generated will assist people with limited 3D development knowledge to create high quality city scenes for different needs. As examples, we show the use of generated 3D city models as the synthetic data source for a scene text detection task and a traffic sign detection task. Both qualitative and quantitative results show that the generated virtual city is a good match to real-world data and potentially can benefit other computer vision tasks with similar contexts.*

## 1. Introduction

Computer vision tasks have witnessed rapid progress thanks to the success of convolutional neural networks. As the neural networks get deeper, the model performance gets better, opening up new applications in the real world. However, deep neural networks are data-driven. Training a robust and accurate network with millions of parameters requires a significant amount of images. For example, Im-

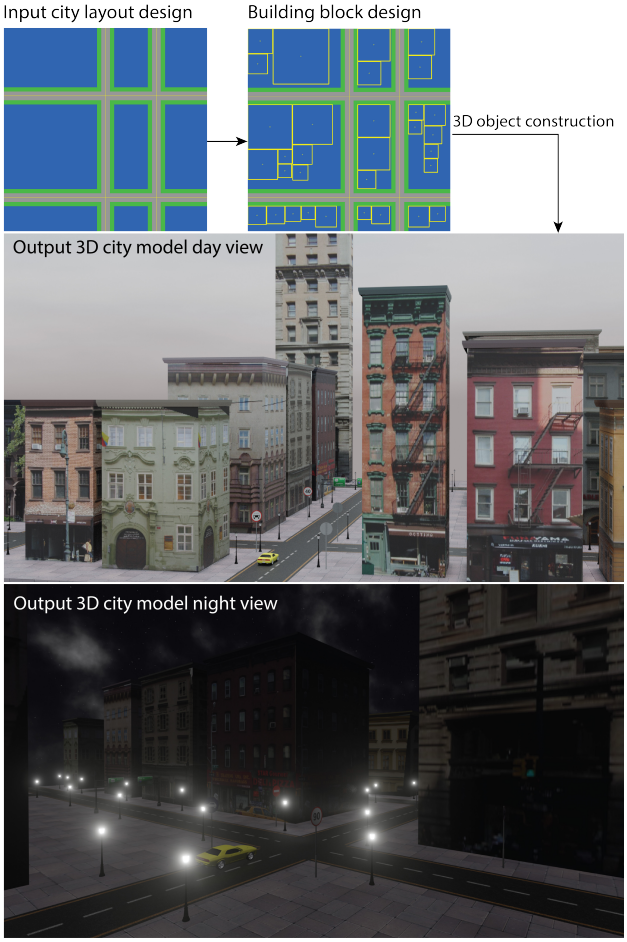



Figure 1: In this paper we introduce a pipeline that takes a simple 2D layout image as input and outputs a virtual 3D city model according to the specified design. The output model consists of important city elements: building blocks, road systems and street lamps. Each building is textured with a real-world building facade. Specific objects, such as road signs, can be added to these scenes to provide additional training and evaluation data for deep learning-based detection algorithms.

ageNet [19] is a commonly used dataset comprised of approximately 1 million images and 1000 object classes. The same thing is true for more specialize object cases like scene text detection. The ICDAR 2015 Competition on Robust Reading Dataset [13] consists of tens of thousands of video frames and over 200,000 annotations within them. As data collection and annotation are extremely expensive, it is not a rare occurrence to take a model pre-trained on a large-scale dataset as the backbone and then fine-tune it for a specific task with a small-scale dataset.

Much effort has been spent developing techniques for dataset augmentation [30]. A widely used methodology is the manipulation of images with image processing operations such as geometric transformations (e.g., cropping, rotation, distortion), and random deformations (adding noise) [25]. Another direction of dataset augmentation is to generate synthetic images of the target category. For example, Generative Adversarial Networks (GANs) have been used to generate handwritten digit images [12] to improve the classification performance of OCR models. SyntheText [10] generates synthetic images for scene text recognition by placing texts into natural images. While such methods proved to be effective in improving the model performance for a specific task, challenges remain in applying them to other problems.

For example, autonomous driving is a combination of a lot of computer vision algorithms that work with city scenes. Algorithms aimed at different tasks are integrated to make quick reactions to various signals captured by sensors and cameras. Those tasks include, but are not limited to, object detection, scene text recognition, and pedestrian behavior prediction. Algorithms usually are trained on datasets consisting of video clips and images collected from the real-world such as KITTI [9]. Existing dataset augmentation methods are restricted from generating videos or images with a variety of city scenes. Therefore, it's not uncommon to utilize virtual scenes to train and test autonomous driving algorithms [3, 1].

Compared with existing dataset augmentation techniques, virtual scenes are extremely salable and also offer environment parameters to simulate different weather conditions. Despite the advantages, generating a city-scale scene with accurate object models from a layout design can be a labor-intensive and time-consuming task even for an expert. This limits the application of virtual scenes to provide training data for computer vision tasks taking images or videos captured by camera as inputs. As an alternative, automatic 3D city model generation with high-quality and versatile textures allow generating an unlimited amount of data with controlled designs and accurate annotations, while saving the effort spent on developing precise object meshes by hand.

In this paper, we propose a pipeline that can automatically generate a virtual 3D city model within seconds from the input of an image. As shown in Fig. 1, the input of the pipeline is an image that describes a city layout design and the output is a 3D model with buildings, roads, and street lamps placed in the corresponding location. To generate a highly restored city scene, we apply real-world exterior textures to each building. The output 3D model is ready to use with popular 3D render engines.

There are three major contributions in this work:

1. We designed a pipeline that allows users to generate virtual 3D city models quickly from the customized layout design. The layout is represented by color blocks, and therefore the input images can be created using any illustration software, which makes the entire pipeline easy to use.

2. The generation process is quick and offers multiple options for customization. The entire scene generation takes 35 seconds on average.

3. Finally, to prove the resulting 3D city model is applicable for downstream tasks, two experiments are conducted. 1. We trained a scene text detector images from the rendered scene. 2. We trained a traffic sign detector with images from the rendered daytime scene and adapted it to the images from the rendered nighttime scene. Both the qualitative and quantitative results prove the quality and effectiveness of our generated models.

All data and code will be released following publication.

## 2. Related Work

### 2.1. 3D city generation from various inputs

Procedural modeling is one of the most widely used methods to quickly generate a complex scene with a large number of components, as it can be cumbersome to generate them manually. It computes output models based on rule sets and grammars. Grammars are used to gradually add more details to models at different levels according to requirements [21]. CityEngine [16] first proposed a system based on procedural modeling and L-systems [17] to generate 3D models for real cities. It deconstructs the problem of city generation into road network construction and building construction. To reconstruct the traffic system, CityEngine requires an input of geographical maps and sociostatistical maps. Later the commercial version of CityEngine, Esri City Engine [23] improved on the pipeline to add photo-realism. However, the input of Esri CityEngine is still GIS information, like CityEngine, which constrains the source of the generated model. The output of CityEngine is a CityGml file, which cannot be edited and loaded directly

by popular 3D development software like Blender [5]. Although it can be rendered into 3D models afterward, the transfer between different model formats and development tools increases the difficulty in use.

Random3DCity [4] also utilizes procedural modeling to generate city models into CityGML files. Instead of reading geographical images, it randomly creates a 3D city scene with $n$ buildings, where $n$ can be any numerical input. The buildings are placed in a $\sqrt{n} \times \sqrt{n}$ grid and the system allows certain angles of rotation. Dylla et al. [8] employed CityEngine to reconstruct ancient Rome with an output model of 400 million polygons, which offers an extremely fine level of geometric detail. Although procedural modeling is not easy to start with for beginners, the resulting 3D models can be of sufficient quality for use in computer vision tasks.

As deep learning is now operating in multiple areas of computer vision, we find an increasing number of works leveraging neural networks throughout the process of 3D city generation. Beer [2] used GANs to learn the segmentation of buildings from an orthographic city image and estimate the heights of city buildings. The result is a Level of Detail (LOD) 1 modeling of the buildings in the input image in CityGML format.

To summarize, existing works can generate 3D city models with fine details. However, there are some obvious gaps that impede them to be widely used outside of the computer graphics field. Procedural modeling is a rule-based coding language. Although through definition, objects with complex structures and details can be generated, it is not an easy job for users to develop a complex rule set or grammars to construct an entire city from scratch. Besides that, the input of GIS files allows for very limited customization. While Beer [2] reconstructs a 3D city using only 2D input images, the reconstructed 3D scenes lack textures and road systems, which makes them impractical to use for computer vision tasks expecting real-world input.

### 2.2. Generation of 3D models for buildings

Similar to 3D city generation, procedural modeling has been applied widely for generating 3D models of buildings. Müller et al. [14] developed a computer graphics architecture(CGA) shape grammar that can produce high visual quality and geometric details for architectural models. So 3D models can be created at a large volume at a low cost. The CGA grammar addressed context-sensitive shape rules and is robust to various architectural shapes and structures. In Nishida et al. [15], the authors proposed using a CNN to learn the parameters of procedural grammars. Given an outline of an architecture's silhouette, their method can generate the grammar parameters for all building components from large-scale building structure to fine-scale windows and door geometry.

Deep neural networks have also been applied in this problem to facilitate the generation of object textures. Isola et al. [11] developed an adversarial neural network that can generate the exterior of a building according to an input image, which depicts its facade design. The model automatically learns not only the mapping between the input and output images, but also the loss function to train such mapping. It has been successfully applied to learn realistic textures of buildings from an input of a building's facade.

To avoid the complexity of procedural modeling, in this paper we propose to generate 3D city models from 2D input images. The output is a 3D object that is ready to be loaded by most 3D rendering engines. The pipeline can quickly generate a 3D city model with buildings, a road system and street lamps. The model can be customized by users to insert objects of interest. All components in the scene will be rendered with high-quality textures. To the best of our knowledge, this is the first attempt at bridging the gap between user input as simple 2D images and a high-quality 3D model output that is ready to be used for object recognition tasks.

## 3. Virtual 3D City Generation Pipeline

The outline of our proposed processing pipeline is shown in Fig. 2. The input is an image representing the layout of a city using different color blocks. The pipeline will take this input and detect boundaries for the color blocks and extract coordinates to place corresponding city elements. Finally, object models will be placed into the scene with textures assigned to complete the 3D city model.

### 3.1. Layout design and parsing

We designed a simple algorithm to automatically generate layout designs with a default size of $768 \times 768$ pixels. Three types of objects that convey the key information of a city layout are included in the output image: building blocks, roads, and street lamps. Roads are represented by greys lines with a width of 32 pixels and will be randomly placed vertically and horizontally. The areas cropped by road lines will be filled with building blocks, which are represented by blue pixels. Each blue block will be surrounded by green pixels with a width of 8 that represents available space for street lamps. Some output layout designs are shown in Fig. 3. Given an input image with different colors representing different city elements, the next step is to extract the coordinates of them and place corresponding objects into the output 3D city model.

***Coordinates of buildings.*** We use a simple find contour algorithm [7] to extract the bounding box coordinates for each blue block. In the real world, buildings are usually grouped together and compose a building block. To generate a similar city scene, each blue area is defined as a building block, and buildings are placed into the block

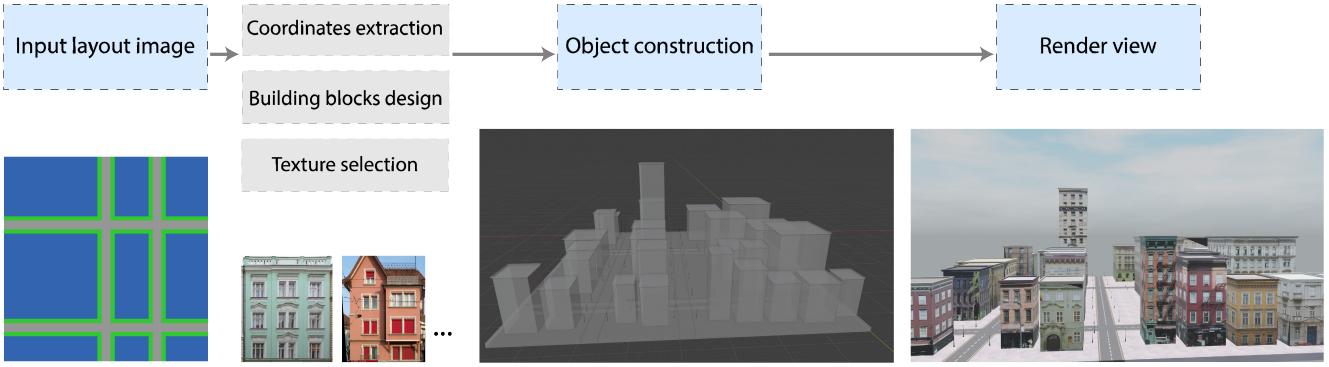



Figure 2: An overview of our 3D city model generation pipeline. The input image contains a layout design of the city where different colors represent different city elements. The output from object construction will automatically add textures. The final 3D model can be rendered in 3D development software.

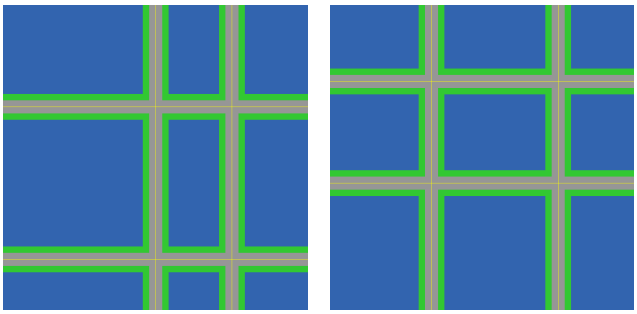


Figure 3: Examples of automatic layout design. Blocks with different colors represent areas to place different objects: pixels in blue represent building blocks, grey lines represent the road system, and pixels in green represent available areas to place street lamps.

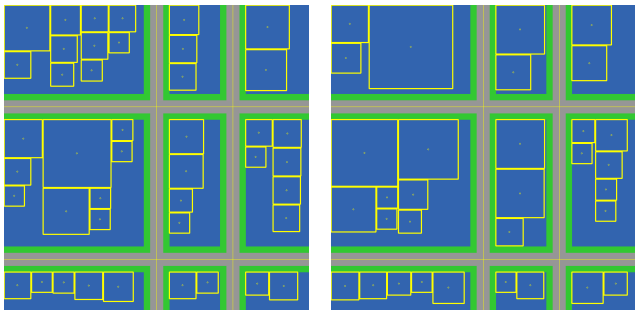


Figure 4: Examples of generated building plans using Algorithm 1 given the same input layout. Yellow squares represent buildings and the dimensions of squares are proportional to the size of the building to be placed.

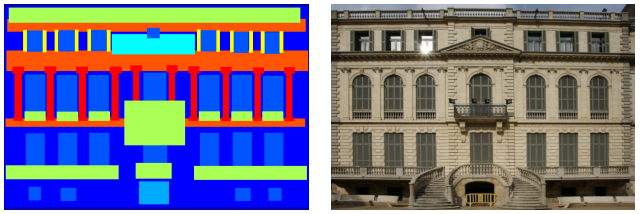


Figure 5: Facade of a building and its exterior. We cropped the image and calculate the dimension of buildings according to the facade annotation.

following Algorithm 1. In our pipeline, the dimensions of buildings to be placed are pre-defined and each building is a cuboid with a square bottom. Given a list of building widths and a block's size, we start placing buildings from the top left by randomly selecting one building that fits into the block. Then the algorithm continues to the area on the right of the placed building and randomly selects other buildings that fit the updated dimension constraints. The area below the placed buildings will recursively call the algorithm. As shown in Fig. 4, the algorithm can place buildings with a proper density into each building block under the dimension constraints.

***Coordinates of road system.*** Each road is defined by its center coordinates, width, length, and rotation angle. We draw a one-pixel yellow line in the center of each road to help coordinate extraction when the road is generated. Firstly all the yellow-lines in the original image are filtered out. If there are consecutive pixels longer than 5 pixels, the start and end pixel location will be captured. Accordingly, we can calculate the center of the road, as well as its length. The width of the road is determined using a gray-scale image converted from the original input. Given the center of a vertical road, the width will be the number of consecutive pixels in the x-axis, and vice versa for horizontal roads.

***Coordinates of street lamps.*** Street lamps are indispensable elements for a city scene. In the layout design, green areas that are along both sides of the roads are space available for street lamps. Street lamps will be placed every 100 pixels along the road. No street lamps will be placed on the interaction area of two roads. One lamp will be deleted arbitrarily if two street lamps are too close to each other.

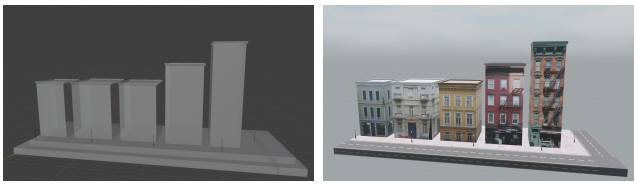


Figure 6: Building models with CMP facade image textures. Left: 3D objects' meshes, Right: rendered view of the buildings with textures added.

### 3.2. Assign real-world facade textures to buildings

To create real-world city scenes, we use the CMP dataset [22] to generate the building textures. The CMP dataset contains 600 rectified images of facades from various sources, and all images have been manually annotated. As shown in Fig. 5, annotation includes 11 classes of building components such as window, door, balcony, etc. We further annotated the dataset with the floor number for each facade. By fixing the floor height, the dimensions of all buildings are proportionally scaled. To prepare a building texture, images are first cropped to contain only part of the facade below the first molding. Given a cropped facade image of size $cw \times cy$ and its floor number $h$, the building dimensions applying the image texture will be $(cw/cy) \times (cy/h \times 3)$. As shown in Fig. 6, the width and height of buildings are proportional and realistic. All buildings have consistent floor height, which further enhances the visual quality.

Roads and street lamps are assigned with uniform default textures. The texture of roofs are randomly selected from the several provided textures that best match the color of the molding in the building texture.

**Algorithm 1:** Building Placement

**Result:** Global coordinates for center and dimension of each building in the block
Input: block's top left corner $(x_1, y_1)$, block's right bottom corner $(x_2, y_2)$, list of buildings' widths;
**while** *Buildings already placed do not exceed block's dimension horizontally* **do**
  Randomly select a building from the list which satisfies the dimension requirements;
  **if** *Right bottom corner of the building is within the block* **then**
    Add the current building to the result;
    Recursively call this function with the area below current building;
    Horizontally move to the area to the right of the current building with gap between buildings added;
  **else**
    exit;

Figure 7: Rendered city scenes with road signs inserted. The bounding boxes of objects are automatically calculated.

### 3.3. Scaling

The output 3D city model has a default size of $n \times m$ meters squared where $n$ and $m$ are dimensions of the input layout image. We also provide a function to quickly scale up the city by mirroring the input layout city. Currently, three types of mirroring are supported: flip vertically(1), flip horizontally(2), and flip both horizontally and vertically(3). The user can create a new layout by specifying a combination of mirroring. In this way, a simple layout can be quickly replicated to generate a more complex city model.

### 3.4. Render view

The output of our pipeline is an object file with textures. It can be imported and edited by most of the popular 3D modeling software. Depending on the requirements, it can also be exported to other formats to be rendered by different rendering engines. Through rendering, the generated 3D city can be utilized in many different ways.

For instance, we can use the model as a generator of training data for computer vision tasks related to city scenes. Given the coordinates of objects, the bounding box of a specific object in the rendered view can be calculated. By adjusting the position of objects and the position of the camera, we can generate an unlimited amount of data with high-quality city scenes to train object detection algorithms. As shown in Fig. 7, we can easily create city scenes with dif-


Figure 8: Scene text can be inserted as signs on the buildings. The distortion of texts is realistic as the camera can capture the same building from different angles.

ferent detection targets such as cars and road signs. Through the control of the environment properties, we can simulate different conditions such as time of day and weather. While real-world datasets are extremely expensive to collect and get annotation, our pipeline provides an alternative with high-quality visual effects.

In this paper, we provide an example of using the generated 3D city model to collect datasets for a scene text detection task. As shown in Fig. 8, texts can be placed on the walls of buildings in the generated city. Compared with generating synthetic scene text images by inserting texts into still images, rendered images using our output 3D model are with realistic characteristics found in the wild scenes, such as distortion and reflection. Additionally, using our pipeline, we can create training data for difficult cases of scene text such as round texts and texts that have been partially occluded by other objects. Such datasets will contribute to training a robust algorithm applicable to many different scenarios.

# 4. Experiments

We generated 100 city models to collect data and demonstrate the quality of rendered virtual cities and the speed performance of our pipeline. The experiments are conducted on a workstation with an Intel Core i9-9900K CPU and two 1080 Ti GPUs. We evaluated different aspects of the 3D city models including model generation time, model quality and rendered view experience.

| Pipeline | Execution time | |
|---|---|---|
| | AVG. (sec) | STD. (sec) |
| Layout design | 1.372 | 0.021 |
| Coordinates extraction | 1.168 | 0.076 |
| 3D model generation | 32.792 | 18.947 |
| Total | 35.829 | 19.717 |

Table 1: Execution time of different steps in the pipeline.

## 4.1. Timing analysis of generating 3D city models

As shown in Table 1, the entire time cost for generating a 3D city model is 35 seconds on average, and it is dominated by the 3D model generation step. This quick generation process is crucial for applications of the virtual cities, as users can view the scenes immediately and make edits if necessary. As shown in Fig. 10a, most of the city generation time is under 60 seconds except for some outliers. These outliers are layout images that have a very complex road system or contain many more buildings than average.

## 4.2. Qualitative analysis of rendered scenes

As shown in Fig. 9, the output of our pipeline is a 3D object with the layout design specified by the input image and building blocks of proper density. All objects in the scene have corresponding textures applied. Through a closer look into the 3D model, as shown in Fig. 10b, the visual effects are realistic because all building textures are pre-processed and building dimensions have been proportionally scaled to match a consistent floor height.

## 4.3. Datasets for scene text detection training

To prove the quality of the rendered scenes using the output 3D city model, we collected a dataset for a scene text detection task. Scene text detection is a highly active research area of computer vision. Because of the rich information embedded in the texts in natural scenes, extracting them can help people better understand the surroundings of a scene. Currently, the state-of-the-art scene text detection and recognition algorithms are trained mostly with synthesized datasets such as SynthText [10]. There also exist other image datasets for this task collected from the real world [20].

We draw inspiration from this work [6] and trained a scene text detector with the dataset collected. The detection model used a backbone of FasterRCNN [18] and the convolutional layers are replaced with ShuffleNet[29] architecture.

To prepare the training data, we created multiple planes with textures of random English words. Then these planes are randomly placed on the walls of the buildings. A camera is set to move along the roads and we captured 100 images

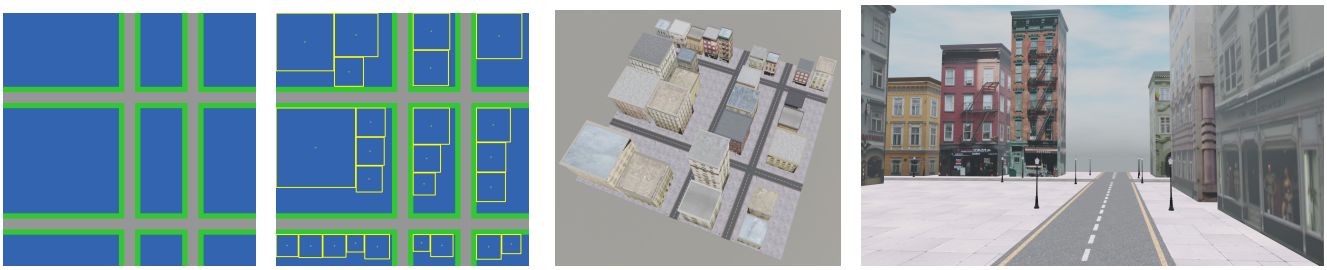
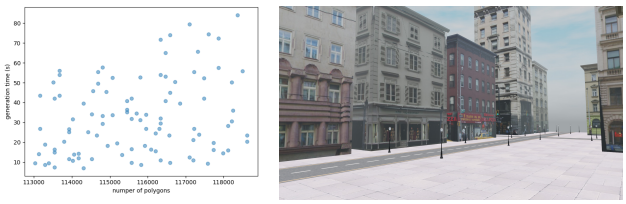


Figure 9: Visualization of outputs from different steps in our pipeline. From left to right: the input layout design, processed layout image with building positions, generated 3D city model, and zoomed-in view of the city scene.



(a) (b)

Figure 10: Left: scatter plot showing the execution time of 3D model construction as a function of model complexity (number of polygons in the scene). Right: rendered 3D city using Blender.

Figure 11: Left: synthetic data generated in SynthText dataset. Right: sample image in ICDAR15 dataset

from the rendered view. Coordinates of bounding boxes of inserted text planes are calculated automatically. We compared the model performance with training the scene text detector using SynthText [10] and ICDAR15 [13] datasets. ICDAR15 has served as a common benchmark for scene text detection algorithms [28, 24, 27]. To be more specific, the scene text detection model is trained separately on three different datasets (ICDAR15, SynthText, and rendered images from our virtual city), each with 100 images. 85 images are used for training and 15 images for validation. The model was trained for 30 epochs on each training set and the results are reported on the ICDAR test set using the model that achieved the best performance on the validation set during training.

As shown in Table 2, fine-tuning the object detection model with a dataset collected from our generated 3D city model achieved comparable performance to both synthetic and real-world fixed datasets. Using images generated by our 3D city model achieved a comparable F-score and the best precision rate among the three datasets. This demonstrates that our method is capable of generating valid training data for text detection tasks. The 3D environment provides the possibility of viewing the same texts from different angles and distances. Therefore the model can learn the characteristics of texts better with diverse conditions. Sample detection results from the model fine-tuned with our generated data are shown in Fig. 12.

| Training set | Precision | Recall | F-score |
|---|---|---|---|
| ICDAR15 | 34.05 | **68.86** | **44.08** |
| SyntheText | 37.95 | 42.28 | 40.01 |
| Ours | **46.59** | 34.25 | 39.48 |

Table 2: Scene text detection performance comparison among different training datasets.

Figure 12: Examples of scene text detection and recognition results. Left: a synthetic image rendered from our virtual city. Right: an image from ICDAR15 test set.

Using training data from ICDAR15 yields the best F-measure, as well as recall rate. This is not surprising because usually the training images and test images are collected under a similar condition. Therefore the model can easily adapt the knowledge learned during training to inference. However, the dataset collected from our generated 3D city model can be improved easily to adapt to a specific dataset like ICDAR15. By introducing more fonts, increasing text density, and adjusting the lighting conditions, an infinite amount of customized high-quality training data can be generated with our virtual 3D city at no cost.

### 4.4. Datasets for traffic sign detection training

We also collected a dataset for the traffic sign detection task. Traffic sign detection is an essential component in Advanced driver-assistance systems (ADAS) as driving operations instantly depend on the guidance of the message conveyed by the road signs. Failing to read road signs such as

Figure 13: Examples of scenes created for traffic sign detection. Left: rendered daytime city scene with traffic signs inserted. Right: rendered nighttime city scene with traffic signs inserted.

stop, speed limit, and pedestrian crossing signs will result in irredeemable losses. Therefore, it's important to accurately retrieve the road signs under different conditions. Specifically, the change of illumination between day and night raises serious challenges of developing ADAS with night vision[26]. In this section, we proved the feasibility of using our output 3D city model to create scenes with different daytime settings and provide training data for traffic sign detection at nighttime.

To prepare the training data, we inserted a random amount of traffic signs along the roads. A camera is set to move along the roads and we captured 120 images rendered from the city with synthetic daylight and 150 images rendered from the city with synthetic night sky. Coordinates of bounding boxes of inserted text planes are calculated automatically. As scenes shown in Fig 13, traffic signs with daylight are clear and easy to recognize. In the contrast, the signs become hard to detect during nighttime due to the low-illumination.

We chose the same object detection model used in the scene text detection task and trained it for traffic sign detection. The detection performance is compared between model trained with only the images under daytime setting and trained with images under both daytime and nighttime setting. For the daytime image dataset, it splits into 100 images for training and 20 images for validation. For the nighttime image dataset, it splits into 100 images for training, 20 images for validation, and 30 images for testing. The detector is firstly trained on images with daytime setting for 30 epochs. Detection results on nighttime testing images with the model achieved the best performance on the validation set during training are reported in Table 3. Then we performed transfer learning by fine-tuning the model trained on daytime images with the nighttime images. Detection performance is reported with the model yields the best performance during fine-tuning with the nighttime image validation set.

As shown in Table 3, fine-tuning the object detection model with nighttime images achieved significant improvement on both precision and recall rate. It shows that our rendered scenes are capable of providing valid training data with different light settings. As shown in Fig 14, after fine-tuning with nighttime images, the detector can retrieve traffic signs under extremely low illumination. This indicates the potential of using our pipeline as a training data source for harder cases, such as traffic sign detection with partial occlusion and adversarial attacks. An unlimited amount of training images can be generated using the proposed pipeline with little effort.

| Training set | Precision | Recall | F-score |
|---|---|---|---|
| Day imagery only | 40.42 | 32.75 | 36.19 |
| Day and Night imagery | **48.48** | **34.78** | **40.50** |

Table 3: Traffic sign detection performance comparison among different training datasets.

Figure 14: Prediction examples of traffic sign detector trained with different synthetic image dataset rendered from our output 3D city. Left: prediction result of the object detector trained with only daytime images and it failed to detect the road sign in a nighttime image. Right: prediction result of the object detector further fine-tuned with nighttime images and the road sign was successfully retrieved under extremely low illumination.

## 5. Discussion and Future Work

In this paper, we developed a pipeline that can take a 2D sketch as input and generate a 3D city model according to the information embedded. We also offer an algorithm to quickly generate a large number of city layout designs to choose from. Beyond layout design, the pipeline also provides other parameters to refine the output 3D models. The model can quickly scale up through flip operations. The textures are processed before being applied to buildings and all building dimensions are proportionally scaled to match a consistent floor height. The output virtual 3D city model can be directly modified and rendered using 3D development tools. Computer vision algorithms related to city scenes can be trained using rendered images generated by the virtual city. The scene text detection and traffic sign detection examples in this paper proved the effectiveness of 3D city models generated using our pipeline. In the future, we can leverage more complex techniques and incorporate accurate GIS information to refine the scene and include more details.

## References


[1] Nvidia drive sim. Accessed: 2020-09-01.

[2] Lukas Beer. Automatic generation of lod1 city models and building segmentation from single aerial orthographic images using conditional generative adversarial networks. *GI_Forum 2019,*, 7:119–133.

[3] Andrew Best, Sahil Narang, Lucas Pasqualin, Daniel Barber, and Dinesh Manocha. Autonovi-sim: Autonomous vehicle simulation platform with weather, sensing, and traffic control. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[4] Filip Biljecki, Hugo Ledoux, and Jantien Stoter. Generation of multi-LOD 3D city models in CityGML with the procedural modelling engine Random3Dcity. *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, pages 51–59, 2016.

[5] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2019.

[6] Fedor Borisyuk, Albert Gordo, and Viswanath Sivakumar. Rosetta: Large scale system for text detection and recognition in images. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 71–79, 2018.

[7] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[8] Kimberly Dylla, Bernard Frischer, Pascal Müller, Andreas Ulmer, and Simon Haegler. Rome reborn 2.0: A case study of virtual city reconstruction using procedural modeling techniques. *Computer Graphics World*, 16(6):62–66, 2008.

[9] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[10] Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. Synthetic data for text localisation in natural images. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[11] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[12] Ganesh Jha and Hubert Cecotti. Data augmentation for handwritten digit recognition using generative adversarial networks. *Multimedia Tools and Applications*, pages 1–14, 2020.

[13] Dimosthenis Karatzas, Lluis Gomez-Bigorda, Anguelos Nicolaou, Suman Ghosh, Andrew Bagdanov, Masakazu Iwamura, Jiri Matas, Lukas Neumann, Vijay Ramaseshan Chandrasekhar, Shijian Lu, et al. Icdar 2015 competition on robust reading. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 1156–1160. IEEE, 2015.

[14] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. In *Acm Transactions On Graphics (Tog)*, volume 25, pages 614–623. ACM, 2006.

[15] Gen Nishida, Adrien Bousseau, and Daniel G Aliaga. Procedural modeling of a building from a single image. In *Computer Graphics Forum*, volume 37, pages 415–429. Wiley Online Library, 2018.

[16] Yoav IH Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM, 2001.

[17] Przemyslaw Prusinkiewicz, Mark Hammel, Jim Hanan, and Radomír Měch. Visual models of plant development. In *Handbook of formal languages*, pages 535–597. Springer, 1997.

[18] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[19] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[20] Asif Shahab, Faisal Shafait, and Andreas Dengel. Icdar 2011 robust reading competition challenge 2: Reading text in scene images. In *2011 international conference on document analysis and recognition*, pages 1491–1496. IEEE, 2011.

[21] Jerry O Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun. Metropolis procedural modeling. *ACM Transactions on Graphics (TOG)*, 30(2):11, 2011.

[22] Radim Tyleček and Radim Šára. Spatial pattern templates for recognition of objects with regular structure. In *Proc. GCPR*, Saarbrucken, Germany, 2013.

[23] Eric van Rees. Esri cityengine 2013. *GeoInformatics*, 17(2):6, 2014.

[24] Wenhai Wang, Enze Xie, Xiang Li, Wenbo Hou, Tong Lu, Gang Yu, and Shuai Shao. Shape robust text detection with progressive scale expansion network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9336–9345, 2019.

[25] Curtis Wigington, Seth Stewart, Brian Davis, Bill Barrett, Brian Price, and Scott Cohen. Data augmentation for recognition of handwritten words and lines using a cnn-lstm network. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 639–645. IEEE, 2017.

[26] Yuxuan Xiao, Aiwen Jiang, Jihua Ye, and Ming-Wen Wang. Making of night vision: Object detection under low-illumination. *IEEE Access*, 8:123075–123086, 2020.

[27] Youjiang Xu, Jiaqi Duan, Zhanghui Kuang, Xiaoyu Yue, Hongbin Sun, Yue Guan, and Wayne Zhang. Geometry normalization networks for accurate scene text detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9137–9146, 2019.

[28] Fangneng Zhan, Chuhui Xue, and Shijian Lu. Ga-dan: Geometry-aware domain adaptation network for scene text

detection and recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9105–9115, 2019.

[29] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[30] Barret Zoph, Ekin D Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V Le. Learning data augmentation strategies for object detection. *arXiv preprint arXiv:1906.11172*, 2019.