Dalian University of Technology
Undergraduate Capstone Project (Thesis)

Design Generation Algorithm of Historic Building with Courtyard-style

Department: International School of Information Science & Engineering

Major: Digital media technology

Name: WANG Zihao

Student Number: 201894189

Supervisor: Prof. LIU Bin

Review Teacher: Associate Prof. XUE Xinwei

Completion Date: 2022-06-04

大连理工大学

Dalian University of Technology

原创性声明

本人郑重声明：本人所呈交的毕业设计（论文），是在指导老师的指导下独立进行研究所取得的成果。毕业设计（论文）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究成果做出重要贡献的个人和集体，均已在文中以明确方式标明。

本声明的法律责任由本人承担。

作者签名：王子浩　　日　期：2022/06/01

关于使用授权的声明

本人在指导老师指导下所完成的毕业设计（论文）及相关的资料（包括图纸、试验记录、原始数据、实物照片、图片、录音带、设计手稿等），知识产权归属大连理工大学。本人完全了解大连理工大学有关保存、使用毕业设计（论文）的规定，本人授权大连理工大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本毕业设计（论文）。如果发表相关成果，一定征得指导教师同意，且第一署名单位为大连理工大学。本人离校后使用毕业毕业设计（论文）或与该论文直接相关的学术论文或成果时，第一署名单位仍然为大连理工大学。

论文作者签名：王子浩　　日　期：2022/06/01

指导老师签名：刘[illegible]　　日　期：2022/06/01

摘要

建筑布局是一种综合性决策过程，由于设计师个体经验差异，导致建筑布局整体设计效率偏低、风格质量有待提高。将计算机设计、计算机思维有机地引入到建筑项目中，可对建筑设计起到巨大推动作用。本文以合院式风格的历史街区建筑布局方案自动生成为目标，探索基于 GAN 网络的图像转换算法以及基于 OpenCV 的自动设计算法，以期快速生成不同的合院式风格建筑布局方案，为设计师基本设计思路的形成提供参考。

首先开展了合院式建筑的肌理分析，进而分析了算法质量和效率要求，确定了算法的设计和美化两大功能和任务。

其次采用 GAN 实现算法的尝试和实践，构建了模型结构，制作了合院式建筑数据集，预测的图像在规定的地块中达到了自动生成的目标。虽然设计结果也能看出合院式建筑的围合性特征，但细节模糊，效果低于预期。

在以上尝试的基础上，考虑到合院式建筑在肌理图上有更多的图形学性质和规律，采用 OpenCV 来实现自动生成算法。该算法以图形学和几何学的视角，通过对合院式建筑的设计规则的研究，从地块的识别、地块切割和分区、建筑细节的修饰多个层面进行算法研究设计，并通过 Python 程序和 OpenCV 库编写实现了完整的合院式建筑布局设计生成系统。

算法程序运行结果表明，针对不同形状和面积地块，可以生成大量建筑布局样本，为设计师提供了更多选择方案，也可以辅助激发其设计思路和灵感。

关键词：合院式建筑；生成设计；算法；OpenCV

Design Generation Algorithm of Historic Building with Courtyard-style

Abstract

The architectural layout is a comprehensive decision-making process. Due to individual designer experience differences, the overall design of the architectural layout is inefficient, and style quality needs to be improved. The organic introduction of computer design into architectural projects can help impetus greatly to architectural design. In this thesis, we aim at the automatic generation of architectural layout schemes for historic districts in Courtyard-style and explore the image conversion algorithm based on Generative Adversarial Network and the automatic design algorithm based on OpenCV in order to quickly generate different architectural layout schemes in Courtyard-style and provide a reference for the formation of designers' basic design ideas.

Firstly, the architectural texture analysis of the Courtyard-style buildings is carried out. Then the quality and efficiency requirements of the algorithm are analyzed, and the two main functions and tasks of the algorithm, design, and beautification, are determined.

Secondly, the algorithm was tried and practiced using GAN implementation. The model structure was constructed, a dataset of Courtyard-style buildings was produced, and the predicted images achieved the goal of automatic generation in the specified land parcels. Although the design results can also show the enclosing characteristics of the ensemble building, the details are blurred, and the results are lower than expected.

OpenCV was used to implement the automatic generation algorithm based on the above attempts, considering that the Courtyard-style buildings have more graphical features and patterns in the architectural texture map. The algorithm is designed by studying the design rules of Courtyard-style buildings from the perspective of graphics and geometry. The algorithm is studied at several levels, from recognizing land parcels, parcel cutting, post-processing, and modification of architectural details. A complete layout design generation system for Courtyard-style buildings is implemented by writing a Python program and OpenCV library.

The results of the algorithm program run show that a large number of building layout samples can be generated for different shapes and sizes of parcels, which provides designers with more options and can also assist in stimulating their design ideas and inspirations.

Key Words：Courtyard-style Building；Generating Designs；Algorithms；OpenCV

囲まれたスタイルの歴史的建造物の設計の生成アルゴリズム

概要

建築レイアウトは総合的な意思決定プロセスであり、設計者の個々の経験により、全体の設計効率が悪く、スタイルの質を向上させる必要がある。 建築プロジェクトにコンピュータデザイン、コンピュータシンキングを有機的に導入することで、建築設計に大きな弾みをつけることができるのである。 本論文では、中庭形式の歴史的地区における建築配置図の自動生成を目指し、GAN ネットワークを用いた画像変換アルゴリズムと OpenCV を用いた自動設計アルゴリズムの検討を行い、異なる中庭形式の建築配置図を迅速に生成し、設計者が基本設計案を形成するための参考とすることを目的としている。

まず、複合式建築物の質感の分析を行い、アルゴリズムに要求される品質と効率の分析を行い、アルゴリズムの 2 大機能とタスク、設計と美化を明らかにした。

次に、GAN 実装を用いてアルゴリズムを試行し、モデル構造を構築し、複合様式の建物のデータセットを作成し、予測した画像は定義されたプロットで自動生成の目標を達成した。 設計の結果、アンサンブル建築の囲い込み特性も明らかになったが、細部がぼやけてしまい、期待値より低い結果となってした。

以上の試みに基づき、ホープウェル様式建築物のよりグラフィカルな性質と規則性をテクスチャマップの観点から考慮し、OpenCV を用いて自動生成アルゴリズムを実装した。 グラフィカルで幾何学的な観点からアルゴリズムを設計し、区画の特定から区画の切断やゾーニング、建築の細部の修正まで、複式建築の設計ルールをいくつかのレベルで検討し、複式建築のレイアウト設計生成システムを Python プログラムと OpenCV ライブラリで完成させ、実装した。

アルゴリズムプログラムの結果、さまざまな形状や大きさの敷地に対して、多数の建物配置サンプルを生成することができ、設計者に多くの選択肢を提供し、設計のアイデアやインスピレーションを刺激する一助となっている。

キーワード：中庭スタイルの建物、デザインの生成、アルゴリズム、OpenCV

Catalogue

1 Introduction

1.1 Research background

Architecture, one of the oldest disciplines, faces the challenges of a new era. On the one hand, the external material world that architecture has to deal with is changing rapidly. On the other hand, its old and traditional approach to research has given rise to many problems:

- The expression of ideas is too time-consuming.
- Design ideas lack a quantitative basis.
- The cost of trial and error is too high.
- There are significant gaps in experience between individual designers and overall inefficiency.

As large-scale urban elements, buildings are the external features and surface culture of the city. The building layout is an essential element of the overall morphological architecture of the city and is an essential component of urban design. At the same time, building layout is also a comprehensive decision-making process related to almost all activities in the construction and development of the city, such as land development and functional structure, and is an essential factor in determining the overall functional positioning and development direction of the city, and is an essential guide to the operation, construction and management of the city itself.

As an emerging discipline in recent decades, Artificial intelligence and computer technology have set in motion a worldwide transformation of information technology in a wide range of industries and disciplines. Combined with the current extensive market environment for computers, architecture also faces the same need for informational transformation. The organic introduction of computer design and computer thinking into architectural projects has been a suitable catalyst for architectural design, helping to transform and innovate the tools in architectural design. It has become the most critical issue in the interdisciplinary exchange.

Computer and artificial intelligence technologies have been used more often in architectural design. For example, computers are used to calculate and simulate environmental conditions around a building or a group of buildings [1]; 3D reconstruction techniques are used to create 3D models of city maps [2]; 3D digital techniques are used to assist urban design work [3] and even planning decision making [4], etc. In the field of aided architectural design, the application of computers is still in its infancy, although it has gained wide attention as a popular research object in recent years.

Computer-aided design, i.e., in architecture, uses computer technology to replace or help architects in architectural design and planning [5]. Due to a large number of unquantifiable

artistic and creative components in architectural design, computers have not progressed as rapidly as other technologies in this field. At present, the research on generative design at home and abroad is still in its infancy, with few available references of varying quality, which has caused some difficulties for domestic research in this area in architecture [6]. For the study of generative design in architecture, expertise in architectural design is required and a combination of expertise in computer programming, computer graphics, mathematics, and other disciplines, which also adds difficulty to the collection and study of data. However, the breakthrough in machine learning technology has made it possible to use deep learning methods to realize ideas and methods for diversifying architectural solutions [7].

1.2 Related work

There are precedents for applying current computer-aided design techniques, both for the internal layout and the external arrangement of buildings.

For the internal layout of buildings, considering that its design rules are relatively obvious and easy to quantify, there have been many application cases, such as the algorithm based on the authors' architectural observations to generate residential units with realistic floor plans [8], or the exploration based on multi-intelligent body system and integer planning algorithm by Southeast University [9]. Here, we focus on the work of Paul Merrell, Eric Schkufza, and Vladlen Koltun at Stanford University [10], who devised a method to automatically generate buildings with internal structures for computer graphics applications. The external appearance of the building is then generated from the internal layout they have designed and can be customized with various decorative styles. They pay special attention to the generation of residences, which is common in computer games and online virtual worlds. Compared to the highly regular layouts often encountered in schools, hospitals, and office buildings, the layouts of residential buildings are not as prescriptive. As a result, the design of building interior layouts is particularly challenging because the goals are less precisely defined and more difficult to manipulate. The design of residential layouts is often an iterative, trial-and-error process that requires expertise. Their approach to computer-generated building layouts is motivated by the approach to building layout design often encountered in real-world architectural practice. The input is a concise list of high-level requirements, such as the number of bedrooms, the number of bathrooms, etc. These requirements are expanded into a complete building program containing a list of rooms, adjacencies, and required dimensions as shown in Figure 1.1(a). This building program is generated by a Bayesian network trained on real-world data. A set of floor plans for implementing the building program is then obtained by stochastic optimization as in Figure 1.1(b). These floor plans can be used to construct a 3D

model of the building. The method provides a complete pipeline of computer-generated building layouts. And because the method is designed for computer graphics applications, the focus is placed on visual similarity to real-world residential layouts.

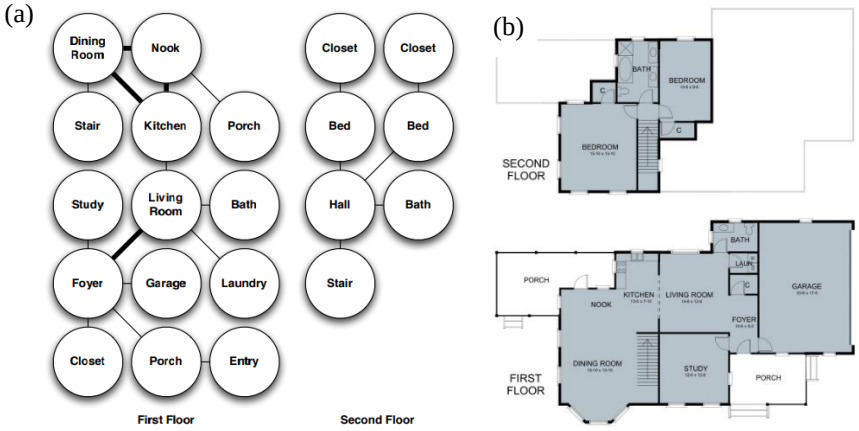

Figure 1.1 Computer-generated building layouts: (a) list of high-level requirements[10], (b) a set of floor plans [10]

Similar to the approach above is a method based on graph transformations by The University of Texas at Dallas [11]. They introduced a method to automatically generate rectangular floor plans based on existing conventional floor plans with the ability to improve further and customize them. First, a biplot is derived from a given input file of the specified floor plan. Then, it either automatically copies different floor plans, preserving the connectivity of the original floor plan, or executes transformation rules to handle the spatial relationships between rooms and generates a modified floor plan that meets specific requirements. Constraints, such as maximum aspect ratios, are then introduced to support the flexibility of various design requirements. And a graphical user interface is provided for users to perform the automatic generation process.

In addition to this, for interior layouts, the School of Architecture at Texas A&M University proposed a physics-based approach to assist design [12]. They create a method for the responsive design process by applying physics-based techniques to architectural space planning. Although responsive design can be applied to many areas of graphic design, space planning is a relatively simple and well-researched area to begin to understand and apply the complexity of this approach. Their research focuses on modeling design goals as objects that apply forces in a physically-based spatial planning system, as shown in Figure 1.2. In their spatial planning approach, architects define program goals in the usual way. These goals are then modeled as physical objects and forces used in dynamic physical simulations. They modeled spaces and walls as point masses, and the adjacencies between spaces were modeled as springs connecting the masses. Targets specified in the building program are translated into

forces applied to the masses. The dynamic simulation proceeds by allowing the mass-spring system to reach equilibrium rapidly. The designer then modifies and adds targets by manipulating the graphical model directly, rather than reassigning design targets in the language of the underlying system. The mass-spring representation allows the graphical model to adapt to these changes immediately. We do not intend to simulate the actual behavior of the architectural elements but rather the way the architect views and interacts with the design elements during the conceptualization process.

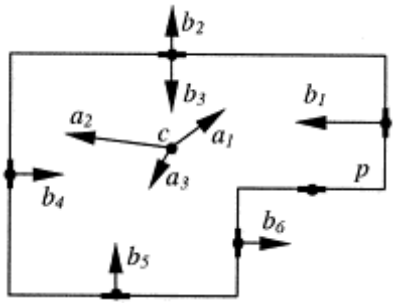

Figure 1.2 The forces used in dynamic physical simulations [12]

The external layout of buildings has also been addressed in various ways, both domestically and internationally. At a relatively early stage, many studies focused on the 'auto-growth' approach to higher volume building layout design, such as the study by Yuchao Sun [13], and the application of cellular automata in building design [14]. With Computer-Generated Plans in a Greenfield Area [13] and cellular automata in building design [14], among others. Furthermore, deep learning started to have new attempts in the field in these years.

Taking Xiaoku xkool as an example [15], its model consists of two main parts, a generator that thinks about how to generate designs, i.e., the model obtained by training against a large amount of data outputs possible high-quality solutions. An evaluator that assesses the value of the design samples, i.e., the generated solutions are evaluated and weighted in different dimensions, considering the base conditions and the surrounding environment to determine the final choice. In the generator, there are three different neural networks used to generate building layout solutions, namely.

a fast generation network for the base design logic, i.e., a neural network based on architectural design logic;

a CNN network, which learns a large number of mature design solutions; and

a GAN network, which generates more solutions based on the CNN network to compare and play against each other to generate solution results.

In the evaluator, the value judgment network calculates the results with the weighted average of Monte Carlo simulation results. The search is repeated continuously through four

steps: selection, expansion, simulation, and backpropagation, and set to stop the search simulation at a reasonable time, with the ten neural network nodes with the most number of search passes returned as the best solution as ten solution results, as shown in Figure 1.3.

Although many proven schemes are used as data for training in training CNN network models, the trained neural networks are not optimal. Therefore, it is necessary to train the generative models using GAN networks among the existing models. Their results are superimposed in the high-dimensional space to find the overlapping part as the optimal solution. By continuously refining the training models and enhancing the targeting of different

Figure 1.3 Solution results returned by Xiaoku xkool [15]

weights, the two models, generation and evaluation, can be sliced into models with a lower level of granularity. Through the accumulation of urban data and adversarial generation, about 80% of the final generated solutions are good quality solutions after many training sessions.

In Xiaoku xkool's work, the floor area ratio or daylight conditions are mostly used as the optimization target. The existing optimization tools are used for layout plan generation. However, the economics and layout structure influence the builder's investment decision, which is a critical factor for customer purchase intention and satisfaction, thus affecting the final presentation of the settlement building layout. The School of Management Science and Real Estate of Chongqing University has also conducted a deeper study on the automatic generation methods of building layouts in urban settlements [16]. The study combines manual layout experience and computer automation to achieve the automatic layout of residential buildings by BL algorithm [17] and genetic algorithm with the maximum building volume ratio as the optimization objective.

Moreover, automatic layout generation experiments are conducted for typical residential types based on the constraints of daylight conditions. The study takes economic efficiency and layout rationality as the optimization objectives to construct the residential building layout model based on satisfying the volume ratio, building density, and daylighting requirements and realizes the automatic layout of residential areas by simulated annealing algorithm. And

according to the real estate product positioning theory, the optimal scheme design is determined to provide decision-makers and designers with referenceable settlement layout schemes and provide new solution ideas for building problems that are difficult to calculate quantitatively.

All the above studies are conducted for the layout design of residential buildings. For stylized buildings, such as traditional Chinese gardens, Huizhou-style buildings, Courtyard-style buildings, etc., which contain apparent and unique features and structural information, developers often need to extract regular features from them and develop algorithms using traditional methods so that they can automatically generate a variety of architectural designs for architects' reference. For example, in the study of Southeast University, pattern-based architectural space generation will be used to extract, construct and apply patterns from multiple aspects such as the overall layout, road system, architectural monoliths, stylistic elements, and ornamental patterns of traditional Huizhou residential villages [18]. It is hoped that by thinking about the characterization of pattern language in architecture, further design generation for the corresponding problems in cities and architecture will be accomplished through practice. In order to show the systematic transformation of the patterns and the construction process of the methodological system, other elements such as environmental and economic elements are not introduced into the core of the program design but are added to the algorithm design as influencing parameters of the relevant functions. In general, the research of algorithms for generative design is oriented toward "pattern languages" [19]. At the level of algorithm exploration, it is mainly limited to two aspects: intelligent optimization algorithms and process modeling algorithm design. In contrast, other algorithms are only discussed in terms of development possibilities.

Focusing on the monolithic generation problem of Huizhou houses, the generation of buildings is divided into three sub-modules: plan layout generation, structure generation, and façade generation according to the patterned characteristics of the spatial composition and façade style of the houses. Based on plan layout determination, for the generation process of structure and façade, similar cases are compared. The data laws between the constituent elements are obtained through the study of a large number of surveying and mapping data. Then, the relevant numerical relationships are classified and discussed to obtain the finalized physical size according to relevant actual data application. In the generation stage of the model, a large number of repeated components are organized and built into a component library. The required components are obtained to be displayed in the model through parameter control, which not only dramatically improves the efficiency of the program but also provides convenience for the subsequent addition of model details, and only requires the content expansion of the components in the component library to obtain a more accurate model. In

generating from plan layout to structure and elevation, the data are continuously output in this system, and at the same time input as conditions to the following link, and combined with the image information model to differentiate and extract the image information to realize the result generation. Finally, the automatic design and generation of Huizhou folk house architectural clusters are realized, as shown in Figure 1.4.

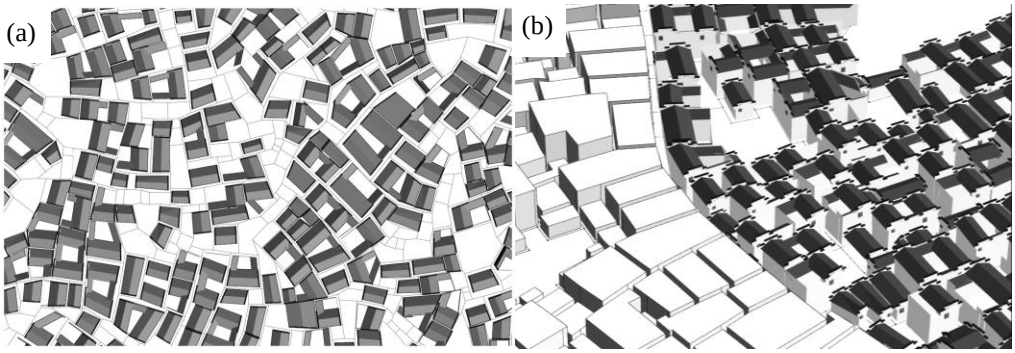

Figure 1.4 Automatic design and generation of Huizhou architecture: (a) architectural layout[18], (b) 3d visualization results [18]

1.3 Research target

This design will develop a new algorithm around a courtyard style historic district building, using deep learning and technology such as graphics processing to learn and imitate the morphological features of stylized buildings in the historic district, find the features of the building form and summaries the rules when designing the building. It also uses the computer to quickly generate different solutions to achieve a computer-aided design of the stylised building. The focus is on outputting a variety of reasonable and diverse design solutions for any input site information, providing designers with a reference for basic design ideas, helping to open up design ideas and injecting new energy into current architectural design work. It also makes visual adjustments to the design solutions, generating architectural texture drawings or 3D schematics that align with the architect's habits and improve their referential nature.

The research focus of this thesis is as following:

(1) procedural translation of architectural design rules: design rules in architecture are mostly expressed as some vague constraints and aesthetic bases. If the rules in architectural design language are to be introduced into the program, it is necessary to organize the specific data effectively reasonably. This transformation needs to ensure the effective input of information on the one hand, and the data characteristics expressed by the information need to be preserved on the other hand.

(2) Algorithm design: Algorithm design is the program's core. The design process requires deductive reasoning in algorithms, thus realizing the projection from the design space

to the program space. The process of algorithm design for the transformation of the actual problem usually requires the introduction of more ways and means to deal with redundant information and edge conditions.

(3) Reasonableness of results: Generative design reflects its advantages by the efficiency of generating results. However, the ability to verify the reasonableness of results needs to be strengthened. How to fully consider the influencing factors of each design aspect and organize each parameter system in a reasonable modeling way when generating the design is the key to affecting the result rationality. In addition, the introduction of the result validation mechanism is another guarantee of the result rationality and a perfect complement to the algorithm design.

2. Requirements analysis and selection of automatic generation algorithm

2.1 Analysis of architectural morphological characteristics

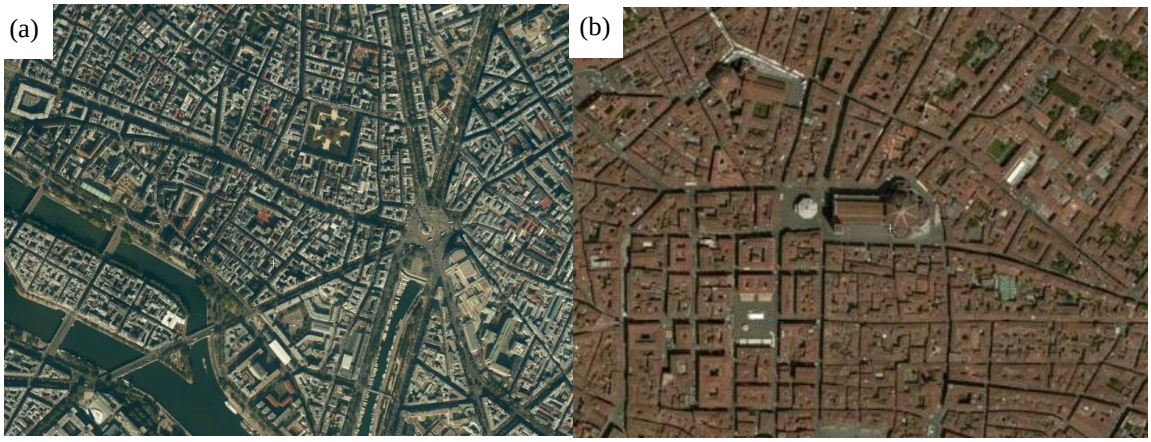

Figure 2.1 Courtyard-style buildings in the world: (a) in Paris, (b) in Florence [4]

The architectural texture is a very important component in artistic design. In the art of composition, any form of compositional meta-table is visually expressed through specific or abstracted points, lines, and surfaces. The combination of graphics and color is then used to form idealized, textured compositional patterns based on beautiful visual effects. In daily life, any material seen everywhere is made up of different materials and materials, and the unique property of these materials and materials is texture. Therefore, when the texture is used in graphic design, it can be expressed in different shapes, colors, and materials and shown to people through specific artistic treatments; the application of texture in interior design can be expressed as the artistic texture expression of interior walls, floors, ceilings, and other decorative materials; texture applied to architecture can be expressed as the facade and internal structure of the building, and the unique texture of the materials it needs to expression [20]. More specifically, the architectural texture combines spatial relationships between architectural objects, including composition, form, and scale. Moreover, different combinations of them can present different texture organizations. The architectural texture is the basic unit of the city in spatial structure and the essential element of urban space [21].

The courtyard-style architecture has a long history in China, and the Beijing courtyard, Wannan courtyard, and Jinzhong merchant's house are all representative buildings [22]. In foreign countries, Courtyard-style architecture is more common, such as Paris, France, and Florence, Italy, famous for their courtyard-style architecture throughout the city, as shown in Figure 2.1. In the case of Paris, France, for example, the traditional French fort often used the enclosure style, and the builders inherited this form to build the city, considering the defensive

function of the city. In addition, Paris had a unified urban plan and was transformed by the Haussmann barons in a relatively short period. Most of the buildings were built in the Haussmann style, with neat and uniform buildings and street spaces, allowing this form to be reproduced continuously and to constitute the urban space [23].

This thesis analyzes the architectural texture of the courtyard-style buildings in Dongguan Street, Dalian, as an example.

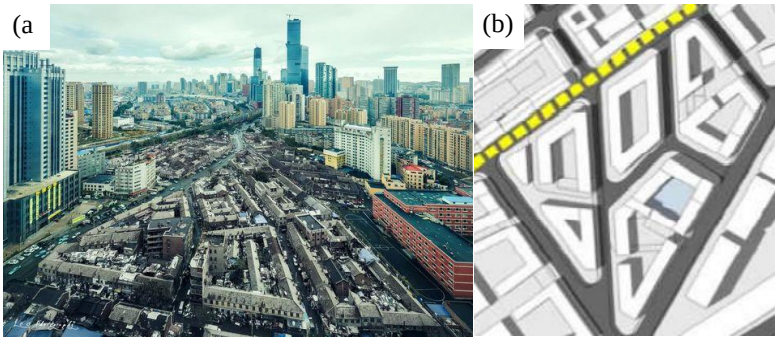

Figure 2.2 Streetscape and architectural texture of Courtyard-style buildings: (a) Dalian Dongguan Street, (b) architectural texture [25]

The traditional architecture of the courtyard style in the old city of Dalian Dongguan Street is a mixture of Chinese courtyard style and Western classical decorative elements, mostly of a "central hall and external corridor" structure. In short, that is, the corridor between four or three sides of the houses connected along the street to construct a layer of fully or semi-enclosed enclosed architectural structure. The shape of the courtyard is mainly irregular

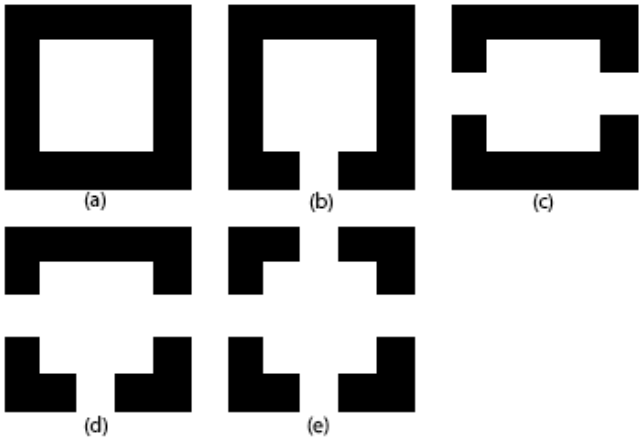

Figure 2.3 Courtyard-style building opening layout: (a) totally closed, (b) opening in one direction, (c) opening in two directions, (d) opening in three directions, (e) opening in four directions

rectangular, triangular, trapezoidal, octagonal, and wedge-shaped [24] (as shown in Figure 2.2). The central hall surrounded by the outer buildings is similar to the patio of southern

residential houses and is generally larger. This highly enclosed building form is more likely to create a sense of belonging, security, and domain than a loose building layout or, for example, the rows and columns of modern residential areas. Hopscotch buildings are not necessarily fully enclosed structures; most buildings will have openings in one or more directions. As shown in Figure 2.3, the number, direction, and size of the openings for each building are not the same [26]. Most of the Courtyard-style buildings on Dong Guan Street in Dalian are distributed in clusters along the main street. There will often be multiple Courtyard-style buildings built adjacent to each other on a large convex quadrilateral or triangular-shaped land parcel with a large convex quadrilateral or triangular a regular and proportional distribution and similar footprint [27].

2.2 Algorithm quality and efficiency requirements

Based on the texture analysis of the courtyard buildings, the algorithm needs to automatically design the distribution location, footprint, and morphological features of each courtyard building on a single or multiple, triangular or arbitrary convex quadrilateral parcel. The morphological features include the external shape of each courtyard building, the shape of the internal nave, and the location, number, and orientation of the openings in the nave. After completing the preliminary design, 3D visualization or other means need to be used to improve the readability and aesthetics of the design drawings.

The design drawings should not take too long to generate and should not take more than one minute at most. Although the architects have certain habits and preferences in carrying out the design, the similarity between multiple design drawings generated by the algorithm should still not be too high. Otherwise, the reference ability will be significantly reduced. Therefore, in terms of operational efficiency, the algorithm needs to generate at least 20 design drawings that are not excessively similar within one hour to achieve a sufficiently significant improvement compared to the average efficiency of current architectural designs [28].

2.3 Implement method choice analysis

In a nutshell, the algorithm needs to perform two parts of the task: design and beautification.

For the design part of the task, the algorithm's goal is to generate a design map similar to the urban texture map. The author can think of two possible ways to achieve this for the time being.

(1) It is necessary to write functions that can reflect the laws of the building structure based on the universal laws of the Courtyard-style building structure and combined with the research on the street morphology [29] and street model [30], using computer vision and image processing libraries such as OpenCV, and using traditional image processing. The input and output interfaces are then designed to be processed and designed for any input of parcel information.

(2) Design dataset and model using machine learning and deep learning techniques. The model is then trained by inputting many texture maps of real-world Courtyard-style buildings and their related constraints. The model can automatically generate building texture maps under a given constrained environment (e.g., a land parcel of a specific size and shape is delineated).

For this part of the beautification task, the algorithm's goal is to improve the readability and aesthetics of the design results, so the goal can be achieved by generating predicted aerial photographs or generating 3D schematics. Three approaches come to the author's mind for the moment as possible implementations.

(1) Using machine learning and deep learning, input many texture maps of real-world Courtyard-style buildings and their corresponding aerial photographs to train the model. The model can automatically generate predicted aerial photographs based on the texture maps already generated by the input algorithm.

(2) Using the building texture map already generated by the algorithm, adding the building height information to generate a point cloud data structure, and then using a 3D visualization library to generate a 3D model.

(3) Use computer vision and image processing libraries such as OpenCV to process further the building texture map generated by the algorithm to obtain a floor plan with 3D visual effects (pseudo-3D image).

3. GAN-based algorithm for generating building layout plans

3.1 Image conversion from architectural texture map to aerial photograph

3.1.1 The basic idea of architectural texture map to aerial photograph

The function is mainly beautifying. The idea was mainly inspired after reading a paper presented at CVPR 2017 by the University of Californian [6].

The algorithm's goal under this idea is to obtain predicted aerial images from architectural texture maps. In other words, it is also the problem of converting one type of image to another, i.e., the Image-to-Image Translation Problem. In this thesis, the authors define the image-to-image conversion problem as converting one possible representation of a scene into another, given enough training data. One reason for the difficulty of language translation mentioned in the paper is that mappings between languages are rarely one-to-one, and concepts in one language are always easier to express than in other languages. Similarly, most image translation problems are also many-to-one (computer vision) or one-to-many (computer graphics). In general, each task is solved using a separate, dedicated mechanism. Nevertheless, in reality, these tasks are essentially the same: predicting new pixel points from pixel points. Thus the problem is further deconstructed into the problem of predicting pixels from pixels or mapping pixels to pixels, and the authors propose Conditional Generative Adversarial Networks (CGAN) to solve it.

Figure 3.1 Map to aerial photo[6]

As shown in Figure 3.1, CGAN enables the conversion from architectural texture maps (or maps) to aerial photographs. In fact, since the task of image conversion is essentially the same, the same or similar results can be achieved by simply replacing the dataset, even if the same network structure and objective function are used. As shown in Figure 3.2, the network can achieve a series of conversions from labeled Labels to real pictures, from grayscale to color pictures, from day to night, from edge to real pictures, and so on., after replacing the dataset. This uses the Conditional Generative Adversarial Network CGAN as a general

solution to the image transformation problem. The network not only learns how to complete the mapping from the input image to the output image but is also able to learn the Loss function used to train the mapping relationship. Therefore, the author no longer needs to design the Loss function or mapping function manually and only needs to change the data set to see the initial results.

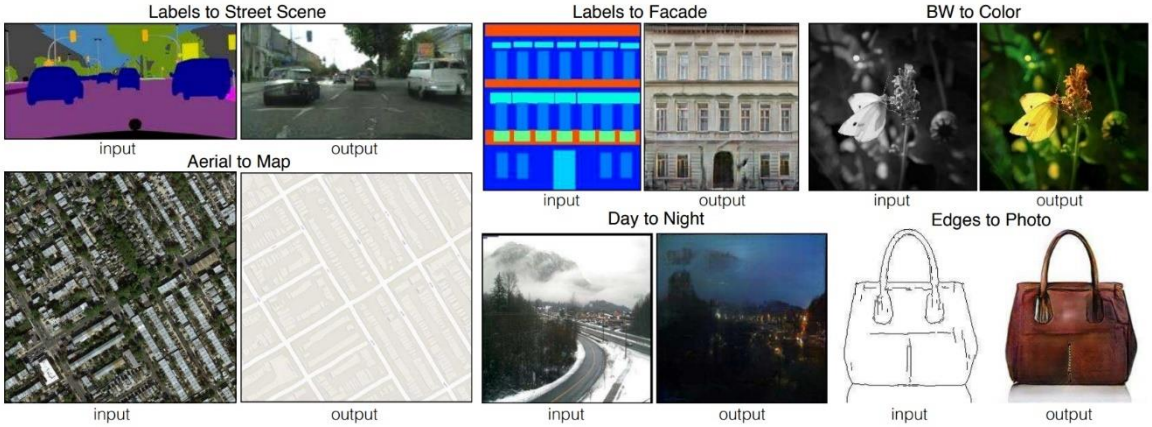

Figure 3.2 Image transformation using different datasets [6]

3.1.2 Collection and manufacture of datasets

Before creating a dataset, it is essential to explain the concepts of Image Content and Image Domain. Image content refers to the inherent content of an image and is the basis for distinguishing different images; image id refers to the common attributes that some images are given. Image transformation is the conversion of one feature of an object to another feature of the same object, such as obtaining a color image of a handbag based on the outline of the handbag. In a nutshell, image transformation is finding a function that will allow the image of domain A to complete the mapping to domain B, thus enabling the transformation.

Therefore, when making the dataset, two corresponding datasets need to be made. The images of set A and set B correspond one to another, and the two corresponding images need to reflect different features of the same object. The current goal is to convert architectural texture maps into aerial photography images, then data sets with pairs of architectural texture maps and aerial photography images are needed.

The first batch of satellite aerial photographs of similar courtyard structures was selected and collected mainly in Qingdao, Tianjin, and Dalian in cooperation with the School of Architecture and Art. After that, the second batch of satellite aerial images was selected and collected in Berlin and Paris. Furthermore, the architectural texture map was obtained using BIGMAP. The size is 256×256 pixels, corresponding to the actual map size of 200 m×200 m. Fifty pairs of images were collected, as shown in Figure 3.3.

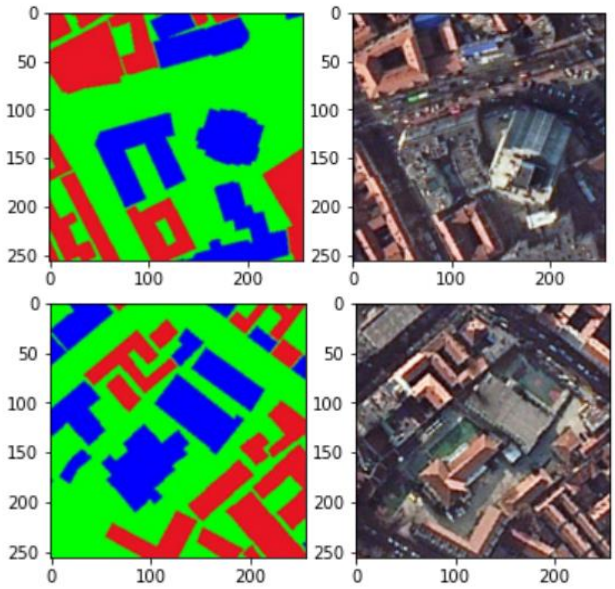

Figure 3.3 Dataset of labeled architectural texture maps and aerial images

After ensuring that the satellite aerial images corresponded to the building texture map numbers one by one, each building texture map was manually colored using Adobe Photoshop in this thesis. After comparing the satellite aerial images, the roads or open spaces with non-building parts are filled with green; for Courtyard-style buildings, with red; for non-condominium buildings, with blue. The purpose is to distinguish courtyard-style buildings from non-courtyard-style buildings in the results for easy observation and analysis.

Due to the small number of Courtyard-style buildings, collecting data can only be done manually, requiring students to use their naked eyes to find Courtyard-style buildings on the satellite map, making it difficult to collect more data in a short period. In order to expand the dataset, the existing dataset needs to be data enhanced. As part of geographic information, both architectural texture maps and satellite aerial photographs possess directional invariance. In order to expand the dataset, slices are intercepted every 10° of rotation for the same geographic target, thus expanding the original 50 pairs of images to 1800 pairs of images. In addition, all datasets are randomly transformed with left-right symmetry before being input to the model for training.

3.1.3 Pix2pixGAN model structure

The generator uses a U-Net network structure. First, eight consecutive downsampling operations are done, and the downsampling model is one layer of convolutional layer, one layer of BatchNormalization, and one layer of LeackyReLU. 128×128×64, 64×64×128, 32×32×256, 16×16×512, 8×8×512, 4×4×512, 2×2×512, 1×1×512 images, and save the output result of each time except the last data.

Following that, seven successive upsampling are done. The upsampling model is one layer of deconvolution, one layer of BatchNormalization, one layer of Dropout (with a probability of 0.5), and one layer of LeackyReLU. 2×2×512, 4×4×512, 8×8×512, 16×16×512, 32×32×256, 64×64×128, 128×128×64 images and concatenate them with the corresponding images saved before, respectively, as skip-connections in U-Net networks for fusing features at different scales.

The discriminator uses the structure of patchGAN, which divides the input image into multiple N×N patches, and then performs the discriminator's truth determination on each patch. In this way, the number of parameters can be significantly reduced, and the speed of operation can be significant.

In addition to the loss function (3.1) included in the GAN, the L1-distance (3.2) is added to the loss function so that the goal of the generator is not only to pass the discriminator judgment but also to ensure that the output image is close to the ground-truth to a certain extent, so the final optimization objective function (3.3) is obtained.

$$\mathcal{L}_{\mathrm{cGAN}}(G,D) = \mathbb{E}_{x,y}[\log\ D(x,y)] + \mathbb{E}_{x,z}[\log\,(1 - D(x,G(x,z)))] \tag{3.1}$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\parallel y - G(x,z)\ \parallel_1] \tag{3.2}$$

$$G^* = \arg\ \min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G) \tag{3.3}$$

The discriminator uses the structure of patchGAN, which simply divides the input image into multiple N×N patches, and then performs the discriminator's truth determination on each patch. In this way, the number of parameters can be significantly reduced, the speed can be increased, and the image input of any size can be processed. Both the generator optimizer and the discriminator optimizer use Adam, with a learning rate of 0.0002.

Otherwise, all model parameters are consistent with those in the paper *Image-to-Image Translation with Conditional Adversarial Networks*.

3.1.4 Experiment and analysis

Development environment:

Operating System: Windows 10 Home 21H2
Processor: Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz
Installed RAM: 32.0 GB
GPU: NVIDIA Geforce RTX 3080 × 1
IDE: Jupyter Notebook 6.3.0
Development language: Python 3.8.8
Deep learning framework: TensorFlow 2.6.0

The results, shown in Figures 3.4, 3.5, and 3.6, are not very satisfactory. There are more noise points in the satellite aerial images predicted by the model. Some of the buildings have basic outlines, but some are broken and distorted, and some of the buildings do not show standard outlines. All in all, the satellite aerial photographs predicted by the model cannot enhance the objectivity of the building texture map.

Figure 3.4 Result of epoch:100 (a) input, (b) ground truth, (c) predicted image

Figure 3.6 Result of epoch:250 (a) input, (b) ground truth, (c) predicted image

Figure 3.5 Result of epoch:200 (a) input, (b) ground truth, (c) predicted image

After discussing with the subject group of the School of Architecture, we concluded the following principal reasons.

(1) The building texture map and the satellite aerial photograph do not fit perfectly. For buildings with high heights, such as Figure 3.7, there is a huge difference in the pixel level between the building texture map and the depiction of the building in the satellite aerial photograph. The height of this building shifts the position of its roof and foundation in the image because the satellite shot is not perpendicular to the building above it. This phenomenon

will cause a massive problem for the proper operation of the patchGAN-based discriminator, and this problem is also the main problem in the dataset.

(2) In the original satellite aerial images, the building shadows are not facing the same direction. Due to the difference in shooting time and the later data enhancement, the shadows of the buildings may face in any direction. Moreover, the shadows produce a considerable change to the color of the building roof, and the model does not adapt well to this change.

(3) In the original satellite aerial photograph, there are many vehicles on the road and many objects distributed on the roof. Many objects containing fine edges spread throughout the satellite aerial photograph severely distract the model from including the simulation of these noise-like objects in the prediction map.

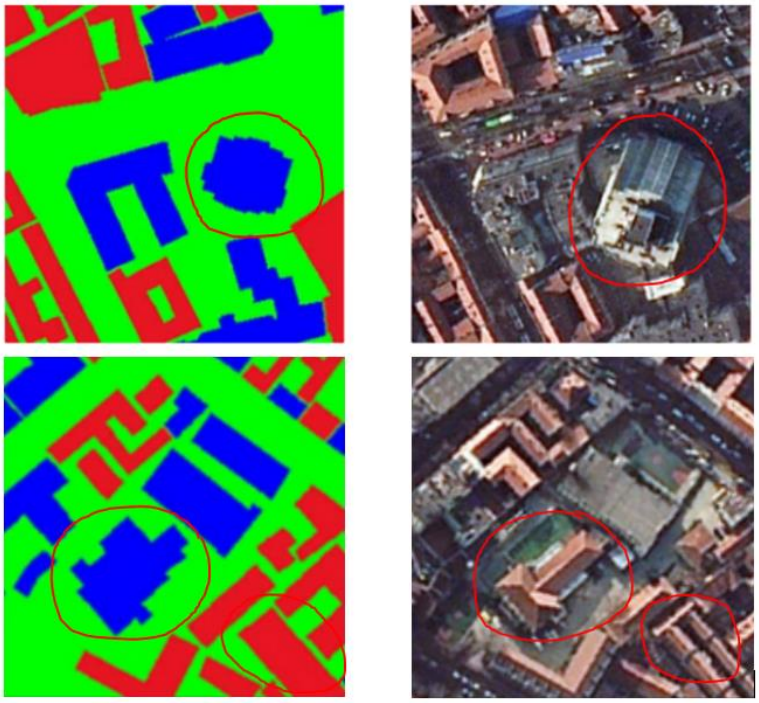

Figure 3.7 Image inconsistencies in datasets

Many problems exist in the original dataset, but it is not easy to correct or improve. For the problem of "the building texture map and the satellite aerial photograph do not fit perfectly." However, it can be corrected manually using the software one by one; the time cost is enormous. For the problem of "the orientation of the building, shadows are not the same," there is no strategy to solve it for the time being. For the problem of "satellite aerial photos contain noise and irrelevant objects," if we use Gaussian blur to pre-process the data set, it will make the final prediction map challenging to produce sharp edges and reduce its reliability.

3.2 Image conversion of land parcel to building texture map

3.2.1 The basic idea of land parcel to building texture map

In the previous attempts on GAN, the desired results were not obtained due to many problems with the dataset. Therefore, in the next attempt, the requirement for the dataset was reduced, and an attempt was made to have the GAN network automatically generate the building texture map.

The automatic generation of building texture maps is the core function of the whole algorithm, which directly determines the evaluation and utilization value of the algorithm, and therefore is quite important. The requirements for the results are rigorous.

The reference paper is still *Image-to-Image Translation with Conditional Adversarial Networks* [6], in which different image conversion effects can be achieved by simply replacing the dataset. Therefore, in this attempt, we hope to achieve different functions by replacing the dataset and solving some problems that existed in the dataset in the previous tests, such as the images between different domains do not fit completely.

In the initial stage of the algorithm design, we want to have the algorithm automatically generate different texture maps of the Courtyard-style buildings on the land parcels by inputting land parcels of specific shapes and sizes. Therefore, there are only two objects that the model has to interact with: the parcel and the texture map of the Courtyard-style building. In this attempt, this thesis hopes to train the model with a large dataset of parcel-buildings texture maps to master the design of the corresponding reasonable courtyard-style building texture maps from a specific shape of the parcel.

3.2.2 Collection and manufacture of datasets

The dataset used this time is much simpler in terms of information than the previous one. As shown in Figure 3.8, the black area on the left side is the texture map of real-world co-located buildings that exist, including buildings in Dalian, Qingdao, Berlin, Florence, Oslo, Paris, and other regions. These architectural texture maps were obtained through BIGMAP and contained 40 buildings in different locations. Moreover, the blue areas on the left and right are land parcels that have been manually marked out in collaboration with the School of Architecture and Art using Adobe Photoshop. This land parcel corresponds to the building defined by the construction team and the architect when the buildings were built. The algorithm needs to focus on the condition that the algorithm needs to design the buildings and generate the building texture within this defined area.

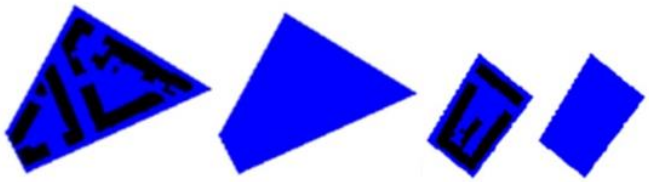

Figure 3.8 Dataset of land block information and architectural texture

Although a sample of 40 different Courtyard-style buildings is a considerable number in architectural design, it is still far from enough for model training. Therefore data enhancement operations are still needed for the dataset. As in Figure 3.9, each image is rotated by 360°, and every 10° is saved as a new sample. In this way, data augmentation was performed on all 40 samples, and finally, 1440 samples were obtained. In addition to this, a random left-right symmetric operation is performed for all samples when training in batches.

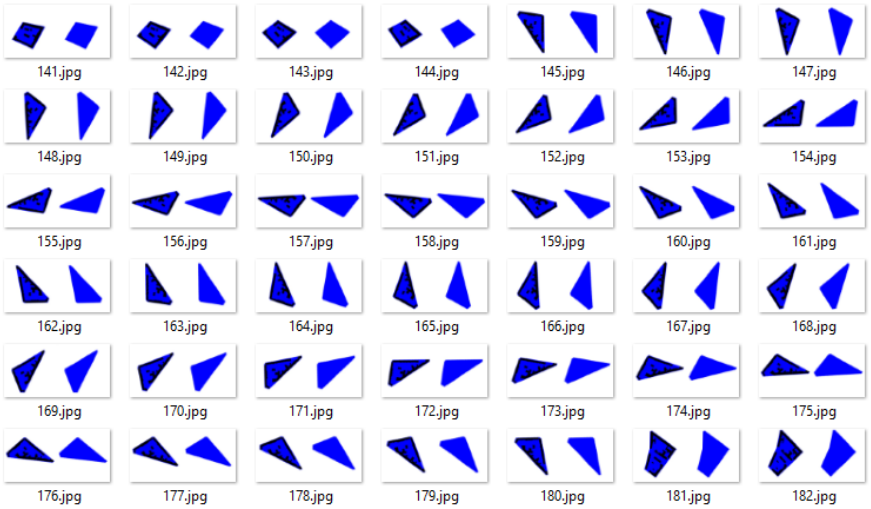

Figure 3.9 Data enhancement with Rotation

3.2.3 Experiment and analysis

The development environment, generator, discriminator, and other model structures remain exactly the same as before, with only the data set replaced.

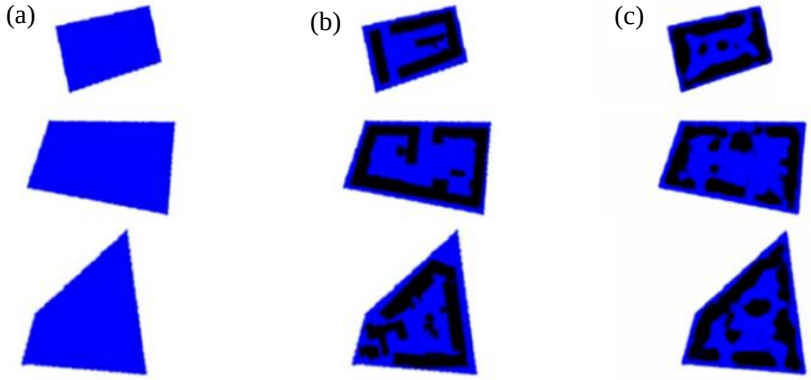

Figure 3.10 Result of epoch:25 (a) input image, (b) ground truth, (c) predicted image

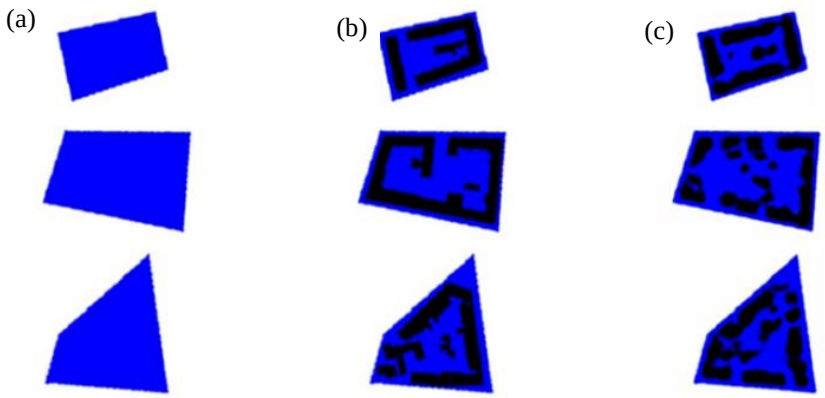

Figure 3.11 Result of epoch:50 (a) input image, (b) ground truth, (c) predicted image

Since the dataset contains much less information and fewer edges than the previous dataset containing satellite aerial images, the texture details are relatively simple, and the model converges quickly. After about 25 iterations, there is no significant progress in the model prediction.

As shown in Figure 3.10, 3.11, the predicted images achieve the goal of generating designs only in the specified parcels, and the designed results can show the enclosing characteristics of the Courtyard-style buildings. However, there is a severe lack of uniform and apparent structures in the predicted images, the model has difficulty in outputting sharp and straight building contours, is very vague in terms of detailing, and only a few very simple images are processed barely satisfactorily, but far from the purpose of the algorithm.

For the reasons for the problem, the following points are summarized.

(1) The number of data sets is too small. The number of non-resampled samples is only 40, which is expanded to 1440 by the data enhancement operation, containing limited adequate information.

(2) Real-world single-combination courtyard-style buildings often have other buildings in the center hall. For example, in Figure 3.12, there are three groups of courtyard buildings, but in the central hall of each group, there are also other types of small buildings. However, for the algorithm, it is only necessary to master the design capability of the combined courtyard-style buildings. These other buildings built in the mid-halls will interfere with the attention of the network.

Figure 3.12 Architectural texture of courtyard buildings

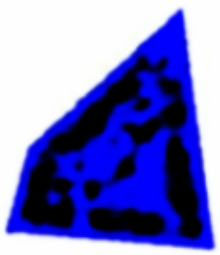

Figure 3.13 Predicted architectural texture

(3) The pix2pixGAN network tends to give very blurred results when dealing with this problem, as in Figure 3.13. Because the network's objective function does not have an internal control factor on its internal blurriness or clarity, the generator's output of the more blurred and distorted images also passes the discriminator. To solve this problem, one can try to introduce a measure of its internal, such as the loss of local descriptors or the use of the L1 paradigm. One can also try to improve the network model, for example, by using an attention mechanism.

After experimenting and practicing with GAN networks, the results are challenging to meet expectations. First, there are enormous obstacles in collecting and producing the dataset. Although Courtyard-style buildings are widely distributed, they are still historical buildings, just like Chinese garden buildings. Therefore, the samples available for collection are minimal, the quality of samples varies, and not all samples are suitable for making datasets. In making the dataset, the manual calibration is more work, and the labor cost is too high. Secondly, the network structure needs to be modified and improved significantly to solve the problems such as the ambiguity of the generated results.

4. OpenCV-based algorithm implementation for generating building layout plans

4.1 The basic idea of generating a building layout plan

4.1.1 Analysis of design rules for courtyard-style buildings

Figure 4.1 3D rendering of modern building: (a) courtyard-style building, (b) modern residential area

After trying and practicing on the GAN network, the effect of each link is difficult to reach the expectation. However, the Courtyard-style building (Figure 4.1(a)) and the modern residential area (Figure 4.1(b)) differ in architectural design in that the Courtyard-style building has a more graphical nature and pattern in the architectural texture map.

There are sometimes multiple ensemble groups of buildings on a single land parcel. For example, there are eight groups of buildings in Figure 4.2(a). Furthermore, graphically, one can think of the generation method roughly as a geometric cut of the original parcel according to a specific pattern, as in Figure 4.2(b). The resulting design will be more natural and diverse if this slicing is better simulated. After slicing, each slice is processed graphically to create an architectural texture. Finally, the final embellishment and finishing are done to create an architectural reference design.

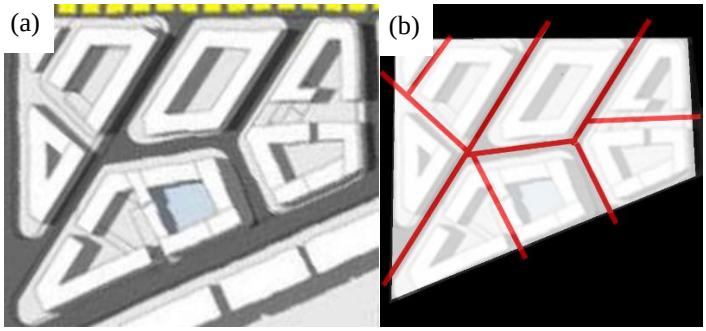

Figure 4.2 Architectural texture of courtyard-style building in Dalian Dongguan Street: (a) original image[4], (b) segmentation

The land parcels to be designed can be approximated as quadrilaterals or triangles. Few parcels of apparently other shapes were encountered in the collected sample, so the algorithm does not consider cases other than quadrilaterals or triangles.

The new small parcels created during the cut can also be primarily approximated as quadrilaterals or triangles. There are no cases in the existing sample where pentagons or other shapes are generated when parcels are cut. Therefore, the rules in the cut score are strictly limited. Based on discussions and analysis with the School of Architecture and the Arts, the following more basic scenarios for segmentation have been identified.

For quadrilaterals:

(1) Choose a point on one side and its opposite side, and tangent along with both points

(2) Choose a corner point and a point on a side not adjacent to the corner point, and tangent along with the two points

For a triangle:

(1) Choose a corner point, and a point on a side not adjacent to the corner point, and tangent along with the two points

(2) Find the center of mass of the triangle and, starting from the center of mass, make parallel or perpendicular lines in the direction parallel to a particular side or perpendicular to a side, segmenting all three sides at once

More triangles and quadrilaterals of different shapes can be obtained with these four options after a single tangent. Very mixed results can be obtained after making successive cuts.

It should be noted that some of the tangent methods can cause the near-circularity of the generated building contours after segmenting to be so low that they appear unreasonable (in short, the generated Courtyard-style buildings are too slender). It is also possible that the generated building footprint is too small, or the generated angle is too small (too sharp). Therefore, the geometry of the generated results (e.g., area, near-circularity) needs to be tested by checking the geometry of the generated results. If the metrics of the tangent results do not meet the requirements, they can be re-segmented.

When slicing, it is necessary to always keep an eye on the area of the current graph to be sliced and the area after slicing. The area after slicing cannot be too small or too large. Moreover, there are many buildings in some samples with a particular gap in the floor area. Therefore, the slicing process also needs to maintain a certain degree of diversity in building footprints.

4.1.2 General flow of generating building layout plans

The algorithm is mainly based on the OpenCV library and tries to automate the design by using the basic design laws of the Courtyard-style building refined manually graphically and geometrically. Its specific flow is shown in Figure 4.3.

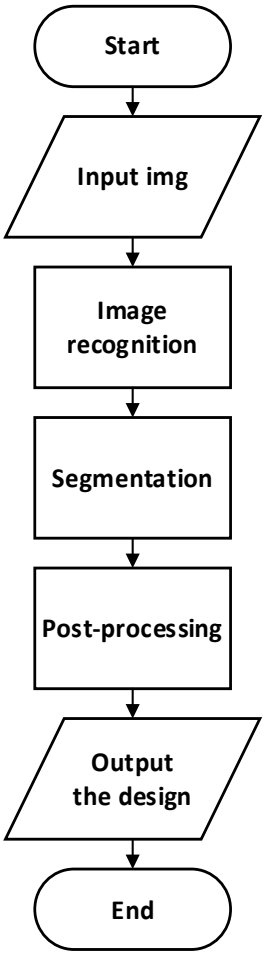

Figure 4.3 The overall flow of the algorithm based on OpenCV

The first step of the algorithm is to input an image file in .png or .jpg format. It contains information about the land parcel to be designed.

The second step is Image Recognition, in which the algorithm uses graphical and geometric methods to recognize the image and process it into digital information for computing.

The third step is Segmentation, in which the algorithm simulates an architect and uses a set of laws to complete a series of rational and diverse designs with a certain probability.

The fourth step is visualization and post-processing. In this stage, the algorithm needs to visualize the grayscale map cut in the previous stage, generate the building texture map, add details, and perform 3D visualization operations.

Finally, the algorithm will output the building texture map and the image after 3D visualization.

4.2 Input of land parcel information

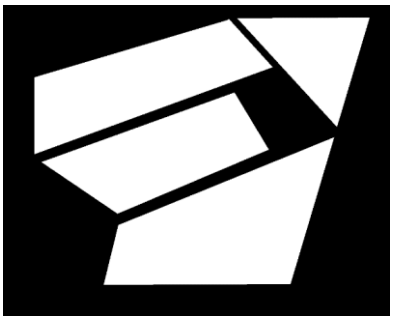

Figure 4.4 Data input example

First, the algorithm inputs an image img in .png or .jpg format. In this image, the background is black, and the land parcels in which the Courtyard-style buildings need to be designed and built are painted white. The number of land parcels can be one or more, but their shape must be triangular or convex quadrilateral, as in Figure 4.4. The image resolution satisfies the real-world scale of 1 meter equals 10 pixels (1m = 10pix).

After inputting the image img, it is converted to an 8-bit grayscale map. To prevent this, the algorithm sets all pixels with values of 126 and below to 0 and all pixels with 127 and above to 255.

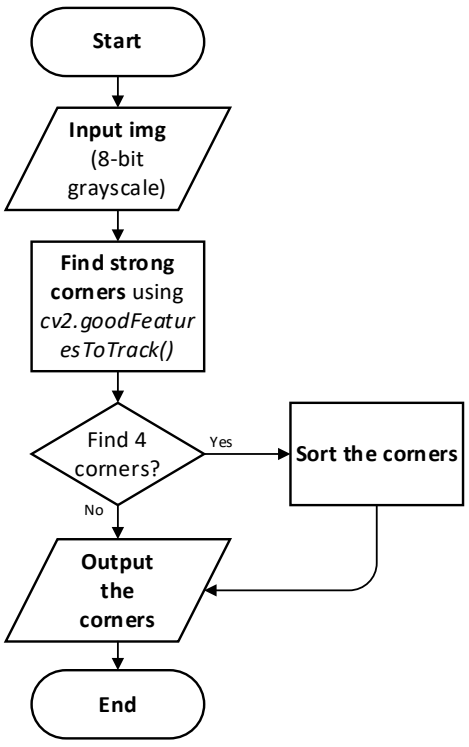

Figure 4.5 The overall flow of the recognition of vertices

4.3 Recognition of land parcel information

In this algorithm, since the main object processed is an 8-bit grayscale map, a series of functions for recognition are required to enable the algorithm to acquire information in the grayscale map, including the number of polygons, the location of polygon corner points, angles, edge locations, area, and near-circularity. By recognizing and converting the image information contained in the grayscale map into numbers and coordinates, the algorithm can acquire the ability to design in geometry and evaluate the merit of the design.

4.3.1 Recognition of polygon's vertices and angles

In this algorithm, it is important to recognize the polygon's vertices. The data structure of the vertices and other points are here in the form of a 1×2size array. This array represents the coordinates of the point on the grayscale map. For polygons, the vertex set uses the data structure of an array of 3×2size (triangles) or 4×2size (quadrilaterals). Moreover, it is necessary to recognize the coordinates of the vertices of the polygon accurately and to sort the vertices in the case that the polygon is a quadrilateral so that the order of the vertices in the vertex set is clockwise (counterclockwise is theoretically possible) order. The ability to accurately order the polygon vertices is the key to the problem. The algorithm designs a method for recognizing the vertices of polygons. Its main flow is shown in Figure 4.5.

In the corner point finding phase, this thesis uses functions from the OpenCV library to accomplish the corner point detection task. In the initial design, Harris corner point detection is used in this thesis. The goodFeaturesToTrack algorithm is an improved version of Harris corner detection, which uses the Shi-Tomasi operator by default and can easily set the maximum number of corner points detected and the quality of the corner points. The specific parameters of the algorithm are maxCorners=4, qualityLevel=0.1, and minDistance=5. After the corner points are found, the coordinates of all corner points are stored in an array of 3×2size (triangles) or 4×2size (quadrilaterals). This array is then sorted by column in subsequent processing.

However, there are still some problems with this algorithm. As shown in Figure 4.6(a), the quadrilateral does not detect all the corner points (the positions marked in red are the recognized corner points) for the parameter qualityLevel=0.1056. This problem may be because the corner points that are not detected are of poor quality. After all, they are formed at too large an angle. However, if the parameter qualityLevel is lowered to 0.1055, as in Figure 4.6(b), not only the remaining corner point is not detected, but a wrong point is added instead. This new problem may be due to the low resolution of the grayscale map, such that some positions are even more accessible to detect than the positions of the actual corner points, as

in Figure 4.6(c). For the same reason, with the parameter qualityLevel=0. 1056, as in Figure 4.6(d), a triangle is detected with four corner points.

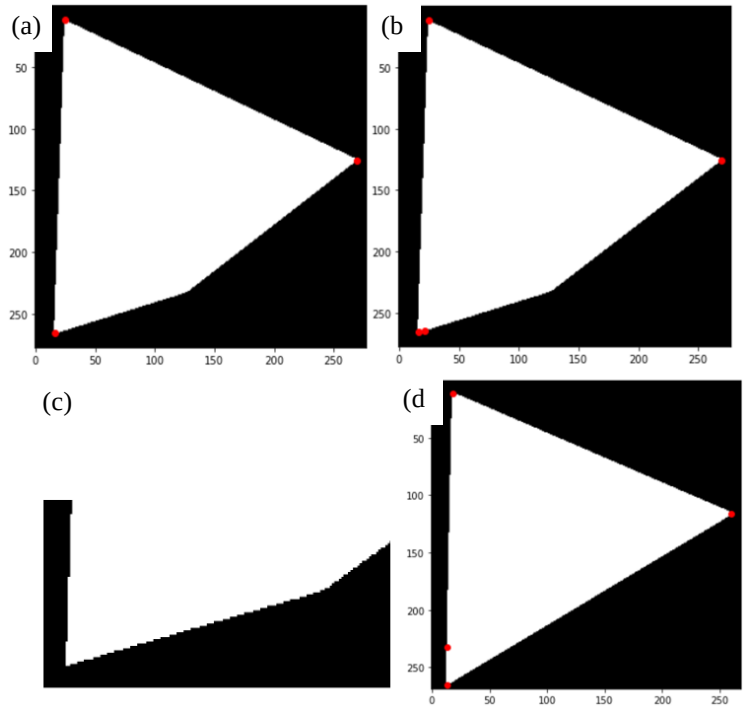

Figure 4.6 Error in corner detection: (a) cannot detect all the corner points, (b) detect wrong corner points, (c) position where the error occurred, (d) detect wrong corner points

Corner point detection is the basis of the whole algorithm. A problem in this part of the test can lead to erroneous results or a dead loop in the program. An attempt to increase the resolution of the grayscale map can improve the probability of this problem to some extent, but it can also significantly slow down the program. Therefore, a balance can only be achieved between the two. Otherwise, it is difficult to solve the problem completely.

In the stage of sorting diagonal point arrays, the primary purpose is to use the calculation of the angle between vectors and vectors to sort them. The main flow is shown in Figure 4.7.

Since the coordinates of all the vertices have been obtained, the coordinates of the center of mass can be obtained by averaging the horizontal and vertical coordinates of all the vertices separately. The vector is then obtained by subtracting the coordinates of the corner points from the center of mass. The angle between the vector and the positive direction of the x-axis is then obtained. Finally, by sorting the angle points according to the magnitude of the angle, a new ordered array of angle points can be obtained.

After obtaining the ordered array of vertices, it is necessary to obtain the angle between each vertex and the adjacent edge from the coordinates of the vertices. The algorithm designs a method to achieve this goal. Its implementation mainly takes advantage of the fact that the

vertex array has been sorted and can easily find the adjacent vertices. Thus, the vectors of two adjacent edges can be easily obtained, and thus the angle between the two vectors can be obtained. Finally, the function returns an array of 3×3size (triangles) or 4×3size (quadrilaterals). The first two columns of the array are used to store the coordinates of the vertices, and the last column is used to store the angles.

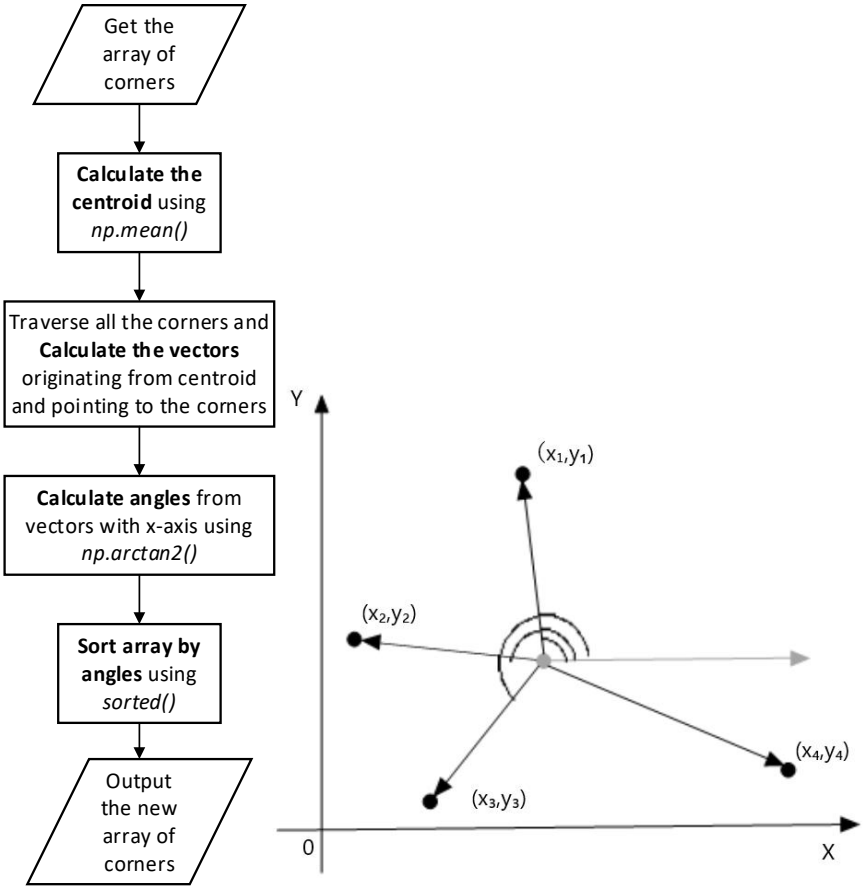

Figure 4.7 The overall flow of the vertex array sorting

4.3.2 Recognition of polygon's edges

In this algorithm, it is vital to recognize the edges of polygons. The data structure of all polygons' edges and line segments is here in the form of an array of 1×4size. The first two and last two bits of this array are the coordinates of the two endpoints of the line segment, respectively. These two endpoints may cause relatively large errors if they are close to each other. Therefore, finding the endpoints that are far away and accurate becomes the key to the problem. The algorithm is designed with two methods to recognize the edges of the recognized polygon.

The first method is the first to be designed and implemented. This function can recognize the edges of polygons when the coordinates of polygon vertices are not obtained or are difficult to obtain.

This function requires Canny edge contour detection for a single polygon first. After obtaining its contour, the Hough transform is applied to its contour to obtain the number of edges of the polygon and its formula.

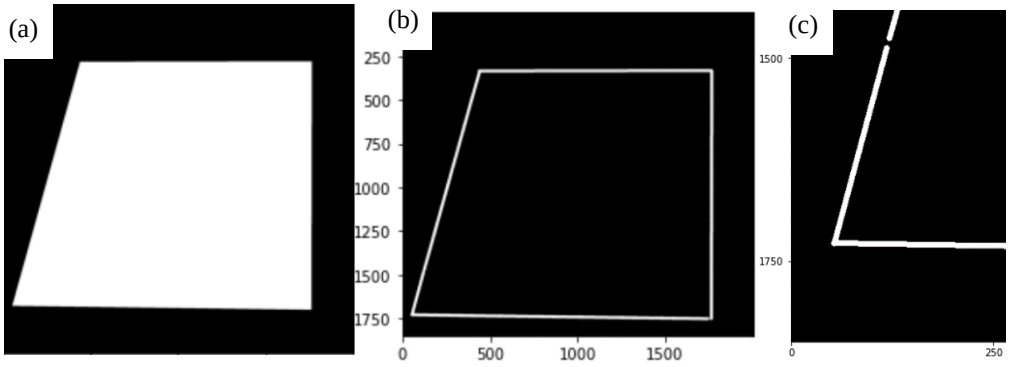

Figure 4.8 An example for recognizing the edges of a polygon: (a) original image, (b) contour detected, (c) unlinked lines

However, due to the apparent shortcomings of the Hough transform algorithm within the OpenCV library in processing the edges detected by the Canny edge detection algorithm. Take Figure 4.8(a) as an example, the grayscale map size is 2000×2000, and the effect of the Hough transform on the result after Canny edge detection is shown in Figure 4.8(b). Although the Hough transform finds out the straight lines on the edges, many of them are not connected, as shown in Figure 4.8(c). Ideally, the graph should have been detected with four straight lines. In fact, a total of 23 straight lines were detected in Figure 4.8(a), including [1755, 1262, 1755, 703], [749, 335, 1277, 335], [1754, 1751, 1754, 1263], [1278, 334, 1755, 334], [1756, 702, 1756, 335], [434, 336, 748, 336], [992, 1741, 1335, 1746], etc. A closer look reveals that the cause of the problem is that multiple straight lines may be detected on the same edge of the polygon. These straight lines may be connected at the beginning and end, or they may overlap. Moreover, this problem usually occurs on the edges close to the perpendicular to the x-axis.

For this problem, multiple parameter corrections to the Hough transform in this thesis do not solve the problem well, so in this project, additional work is needed to improve the Hough transform algorithm in OpenCV. The improved version of the process to obtain the edges by the Hough transform algorithm is shown in Figure 4.9.

The idea of clustering is utilized to solve the problem. The improvement goal is to return only three lines for triangles and four lines for quadrilaterals. Since the additional line segments recognized by the Hough transform tend to be distributed on the same side of the polygon, these line segments all possess the same slope, horizontal intercept, and vertical

intercept. The line segments for these three features can be grouped by clustering the line segments on the same edge. After clustering, all endpoints of line segments in the same cluster are traversed, and the distances between them are calculated, with the two farthest points being the final choice. This is done for each cluster to obtain the desired result.

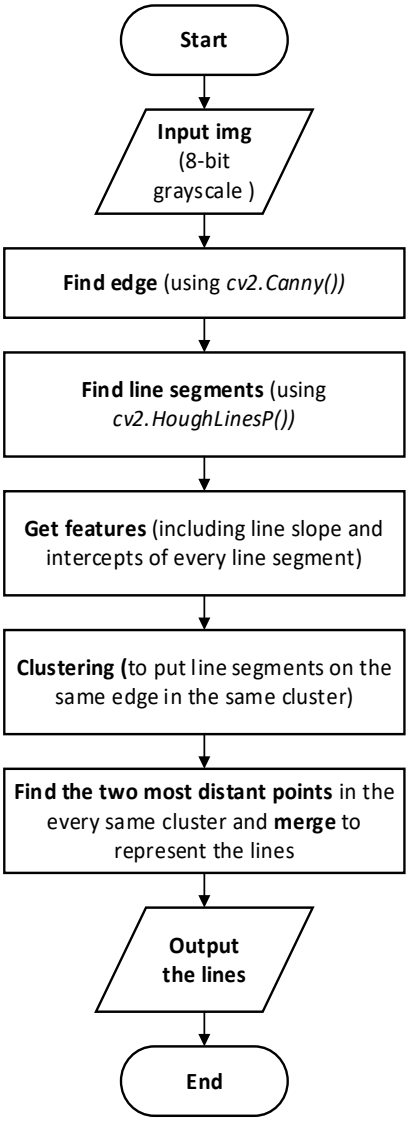

Figure 4.9 The overall flow of the improved version of the Hough transform

The specific flow of the Get features section is shown in Figure 4.10.

As in Figure 4.10, in the stage of obtaining features, this thesis creates an n×8-dimensional array. The line segments are stored with the line, the angles to be calculated, the horizontal intercept x-intercept, the vertical ending y-intercept, and the cluster number used in the next Clustering stage, respectively.

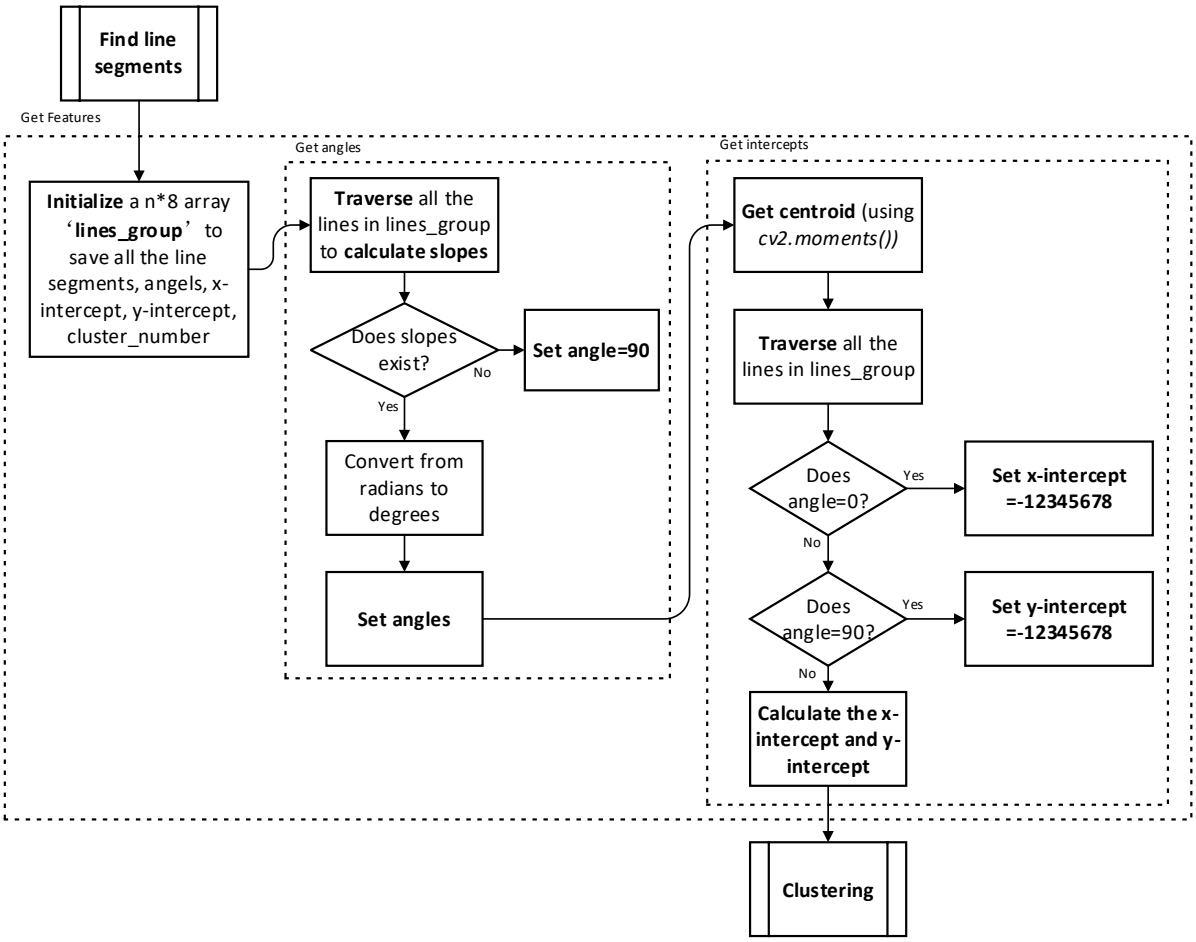

Figure 4.10 The overall flow of the feature recognize (Get features)

When calculating the slope of a line segment that is close to horizontal or perpendicular to the x-axis, the horizontal or vertical intercept can become large or small in value, which can lead to numerical overflow or to errors that can lead to large gaps in the intercept of the same parallel or perpendicular line segment. To prevent this phenomenon, the horizontal or vertical intercepts of line segments close to the horizontal or perpendicular to the x-axis are set specifically to -12345678. Only one of the horizontal or vertical intercepts must be within the allowed error range when clustering. However, if this is done, there may be cases where the horizontal or vertical intercepts of lines on two different edges are the same for some polygons. Therefore, instead of calculating the distance from the origin of the intersection point with the coordinate axis, the intercept here is the intercept of the "line passing through the center of mass of the polygon and parallel to the coordinate axis." In particular, therefore, the x-intercept in Figure 4.10 refers to the intercept with the "line passing through the center of mass of the polygon and parallel to the x-axis," and the y-intercept refers to the intercept with the "line passing through the center of mass of the polygon and parallel to the y-axis." The y-intercept

is the intercept with the "line through the center of the polygon and parallel to the y-axis. The coordinates of the center of mass will be obtained.

The detailed flow of the Clustering section is shown in Figure 4.11.

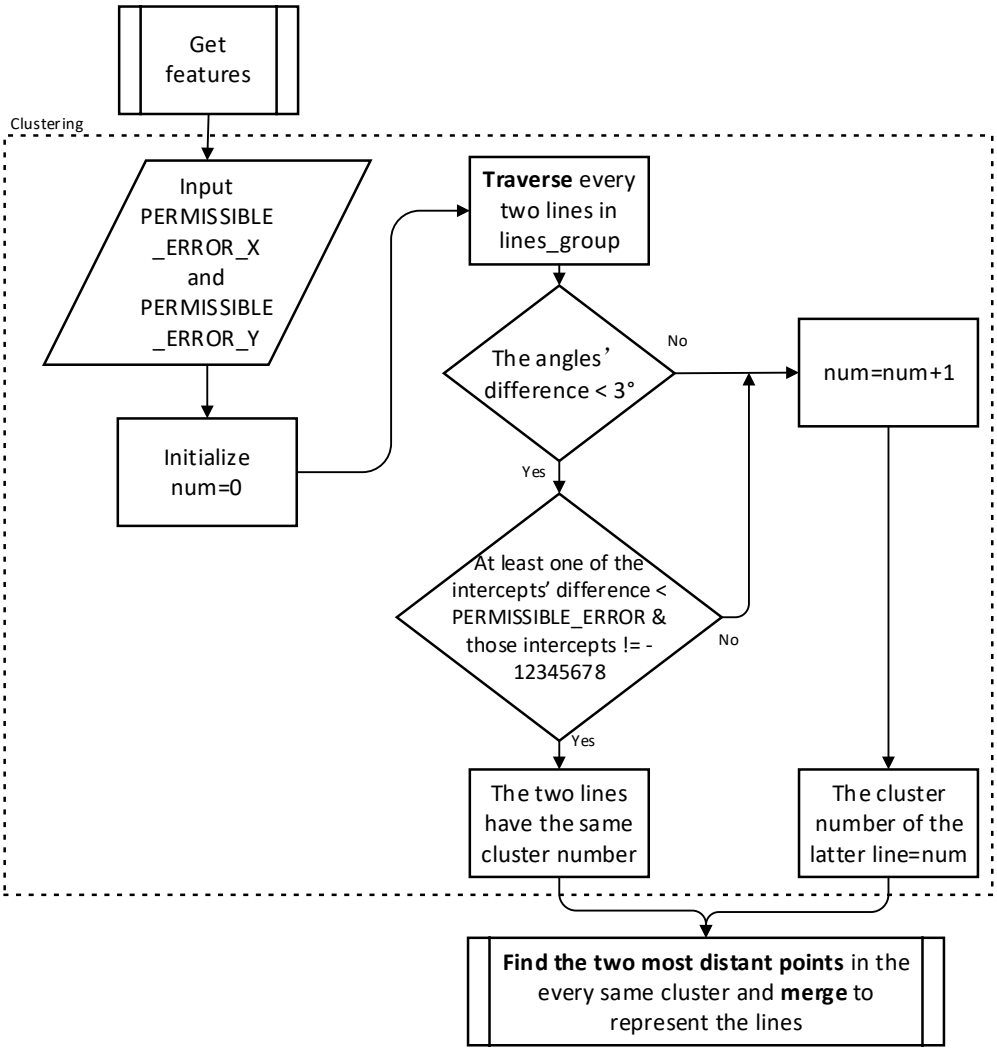

Figure 4.11 The overall flow of the feature clustering (Clustering)

Since the data processed by this algorithm are mainly grayscale maps, and the resolution of grayscale maps is not high to improve the operation's efficiency, there is a relatively obvious error when using coordinates for calculation. Therefore, when clustering, the primary approach taken is to compare whether the size of the difference between the feature values is within the allowed error range. As shown in the figure, a permissible error, PERMISSIBLE_ERROR_X, and PERMISSIBLE_ERROR_Y, is entered in the clustering phase, and the permissible error is generally related to the resolution of the grayscale map processed by the algorithm. Here, the permissible error is tentatively set to 1/50 of the grayscale image size processed by the

algorithm. e.g., if the grayscale image size is 2000×1000, then PERMISSIBLE_ERROR_X is 40 and PERMISSIBLE_ERROR_Y is 20.

When clustering is done again, a variable num is first initialized and assigned to 0. This value will indicate how many different edges have been found (num-1). Every two of every two lines in the lines in a group will be traversed. If the angle difference between the two lines is less than 3° and the corresponding difference between x-intercept or y-intercept is less than PERMISSIBLE_ERROR_X or PERMISSIBLE_ERROR_Y if it is not equal to -12345678, then the cluster number of both can be set to (the cluster number of the first line segment in the array is assigned to 0). Otherwise, it is assumed that the two segments are not distributed

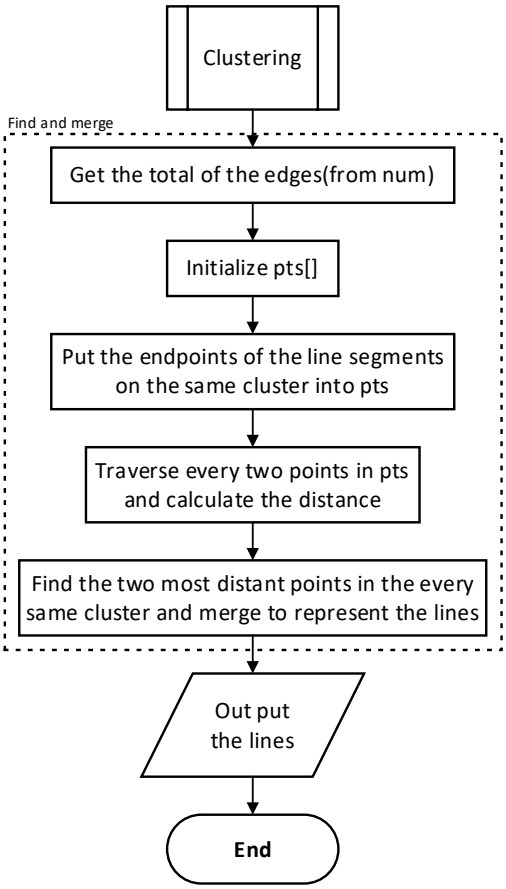

Figure 4.12 The overall flow of finding the two most distant points

on the same edge, num=num+1, and the cluster number of the segment with a cluster number that has not yet been assigned is assigned to num.

In the last stage, it is necessary to find the two most distant points in every same cluster and merge them to represent the lines. The flow is shown in Figure 4.12.

Since the clustering of line segments has been done in the Clustering phase, the line segments of the same cluster should now all be distributed on the same edge of the polygon. Therefore, the endpoints of these line segments should also be on the same edge. Since the data type of the line segment is a 1×4 array, the first two and the last two are the coordinates of two endpoints, so it is only necessary to split the data of the line segment to get its endpoints.

By putting all the endpoints of the line segment on the same side of the polygon together, iterating through all the points, and calculating the distance of each two points, we can get the coordinates of the two farthest points. These two points can be reassembled to give a more accurate representation of the polygon's edges.

The second method is much simpler in principle but requires the input of the recognized vertices. In the function, the vertices are connected sequentially in the order they are entered, forming an expression for the edges and outputting it. Although it is more geometric and runs much more efficiently than the first method, considering that the corner point detection algorithm goodFeaturesToTrack still has some flaws (see 4.3.1 for details), so for some polygons of specific shapes, the vertices may be incorrectly recognized or may not be recognized. In this case, the error may cause the second method to fail to recognize the normal edges, so the second method cannot fully replace the first method in terms of functionality. In

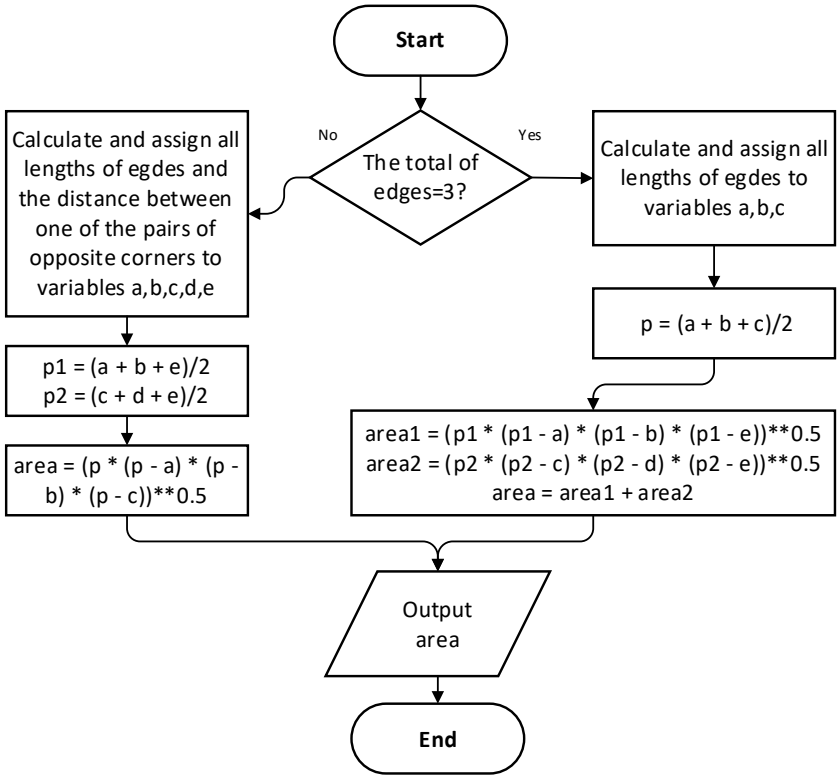

Figure 4.13 The overall flow of acquire the area of the polygon

fact, in this algorithm, the second method is used to obtain edge information in most cases, and the first method is used only when valid corner points are not obtained.

4.3.3 Recognition of polygon's areas

In this algorithm, there are two main ways to obtain the area:

1. by finding the area from the coordinates of the vertices of the polygon
2. by counting the number of pixels based on the pixel value size.

The first method is the first way that comes to mind in this thesis. This method mainly uses the vertex coordinates of polygons and Heron's formula for geometric operations. The specific flow is shown in Figure 4.13.

Since this algorithm only considers the case of triangles and arbitrary convex quadrilaterals, if the number of sides of the polygon is not 3, then the polygon must be a convex quadrilateral.

The second method is different from the first one. For an 8bit grayscale map img, the number of all pixels with a pixel value of 255 is counted directly. This method is more straightforward and more efficient, so after implementing this method, all functions in this algorithm that need to calculate the area of polygons are replaced with this method.

4.3.4 Other recognition functions

In addition to the above recognition functions, the algorithm still includes some necessary recognition functions used repeatedly in the following segmentation stages. These include: finding the center of mass of a geometric figure, calculating the intersection of two lines, finding the length between two points, obtaining the perpendicular of a point to a line, calculating the distance from a point to a line and obtaining the perimeter.

4.4 Segmentation of land parcels

4.4.1 General flow of segmentation method

This stage is the core function of the whole algorithm, and its effect affects the usability and reference value of the whole program. After obtaining the grayscale map and the recognized information, the algorithm can take into account the shape characteristics, area, and angle of the polygon, decide the slicing method randomly according to the set probability, and represent it on the grayscale map. Finally, the grayscale map after slicing is returned. Its specific flow is shown in Figure 4.14.

First, input an 8-bit grayscale map. Initialize lists: the waiting_list, completed_list, and temp_list, which are used to store the grayscale map to be processed, the grayscale map that

has been cut, and the grayscale map that has been temporarily saved, respectively, in the following process.

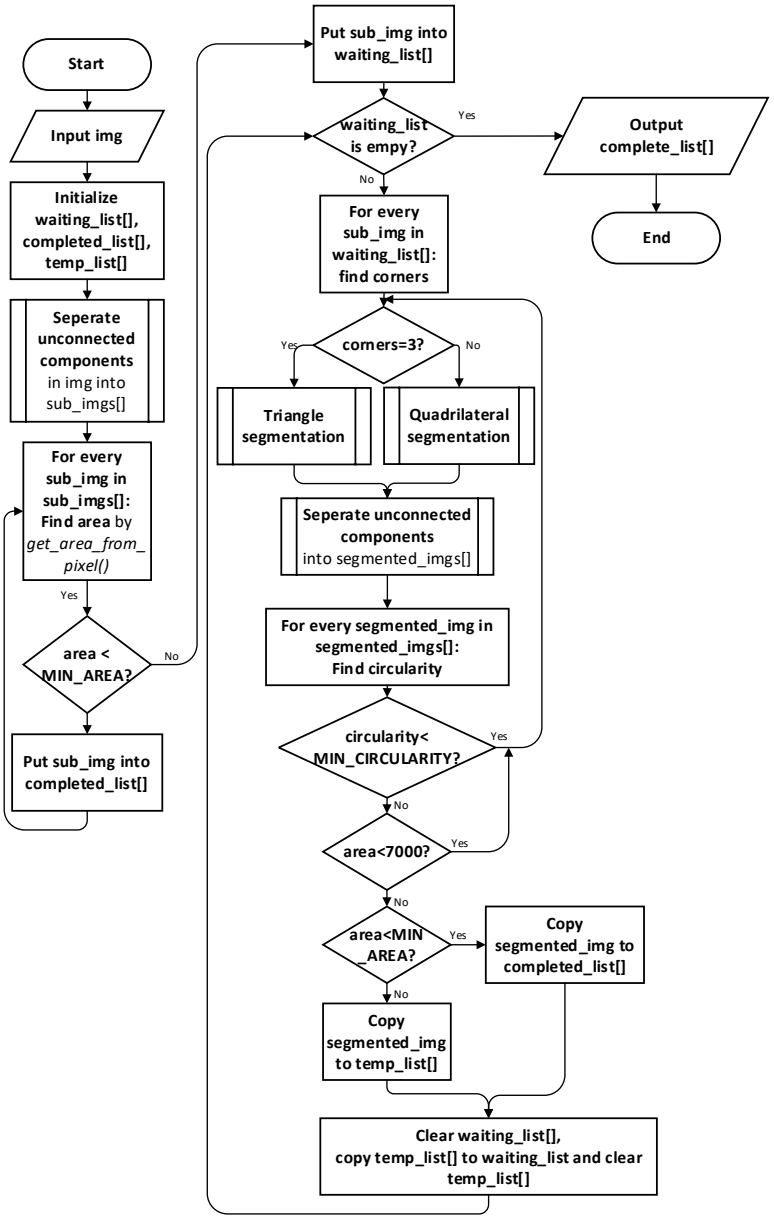

Figure 4.14 The overall flow of segmentation

Then, since the input original grayscale map may contain multiple connected components (land parcels), it is necessary to perform a connected component separation operation (see 3.4.2 for a detailed description). After the separation of connected components is achieved, each connected component(parcel) is placed separately on a new gray map. These new grayscale maps containing only one connected component(parcel) are placed in a list sub_img.

Next, find all the land parcels' areas in the grayscale map in sub_img. If the area is less than the minimum required area MIN_AREA, it is considered not to be cut and placed in completed_list. Otherwise, it is placed in the waiting_list.

A loop body follows this. As long as the waiting_list is not empty, it will keep looping. In this loop body, the grayscale maps in the waiting_list are traversed. First, the grayscale graph's corner points (vertices) are recognized. If there are three vertices, it is a triangle land parcel, and a predefined Triangle segmentation operation is performed (see 4.4.3 for a detailed description). This operation will cut the triangle and the land parcel outline according to their geometric properties. The result is shown in Figure 4.15, which yields a grayscale map containing two connected components.

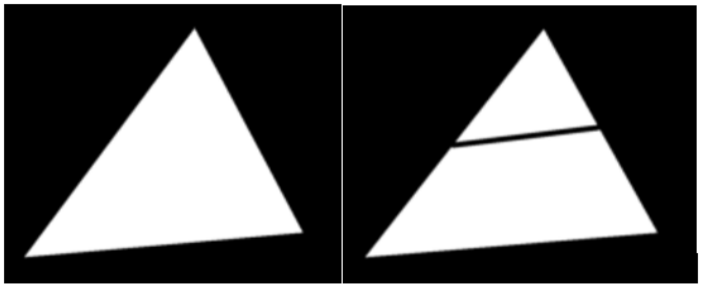

Figure 4.15 Example of segmentation result

If there are four vertices, it is a quadrilateral parcel, and a predefined Quadrilateral segmentation operation is performed (see 3.4.4 for a detailed description). This operation cuts the geometric properties of the triangle and the parcel outline to obtain a grayscale map containing two connected components.

The cut grayscale map may not contain the required two connected components, or the area may still be large and need to be cut again. Therefore, the grayscale map just after cutting is separated from the connected components. The two separated grayscale maps are analyzed and inspected. The detection metrics include near-circularity and area. If the quadratic follow up result of the near circularity of either of the two grayscale maps is less than the minimum allowable value APRATIO or the area is less than the minimum allowable area MIN_TOLE_AREA (generally 7000 pixels), the cut just performed is considered to be a failure and the result obtained is not architecturally correct. If this is the case, the cut result must be discarded, returned to the top of the loop at this layer, and cut again. If the near-circularity and area of both grayscale maps meet the requirements, they pass the test. Further, determine whether their areas are smaller than the minimum required area (MIN_AREA). If True, no further cutting is needed, and they are placed in completed_list; if False, they are placed in temp_list.

After completing the traversal of the grayscale map in waiting_list, empty waiting_list. Deep copy the grayscale map in temp_list to waiting_list and empty temp_list again. Perform

the next round of traversal of the grayscale maps in waiting_list. After a certain iteration, the area of all grayscale maps will meet the requirement, and then there will be no grayscale maps in temp_list to copy to waiting_list. In the conditional judgment of the loop, waiting_list is empty and does not meet the condition of continuing the loop. Then, the loop body is completed and jumps out. At this point, completed_list contains a lot of cut grayscale maps containing a single concatenated field, and these are the desired output.

Although the cutting method is based on certain qualifications, it still relies heavily on probability, so it may produce some unreasonable results uncontrollably when dealing with some polygons with special shapes. Two indicators, quadratic heel APRATIO of the near-circular rate and MIN_TOLE_AREA of the minimum allowable area, are set to solve this problem. If the result does not satisfy either of these two indicators, it needs to be recut. Although this completely solves the above problem of generating unreasonable results, if the indicator is set too harshly, it may seriously reduce the operational efficiency because the cutting method is based on probability. The program may still not satisfy the indicator after many attempts and need to keep trying again and again.

Take APRATIO, for example. The minimum allowable value of the quadratic heel result from the near-circular rate, and the size of its value seriously affects the efficiency of running the program. A runtime test is performed to further understand the extent to which it specifically affects running efficiency. The test consisted of the time required to run the complete parcel segmentation function with the variable APRATIO. 7 groups were run at a time, with each group running the parcel segmentation function 20 times for the same parcel at a resolution 1000×800. The average time of the seven groups and the standard deviation between the times of each group were counted. The average time of the runs reflects the basic running efficiency of the program. Since this algorithm contains probability, the standard deviation of the running time is added to measure the stability of the running efficiency. The test results are shown in Figure 4.16.

Testing environment:

Operating system: Windows 10 Home 21H2
Processor: Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz
Installed RAM: 32.0 GB
GPU: NVIDIA Geforce RTX 3080 × 1
IDE: Jupyter Notebook 6.3.0
Development Language: Python 3.8.8
Image processing framework: OpenCV4.5.5

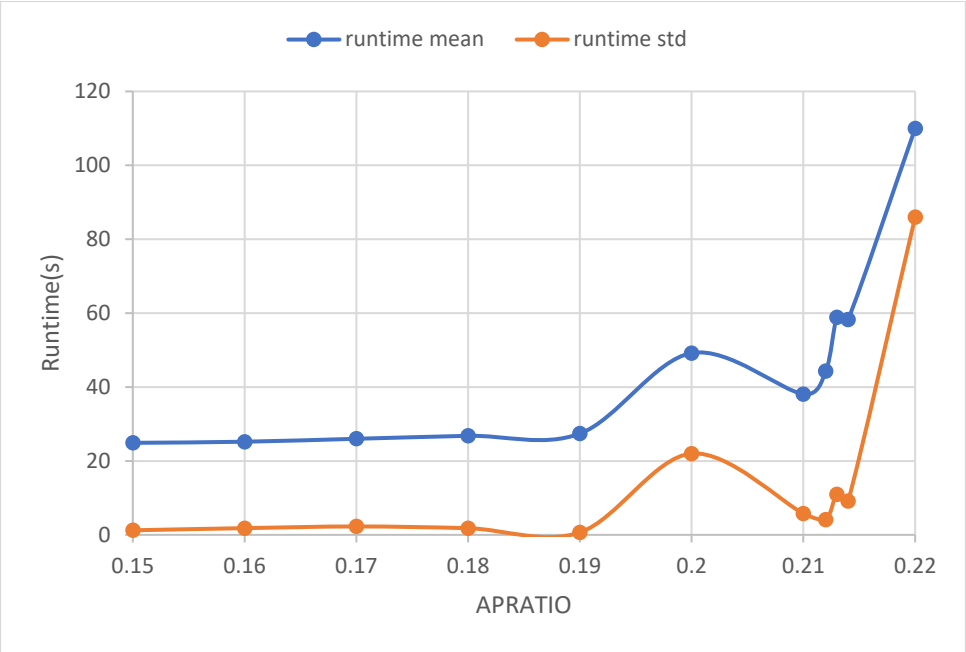

Figure 4.16 Test results of the effect of APRATIO on program efficiency

According to the graph, if APRATIO is set greater than 0.21, the average running time and standard deviation are significantly increased, indicating that the efficiency and stability of the program's operation are significantly reduced. The time to generate images at this point is often too long to meet the architectural requirements. Therefore, setting APRATIO to 0.21 is a relatively more appropriate parameter configuration.

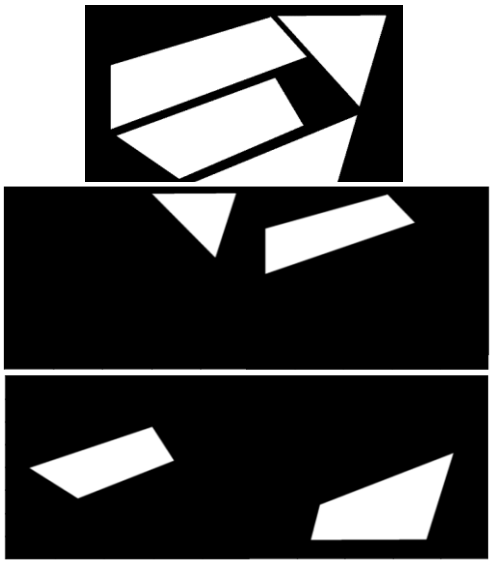

Figure 4.18 Example of result of
separating unconnected components

4.4.2 Separating disconnected components

In segmentation, the image may contain two or more connected components (parcel), as in Figure 4.17, while the operations Triangle segmentation and Quadrilateral segmentation can only operate on grayscale maps with a single connected component. At this time, it is necessary to separate the disconnected components and convert a grayscale map containing two or more connected components into two or more grayscale maps containing one connected component, and keeping the original position of each connected component unchanged, as in Figure 4.18.

The main flow of separating the non-connected components is shown in Figure 4.19.

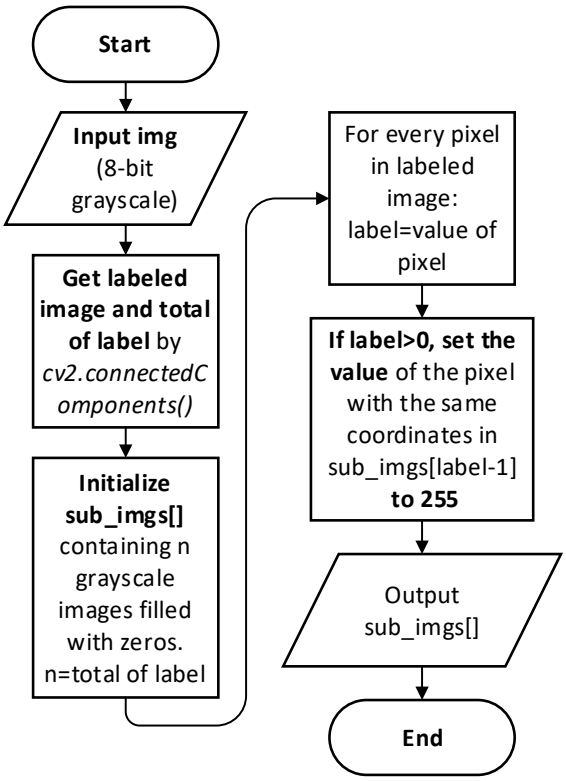

Figure 4.19 The overall flow of separating disconnected components

The input grayscale image is first processed using a function in OpenCV that detects the number of connected components in the 8-way connectivity mode and obtains the total number of labeled images and the number of disconnected components contained in the original grayscale image. The labeled image contains a label for each pixel of the original grayscale map, indicated by the numbers 1, 2, 3... (different numbers indicate different disconnected components).

Initialize a list sub_imgs containing n grayscale images of the same size as the original grayscale image (filled with zeros), n being the number of disconnected components total of the label.

Iterate over each pixel on the labeled image. If the pixel value is greater than 0, the pixel's value at the same location as the pixel is set to 25 on the grayscale map with the same index as the pixel value -1 in the list sub_imgs. For example, if the value of the pixel with coordinates (223,142) on the labeled image is 3, the value of the pixel with coordinates (223,142) in the corresponding grayscale map in sub_imgs is set to 255. value in the corresponding gray map in sub_imgs is set to 255.

Finally, the list sub_imgs is output.

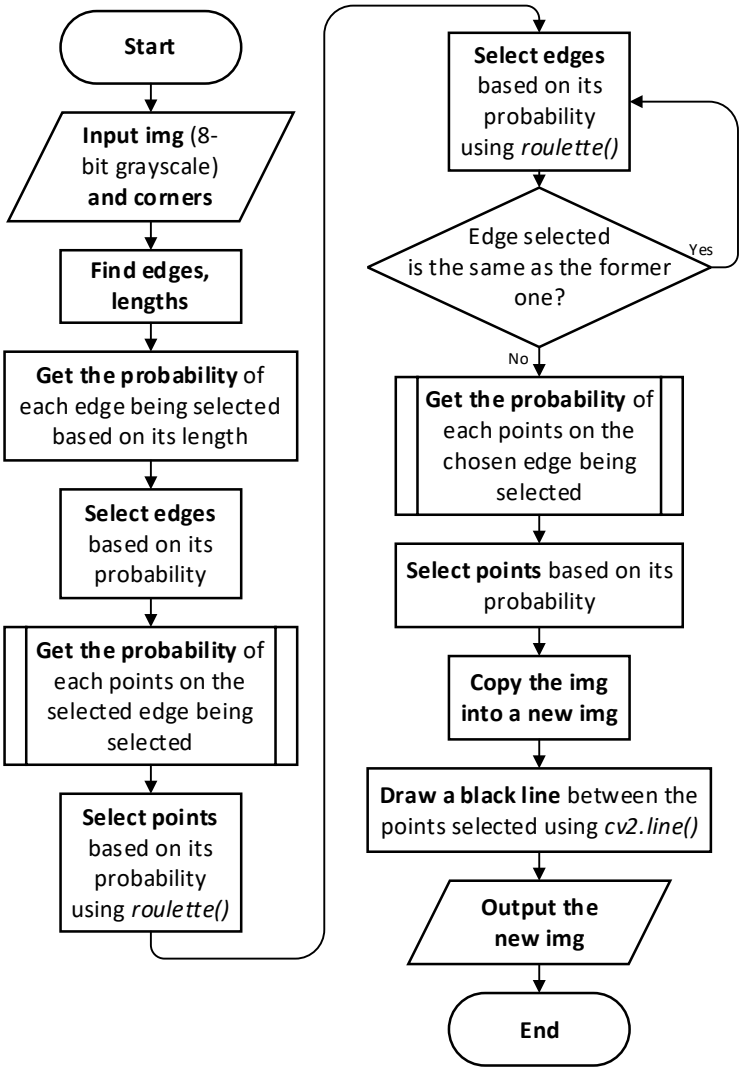

Figure 4.20 The overall flow of the first segmentation module of triangle segmentation

4.4.3 Triangle segmentation

The primary purpose of this operation is to slice the input triangle-shaped land parcels (grayscale maps). In order to ensure the diversity of its slicing methods, the program needs to introduce functions concerning probability and randomness. Furthermore, the rules of the cut score are based on the design experience of the Courtyard-style buildings summarized in the sample. This part has been modularized in code, and two modules of tangent rules have been added. If more tangent rules for triangle shapes are obtained, more modules can continue to be added.

For the first segmentation module, simply speaking, it is to select two sides of the triangle according to a certain probability distribution and then find two points from the two sides according to a particular probability distribution. The two selected points slice the triangle. The specific flow is shown in Figure 4.20.

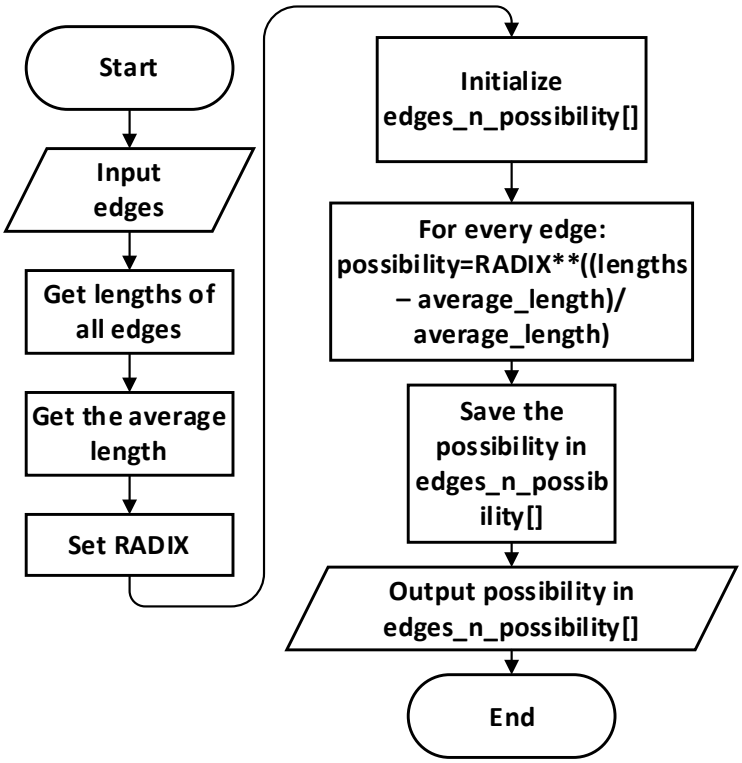

Figure 4.21 The overall flow of obtaining the probability of each edge

First, input the 8-bit grayscale map and its corner points, and then use its corner points to find the edges and their lengths. Next, the probability of each edge is obtained based on the length of the edge. The specific flow of obtaining the probability of each edge is shown in Figure 4.21.

For the input edge information, the length of each edge is first calculated and the average length. Then a parameter RADIX is set, and a list edges_n_possibility is initialized to hold the

probabilities and the corresponding edges. Where RADIX determines how much of a probability boost there will be when the lengths of the sides of the triangle are relatively long. After calculating the probabilities, all probabilities and corresponding edges are saved in edges_n_possibility and are output.

Using the exponential function to represent the probability of each edge being selected reasonably increases the probability of longer edges being selected while ensuring that each edge has a probability of being selected. This treatment ensures the rationality of the design while taking into account the diversity of the generated results.

After obtaining the probability of each edge, use roulette method (see 3.4.6 for details) to randomly draw an edge with a given probability and obtain the probability of each point in it being selected (see 3.4.5 for details). Based on the probability of each point, a point is randomly selected.

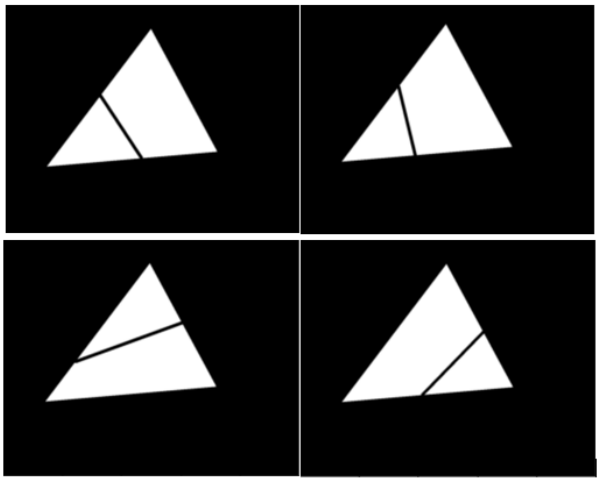

Figure 4.22 Examples of result of the first segmentation module of triangle segmentation

Next, draw an edge again with the given probability and ensure it differs from the previously drawn edge. Get the probability that each point on the selected edge is selected (see 3.4.5 for details). Based on the probability of each point, a point is again randomly selected.

Then copy the original image to a new image and connect the two points just selected with a black line. Finally, the new image is output. The results are generated based on probabilities, four of which are shown in Figure 4.22.

The second module of the cut rule differs more from the first one. In simple terms, the center of mass of the triangle is found first, and the cut is achieved by making a perpendicular line from the center of mass to each side or by making a line where the center of mass intersects each side and is parallel to the other sides. The specific process is shown in Figure 4.23.

First, input the 8-bit grayscale gray map and its corner points, and copy the original grayscale map into a new image.

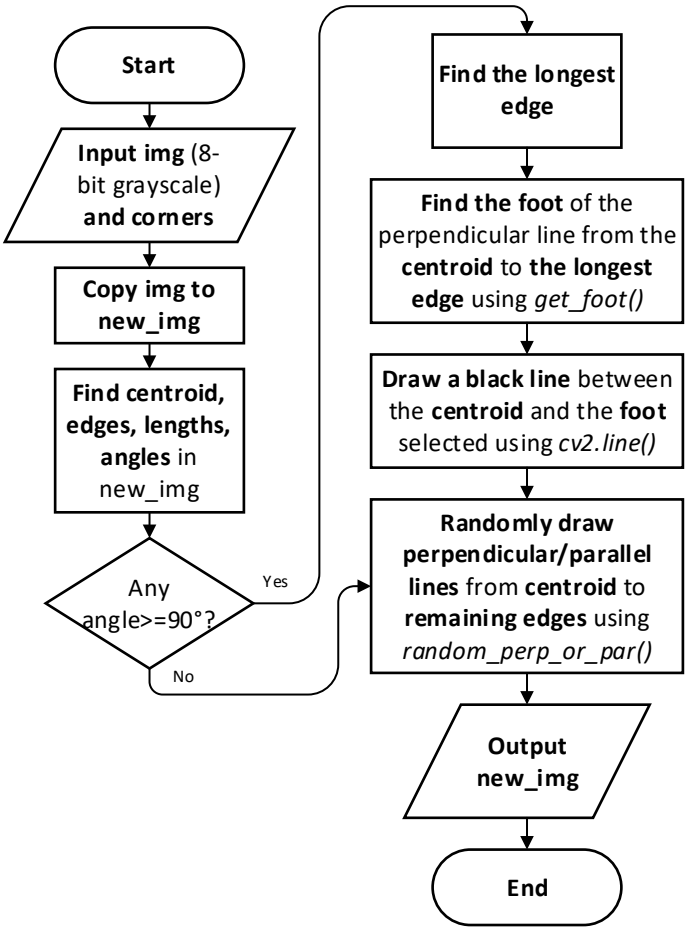

Figure 4.23 The overall flow of the second segmentation module of triangle segmentation

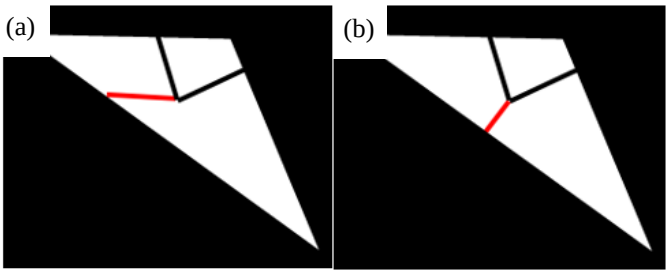

Figure 4.24 Two result of the second segmentation module of triangle segmentation: (a) wrong case, (b) right case

Then find the triangle's center of mass, sides, the length of each side, and the angle of each corner in the new image. Check if any angle is greater than or equal to 90°, i.e., if it is a right triangle or an obtuse triangle. For the longest side of a right or obtuse triangle, it does not

make sense to make a line where the center of mass intersects it and is parallel to the other sides. As shown in Figure 4.24(a), the result will be a concave quadrilateral in this division method. Therefore, only a perpendicular line can be made through the center of mass for the longest side of a right triangle or an obtuse triangle, as in Figure 4.24(b).

Therefore, when the triangle has an angle greater than or equal to 90°, its longest side will

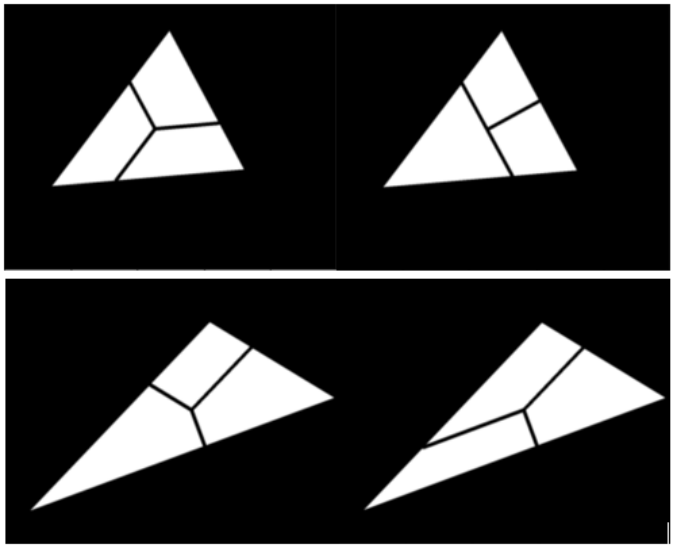

Figure 4.25 Examples of result of the second segmentation module of triangle segmentation

be found based on the length of each side previously found. Then find the perpendicular to the center of mass and the longest side. A black line connects the center of mass and the perpendicular foot. As for the two remaining sides, the cut is chosen with equal probability for each cut: perpendicular/parallel (parallel to one neighboring side)/parallel (parallel to the other neighboring side), and the cut is completed with the black line. If the triangle has no angle greater than or equal to 90°, each side is chosen with the same probability for each cut.

The final output is the cut image, a grayscale map containing three connected components. Some of the results are shown in Figure 4.25.

4.4.4 Quadrilateral segmentation

The main purpose of this operation is to slice the input quadrilateral-shaped land parcel s(grayscale maps). In order to ensure the diversity and rationality of its slicing method, the procedure introduces the same functions concerning probability and randomness as the slicing method for triangular parcels. Moreover, the rules of the cut score are based on the design experience of Courtyard-style buildings summarized in the sample. Two cut score rule modules have been added.

For the first cut rule module, the program selects an edge of the quadrilateral with a certain probability. After that, it finds the opposite side of the selected side. Two points are found according to a certain probability distribution from the two sides. The quadrilateral is sliced by the two points selected. The exact flow is shown in Figure 4.26.

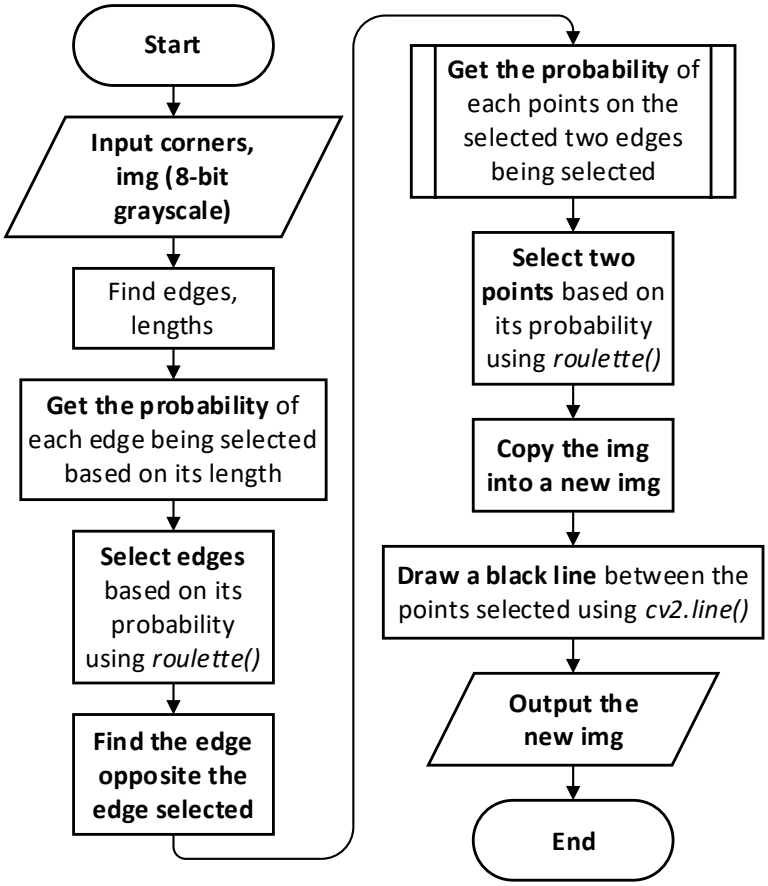

Figure 4.26 The overall flow of the first segmentation module of quadrilateral segmentation

First, input the 8-bit grayscale map and its corner points, and then find four edges and their lengths based on their corner points. Based on the length of each edge, the probability of each edge being selected is determined (see Figure 4.21 for the process). Next, use the roulette method to select an edge of the quadrilateral according to the calculated probability. Then add 2 to the ordinal number of the selected edge to get the ordinal number of the opposite edge (if the result is greater than 3, you need to subtract four again).

After selecting two sides of the quadrilateral, obtain the probability of each point in both sides being selected (see 4.4.5 for details). Based on the probability of each point, one point on each of the two sides is randomly selected. Then copy the original image into a new image,

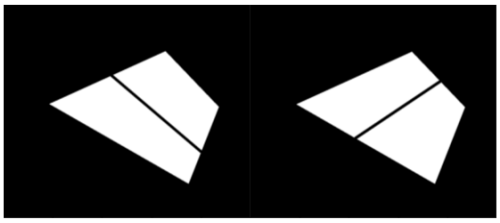

Figure 4.27 Examples of result of the first segmentation module of quadrilateral segmentation

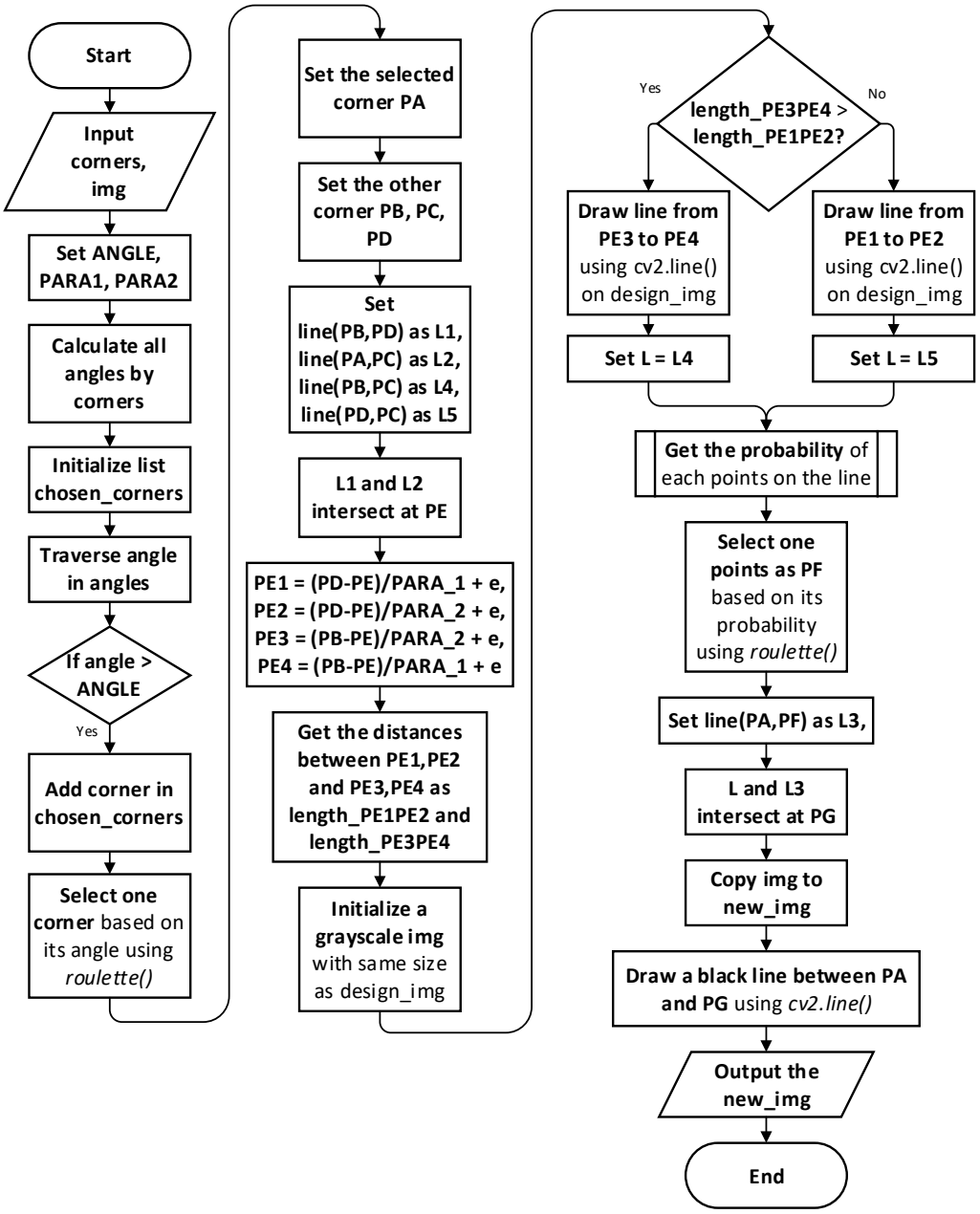

Figure 4.28 The overall flow of the second segmentation module of quadrilateral segmentation

and connect the two points just selected with black lines. Finally, the new image is output. Some of the results are shown in Figure 4.27.

Unlike the first cut rule module, which cuts from edge to edge, the second cut rule module implements a cut from vertex to edge. In short, it is to find the vertex in the quadrilateral that exceeds a specific angle and the appropriate direction to cut. The specific flow is shown in Figure 4.28.

First, input the 8bit grayscale map and the detected angle points, then set three parameters: ANGLE (range is a positive integer from 0 to 180 degrees, usually around 90 degrees), PARA1 (range is a positive integer greater than 2, usually around 5), PARA2 (range is a positive integer greater than two and greater than PARA1, usually around 12), which correspond to the lowest angle and two suitable cutting direction intervals.

Next, the angle of each vertex is calculated, and the vertices with angles exceeding ANGLE are stored in a list named chosen_corners. Then use roulette method to select a vertex in the chosen_corners and name it randomly PA. the rest of the vertices will be named PB, PC, and PD in clockwise order, and the edges of the quadrilateral will be named L1(PB, PD), L2(PA, PC), L3(PB, PC), L4(PD, PC) in turn. Refer to Figure 4.29 for all naming rules in this program.

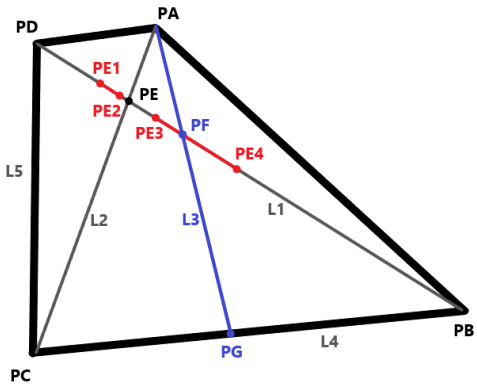

Figure 4.29 The naming rules in the second segmentation module of quadrilateral segmentation

Get the point where L1 intersects L2, named PE. up to this point, calculate the four points between PD and PB by the formula, PE1 = (PD-PE)/PARA_1 + PE, PE2 = (PD-PE)/PARA_2 + PE, PE3 = (PB-PE)/PARA_2 + PE, PE4 = (PB-PE)/PARA_1 + PE. the purpose of finding these. The purpose of the four points is to select the direction of the cut. The purpose of the two parameters PARA1 and PARA2 is to limit the left and right boundaries of the cut direction, respectively. The smaller the difference between PARA1 and PARA2, the more restrictions the cut direction receives. The smaller the range of options available, the lower the diversity, but the less likely to generate an unreasonable design.

Suppose the distance between PE1 and PE2 is smaller than the distance between PE3 and PE4. In that case, it is more suitable to pass the cut line between PE3 and PE4 from the figure because if the cut line is passed between PE1 and PE2, it will make the difference between the two connected component areas in the result too large. Therefore, assuming that the distance between PE1 and PE2 is less than the distance between PE3 and PE4, the program will choose a random point PF between PE3 and PE4. the line formed by connecting PA and PF will intersect at L4, and the intersection point is named PG. PA and PG are the cut lines. The program will use the black line to connect the selected two points. Finally, the new image will be output.

If set ANGLE=90, PARA1=5, PARA2=12, the part of the result is shown in Figure 4.30.

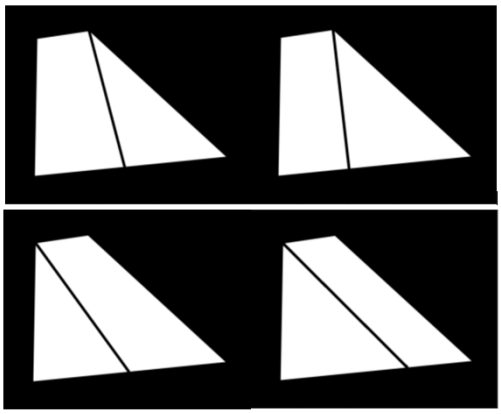

Figure 4.30 Examples of result of the second segmentation module of quadrilateral segmentation

4.4.5 Probability of points and roulette method

This function is mainly for selecting a suitable point on this edge as the endpoint of the cut line after the algorithm has selected one of the sides of the polygon.

There are some architectural design requirements when selecting a point. For example, the selected point should not be too close to the endpoint of an edge (or line segment); the midpoint of an edge (or line segment) is more likely to be used as a reference point for the design compared to other points.

Thus, the basic idea of the function is to draw lines (consisting of pixel points) on a blank grayscale map (filled with zeros). The pixels with non-zero values are the pixels that can be selected, and the value of the pixel is positively correlated with the probability of being selected, i.e., the larger the pixel value, the more likely it is to be selected, and the pixel value is zero, the less likely it is to be selected.

In the actual program, the selected edge is first drawn with a line segment width of 1 and a pixel value of 15. Then a constant parameter K is set, defaulting to 5. Two solid circles are drawn with the center of the circle as the endpoint of the edge, a radius of LENGTH/K, LENGTH is the length of the edge and a pixel value of 0. This allows pixels near the endpoint

to be excluded from are not considered. Finally, find the midpoint of the edge and set the pixel value to 255, which increases the probability that the midpoint will be selected, especially if the length of the edge is not very long.

Finally, traverse this grayscale map and put the coordinates of all points with non-zero pixel values and their pixel values in a list. After completing the traversal, the roulette method is used to select based on the pixel values in the list as the probability magnitude.

The primary purpose of the roulette method is to randomly select a candidate based on the input probability value, using a roulette-like approach.

The roulette method mainly deals with an array where each row holds a candidate for a random selection (coordinates of the sides of the polygon, coordinates of the points, etc.). In contrast, the last column holds the weight of each candidate (representing the probability level).

At runtime, the function adds up the weights in the last column of the array in descending order to get the sum of the weights SUM and then obtains a random number POINTER, ranging from 1 to SUM. Finally, the weights are added up from lowest to highest in order of serial number. When the result of the accumulation exceeds POINTER, the accumulation is terminated, and the serial number of the last candidate added is returned as the result of the selection.

4.4.6 The preliminary segmentation results of the input parcel

Figure 4.31 Example of input image

Although only four types of segmentation have been added for the core so far, two each for triangles and quads, the diversity of results is obvious in practical tests, especially for larger land parcels, since most of the cuts can produce new triangles and quads at the same time.

For example, Figure 4.31 is a 1000×800 8bit grayscale image that contains a quadrilateral (trapezoid). If a cut is made for this image, many different styles of cuts can be obtained, as in Figure 4.32.

The number of quadrilaterals and triangles generated in these cuts vary in size, but they all meet the design requirements and aesthetic requirements of the Courtyard-style building.

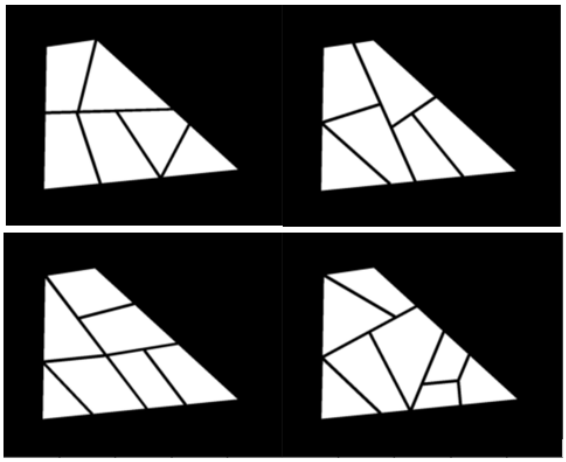

Figure 4.32 Examples of results of segmentation

However, for the real-life environment of the land parcel on which the Courtyard-style building is located, this grayscale map is slightly larger if converted at a scale of 10 pixels = 1 meter. In other words, it is not often that the same land parcel size is used exclusively for a Courtyard-style building in the real world. Therefore, replacing the land parcel with one closer to the same size in the real world was tested, and the results are shown in Figure 4.33.

In this test, the algorithm segments a grayscale map containing three separate parcels at the same time, which contains triangles and trapezoids. The resolution of the grayscale map remains 1000×800, a scale that is more realistic from an area point of view. Moreover, the generated results are still both diverse and reasonable.

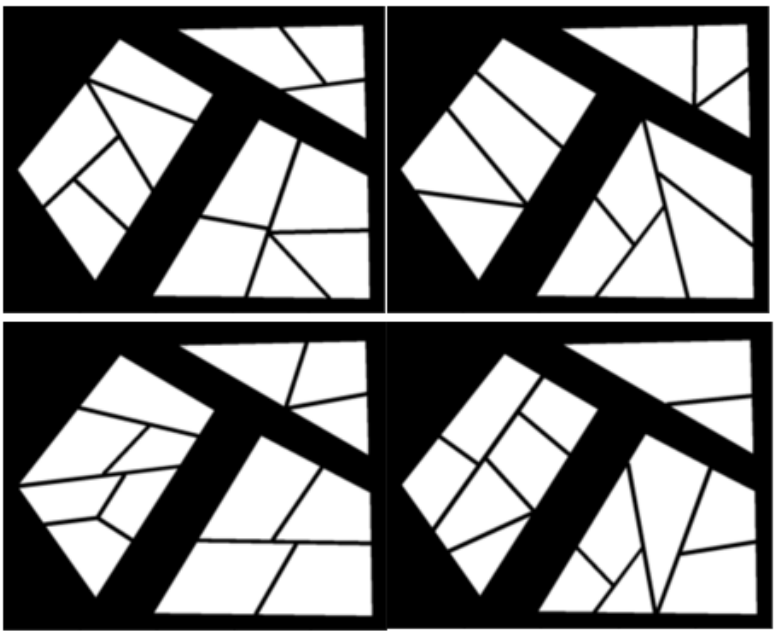

Figure 4.33 Examples of results of segmentation of multiple parcels

4.5 Visualization and processing of design results

Post-processing mainly aims to improve the aesthetics and reference of the generated results. Specifically, it is to add details to the segmented results and further process them into visual effects similar to architectural texture maps or for 3D visualization.

4.5.1 Visualization of architectural texture map

The building texture map is an excellent tool to reflect the building's morphological characteristics and appearance design. The reference ability of the results generated by this algorithm can be significantly improved if the basic texture map of the building to be designed is automatically generated.

Considering the apparent morphological relationship between a courtyard building and the land parcel on which it is located. For example, an aerial photograph of a courtyard building in Florence, Figure 4.34(a), can be approximated as Figure 4.34(b). For such building complexes with typical enclosing nature, their myograms can be approximated by multiple graphical erosion operations and addition and subtraction operations on their land parcels.

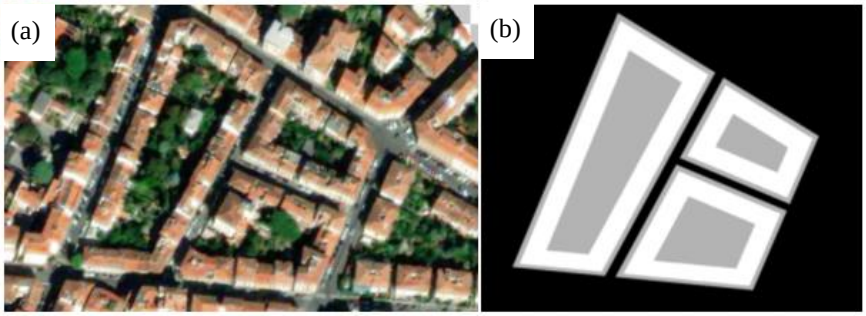

Figure 4.34 Courtyard-style building in Florence: (a) aerial image, (b) architectural texture

Therefore, generating the building texture map can be obtained in simple steps.

First, a grayscale map containing the completed cut land parcels is obtained, and an erosion operation is performed (kernel_size is 40 and iteration is 1). This is to indent the cut parcel to a real-world distance of 4 meters as a sidewalk. Save the output preliminary gray map result 1.

Perform an erosion operation on the preliminary gray map result (kernel_size is 70, iteration is 1). The purpose is to obtain the center hall part of the courtyard building with a depth of 7 meters. Save the output preliminary grayscale result 2.

Subtract the preliminary gray map result 1 from the preliminary gray map result 2 to get the texture of the courtyard building and save the output gray map result 3.

Create a new grayscale map of the same size, filled with zeros, and name it dst. Then the final effect dst can be obtained by adding the original grayscale map, and the grayscale map result 3 in a certain ratio.

Some examples of its final result are shown in Figure 4.35.

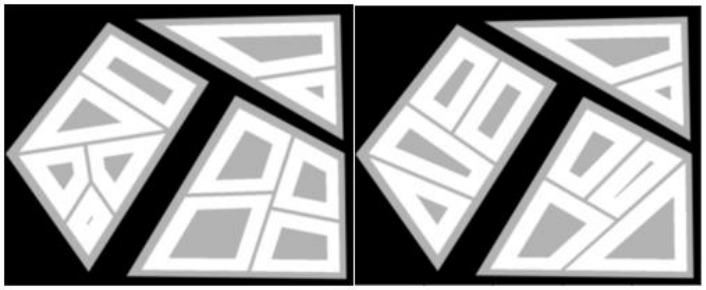

Figure 4.35 Examples of generating architectural texture

4.5.2 Processing of architectural texture map

In architectural design, buildings sometimes have overly sharp angles at the apex, as in Figure 4.36(a). Some architects may consider this design more visually striking and stylistic, while others may avoid this situation in their designs or remove the overly sharp edges. As in Figure 4.36(b), the original sharp angle has been removed and turned into two obtuse angles.

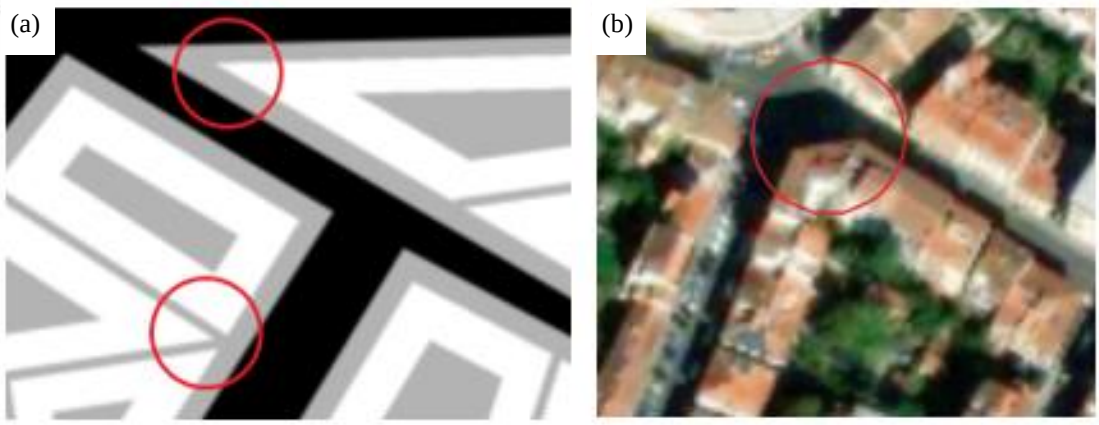

Figure 4.36 Sharp angles: (a) in architectural texture generated, (b) removed in real world

This algorithm also needs to simulate this situation and detect the sharp angles in the architectural texture map. For corners that are too sharp, they are directly removed; for obtuse corners, they are not processed; for corners in between, they are removed according to a certain probability and the size of their angle (i.e., the smaller the angle, the more likely they are to be removed).

As for the method of excision, there are two relatively in line with the principles of architectural design.

1. making a perpendicular line to cut an acute angle to get an obtuse angle and a right angle

2. isosceles triangle to cut the acute angle to get two obtuse angles of equal angle

For a single connected figure, the specific process of cutting the acute angle is shown in Figure 4.37.

First, enter an 8-bit grayscale map containing a single connected map and copy an identical grayscale map. Find the vertices and angles in it and initialize a list.

Iterate through the angles of all the vertices, and when there is a vertex with an angle less than 30°, add the vertex to the list directly; when the vertex angle is between 30° and 90°, get a random integer between 30 and 90, and when the integer is greater than the vertex angle, add the vertex to the list.

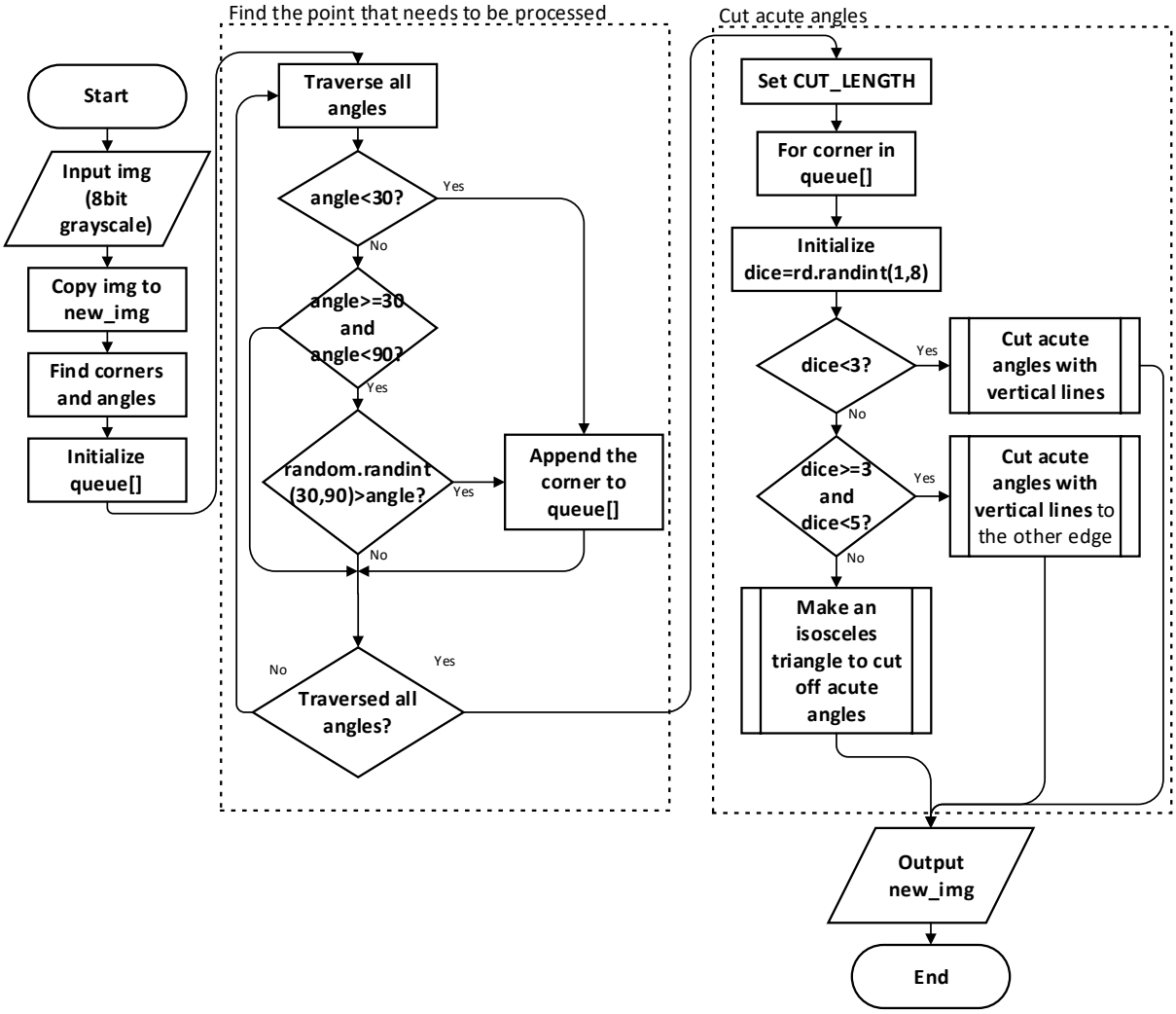

Figure 4.37 The overall flow of de-sharpening in single connected component

Set a parameter CUT_LENGTH, determining how big the "scar" will be when the acute angle is cut off. Then iterate through the list of vertices that need to be processed. For each vertex, get a random integer dice between 1 and 8. When dice is less than 3, make a

perpendicular line to cut the acute angle; when dice is between 3 and 5, make a line perpendicular to the other edge to cut the acute angle; otherwise, make an isosceles triangle to cut the acute angle.

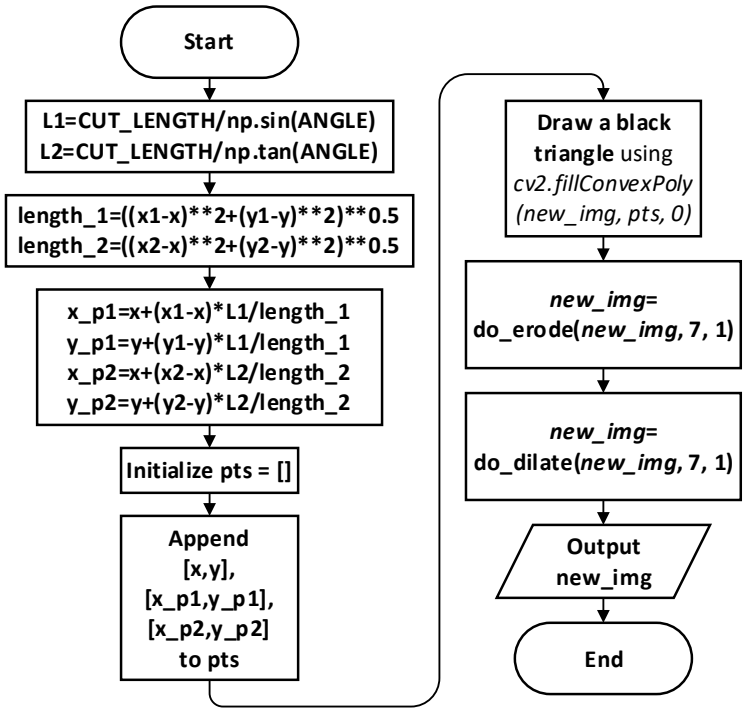

Figure 4.38 The overall flow of cutting sharp corners by making vertical lines perpendicular to an edge

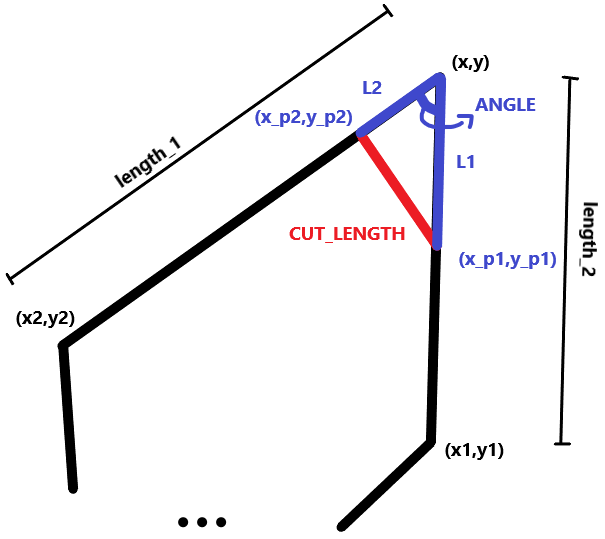

Figure 4.39 The naming rules in cutting sharp corners by making vertical lines perpendicular to an edge

Whether the dice is less than 3 or between 3 and 5, the way to cut is to make a perpendicular line perpendicular to one edge to cut the acute angle. The specific process is shown in Figure 4.38, 4.39.

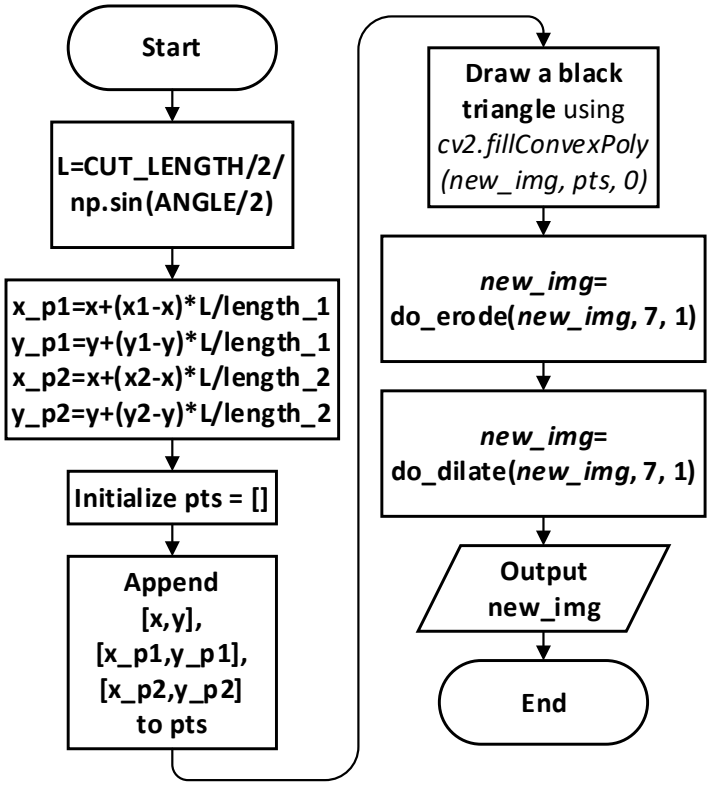

Figure 4.40 The overall flow of cutting sharp corners by make an isosceles triangle

In short, find a vertical line perpendicular to an edge and of length CUT_LENGTH. Along this vertical line and the acute angle to be cut, draw a right triangle in black. Since this triangle is black, it will cover the white area of the original image. The image is then subjected to an open operation to prevent incomplete coverage.

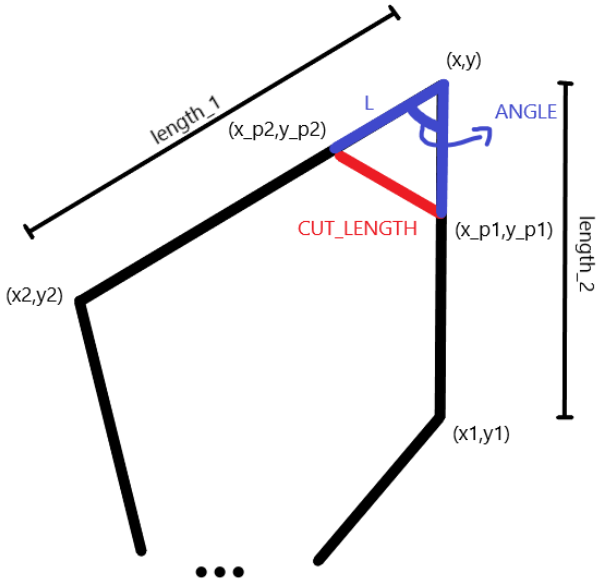

Figure 4.41 The naming rules in cutting sharp corners by make an isosceles triangle

When dice are greater than or equal to 5, the program will do an isosceles triangle to excise the acute angle. The specific flow is shown in Figure 4.40, 4.41.

In short, find an isosceles triangle with a base length of CUT_LENGTH and draw such a black isosceles triangle to cover the white area of the original image. An open operation is needed to perform on the image at the end to prevent incomplete coverage.

Some examples of a de-sharpened architectural texture map are shown in Figure 4.42.

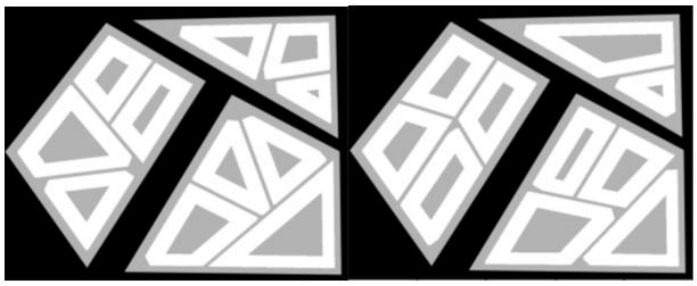

Figure 4.42 Examples of a de-sharpened architectural texture

It is evident that after the de-sharpening, the automatically generated architectural texture map is more natural and realistic in general and more in line with the actual architectural design. The sharpening makes the results much more referable.

4.5.3 3D visualization of architectural texture map

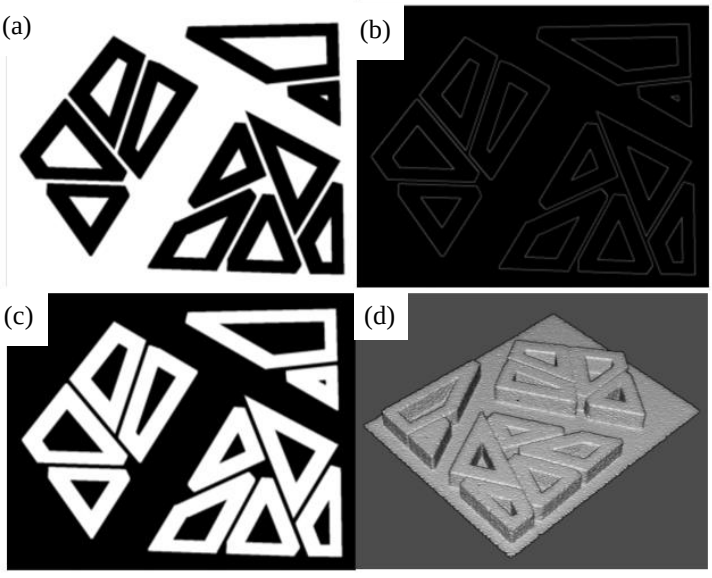

Figure 4.43 Process of 3D visualization: (a) background minus architectural texture, (b) the contour of architectural texture, (c) architectural texture, (d) point cloud

So far, the program has not added height information to the generated building texture maps. If the main function is a residential area, the buildings of the Courtyard type are the

same height as each other. If the standard is a residential area, the point cloud data can be generated using the existing building texture map to realize 3D visualization.

For generating the point cloud data, we mainly use the way of accumulating the grayscale map layer by layer. We read the plane coordinates of pixels with a pixel value of 255 in each layer of the grayscale map as x, y-axis coordinates, and the z-axis coordinates are set as the current layer to obtain the 3D coordinates of each point. Specifically, the grayscale map of the first layer is the result of a grayscale map filled with 255 minus the architectural texture, as in Figure 4.43(a). The second layer to the 79th layer is a contour map of the building texture, as in Figure 4.43(b). Layer 80 shows the building texture (without parcel information), as in Figure 4.43(c). In order to prevent the performance from being too low due to the excessive number of points, when collecting the point cloud data and extracting the coordinates of the points from the grayscale map, the coordinates are extracted every 5 to 9 pixels with the value of 255, depending on the number of layers in which the grayscale map is located. In this way, the data size of the point cloud can be reduced significantly, and the operation efficiency can be improved.

The final point cloud data obtained by matplotlib and pyvista 3D visualization library is shown in Figure 4.43(d).

However, even though the number of point clouds was reduced as much as possible, the time required to generate point cloud data and 3-dimensional effects was still much longer than expected. When generating the design sample as in Figure 4.43(c), generating a building texture map only takes 5-15 seconds, while generating the point cloud data often takes more than 50 seconds, reaching more than eight times the time required to generate the texture map. Due to the severe impact on operational efficiency, this 3D visualization process was temporarily removed in the final experiment. Only the building texture map was generated and taken as the final result.

5. Experiment and analysis of OpenCV-based generation algorithm

This chapter aims to conduct experiments on the automatic generation of ensemble layout schemes using typically shaped land parcels of different sizes and to verify the validity and reliability of the algorithm.

5.1 System environment

Operating System: Windows 10 Home 21H2
Processor: Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz
Installed RAM: 32.0 GB
GPU: NVIDIA Geforce RTX 3080 × 1
IDE: Jupyter Notebook 6.3.0
Development Language: Python 3.8.8
Image processing framework: OpenCV4.5.5

5.2 Experimental procedure

5.2.1 Layout generation for triangular parcel

Triangular parcels with areas ranging from 240.91 m² to 4230.86 m² were used to test the algorithm's effectiveness. The information input is in the form of grayscale maps containing triangular parcel information, as shown in Figure 5.1. A total of 20 grayscale maps with dimensions of 500×500, 550×550, 600×600, 650×650 ... 1400×1400, 1450×1450, and 1500×1500 were input to characterize the triangular parcels with areas ranging from 240.91 m² to 4230.86m², respectively.

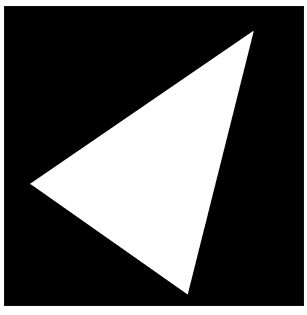

Figure 5.1 Triangular land parcel

As mentioned earlier, the experimental goal is to perform an automatic architectural design with the theme of courtyard-style buildings on multiple grayscale maps containing information about a single triangular parcel to obtain a de-sharpened architectural texture map (grayscale map) containing information of the original parcel. Each grayscale map is run seven

times, and seven rounds of layout schemes are generated automatically. At the same time, the average running time and the standard deviation of the seven runs with the design under different land parcel sizes are counted.

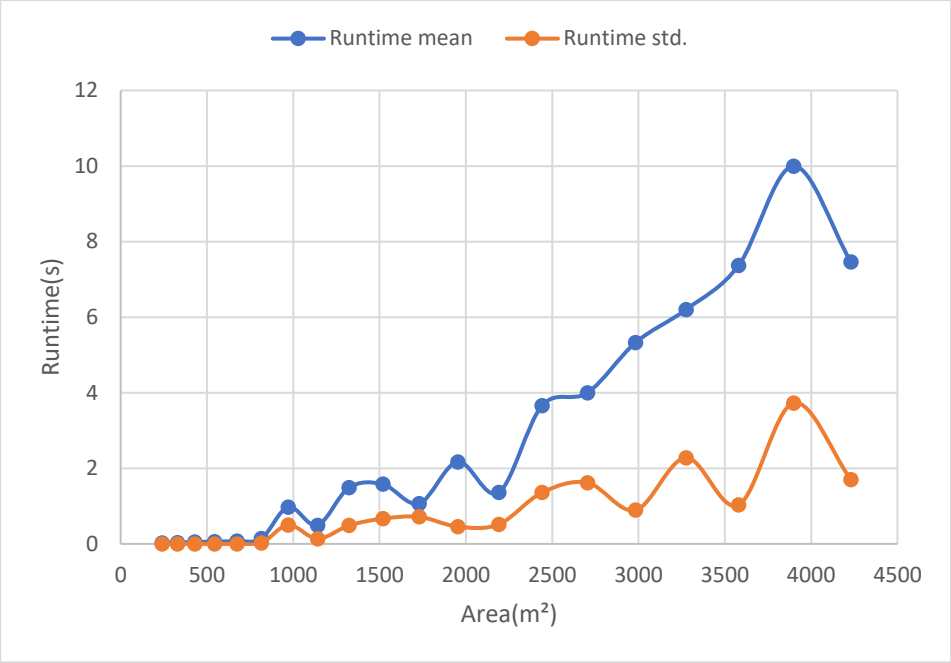

Figure 5.2 Efficiency test results for triangular land parcel

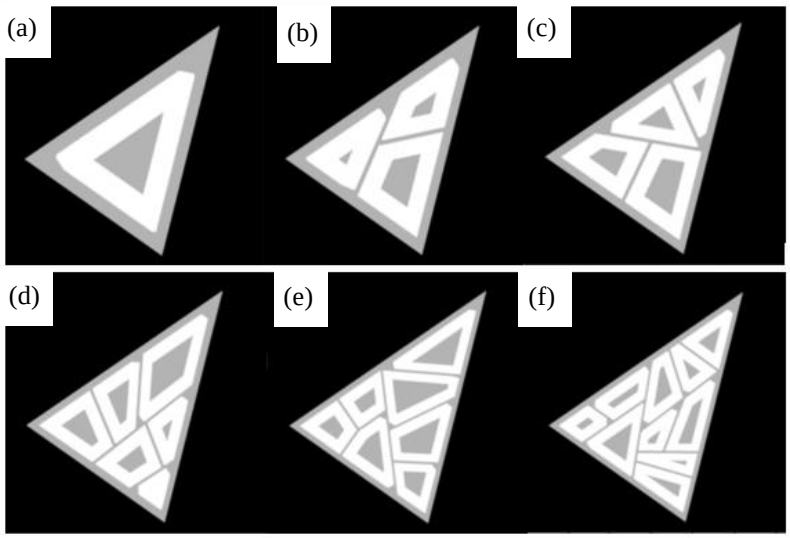

Figure 5.3 Some generated results for triangular land parcel:(a) 1140.55m²; (b) 1323.31m²; (c) 1953.25m²; (d) 2983.54m²; (e) 4230.86m²; (f) 4565.26m²

The results of the first set of performance tests are shown in Figure 5.2, as the area covered by the land parcels increases, because the generated graphs become more and more complex, as in Figure 5.3. Generally speaking, in reality, most of the land parcels are similar

in size to Figure 5.3 (c), as for Figure 5.3 (d) (e) (f) here only for performance testing, the reality does not exist that requires the design of a co-located building for a land parcel of such size completely. As can be seen in Figure 5.2, the average running time keeps rising, along with the standard deviation of the running time, indicating that the computation is getting larger and the running efficiency is becoming more and more unstable. In the design process, there may be several failed cuts that lead to the need for re-cutting, which is especially common in the case of large land parcel area and large number of operations, and will significantly affect the stability of the program running efficiency. A point cloud model is generated for one of the design results with a land parcel area of 1493m², as shown in Figure 5.4.

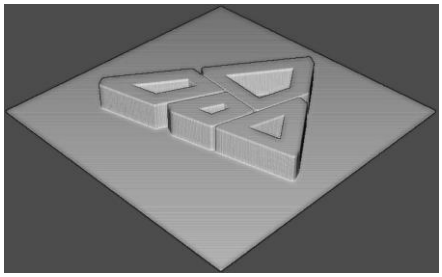

Figure 5.4 Point cloud model for triangular land parcel

5.2.2 Layout generation for quadrilateral parcel

Quadrilateral parcels with areas ranging from 1229.19 m² to 4936.18 m² were used to test the effect of the algorithm. The information input is in the form of grayscale maps containing information about the quadrilateral parcels, as shown in Figure 5.5. A total of 11 grayscale maps with dimensions of 500×500, 550×550, 600×600, 650×650 ... 950×950 are input to characterize quadrilateral parcels with areas ranging from 1229.19 m² to 4936.18 m², respectively.

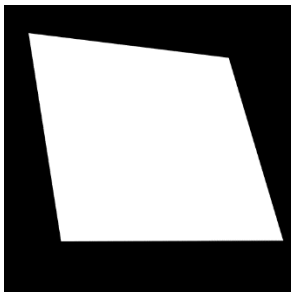

Figure 5.5 Quadrilateral land parcel

As mentioned earlier, the objective of the experiment is to design an automatic architectural design with a courtyard building theme for multiple grayscale maps containing information about a single quadrangular land parcel in order to obtain a de-sharpened architectural texture map (grayscale map) containing information about the original land parcel. Each grayscale map is run seven times, and seven rounds of layout schemes are

generated automatically. At the same time, the average running time and the standard deviation of the seven runs with the design under different land parcel sizes are counted.

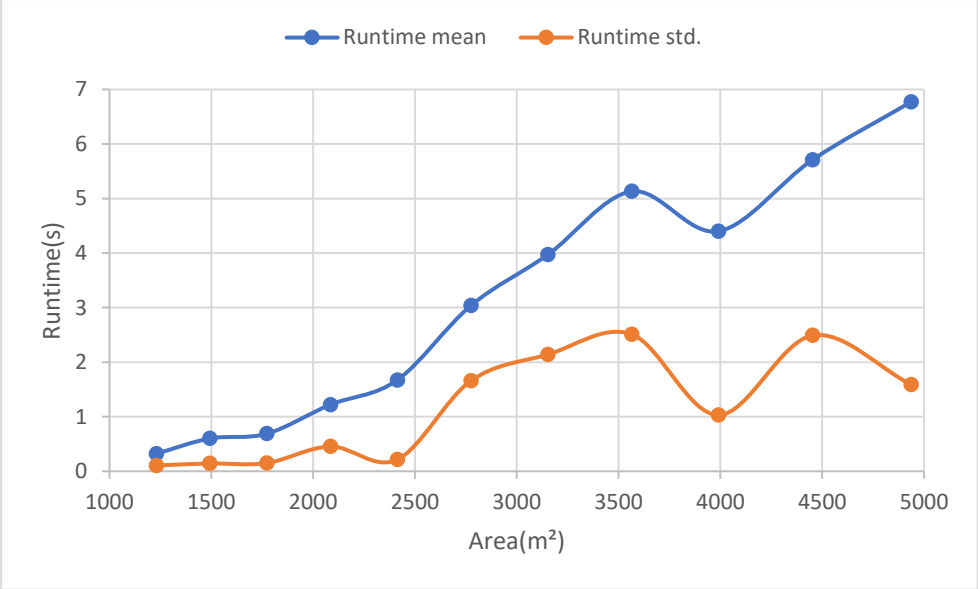

Figure 5.6 Efficiency test results for quadrilateral land parcel

The second set of results is shown in Figure 5.6, and as in the first set, the average run time and the standard deviation of the run time keep increasing as the area covered by the land parcels increases. Some of the results are shown in Figure 5.7. A point cloud model was generated for one of the design results with a land parcel area of 1493m², as shown in Figure 5.8.

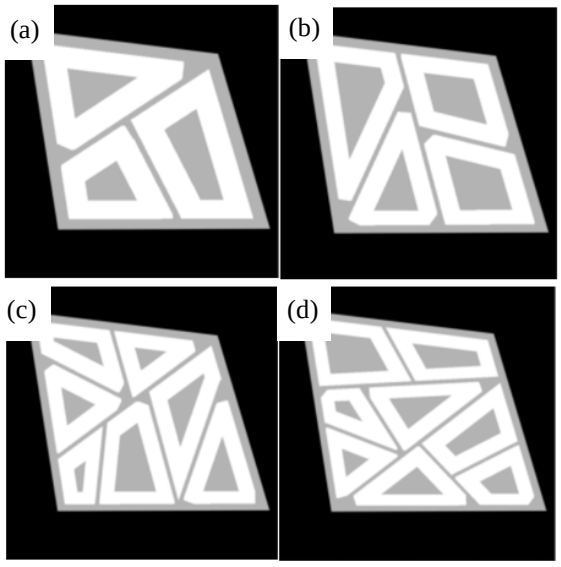

Figure 5.7 Some generated results for quadrilateral land parcel: (a) 1229.19m²; (b) 2084.64m²; (c) 3152.42m²; (d) 3564.63m²

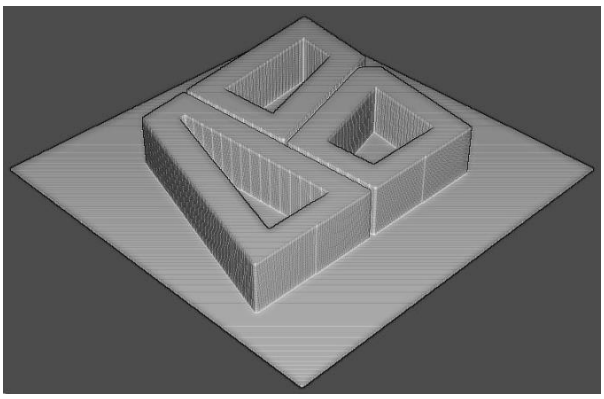

Figure 5.8 Point cloud model for quadrilateral land parcel

5.2.3 Layout generation for combined parcel

First set of experiments:

A combination of quadrilateral and triangular parcels with areas ranging from 971.88 m² to 3908.63 m² was used to test the algorithm's effectiveness. The information input is in the form of grayscale maps containing information about one quadrilateral and one triangular parcel, as shown in Figure 5.9. 11 grayscale maps with dimensions of 500×500, 550×550, 600×600, 650×650 ... 950×950, 1000×1000, and 1050×1050 were input to characterize the combined quadrilateral and triangular parcels with areas ranging from 971.88 m² to 3908.63 m², respectively.

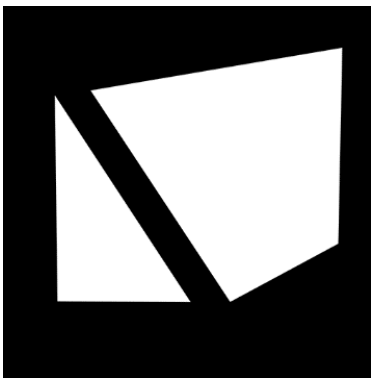

Figure 5.9 Combined land parcel

As mentioned earlier, the objective of the experiment is to design an automatic architectural design with the theme of courtyard buildings on multiple grayscale maps containing information on the combined quadrilateral and triangular parcels to obtain a de-sharpened architectural texture map (grayscale map) containing information of the original parcels. Each grayscale map is run seven times, and seven rounds of layout schemes are automatically generated. At the same time, the average running time and the standard deviation of the seven runs with the design under different land parcel sizes are counted.

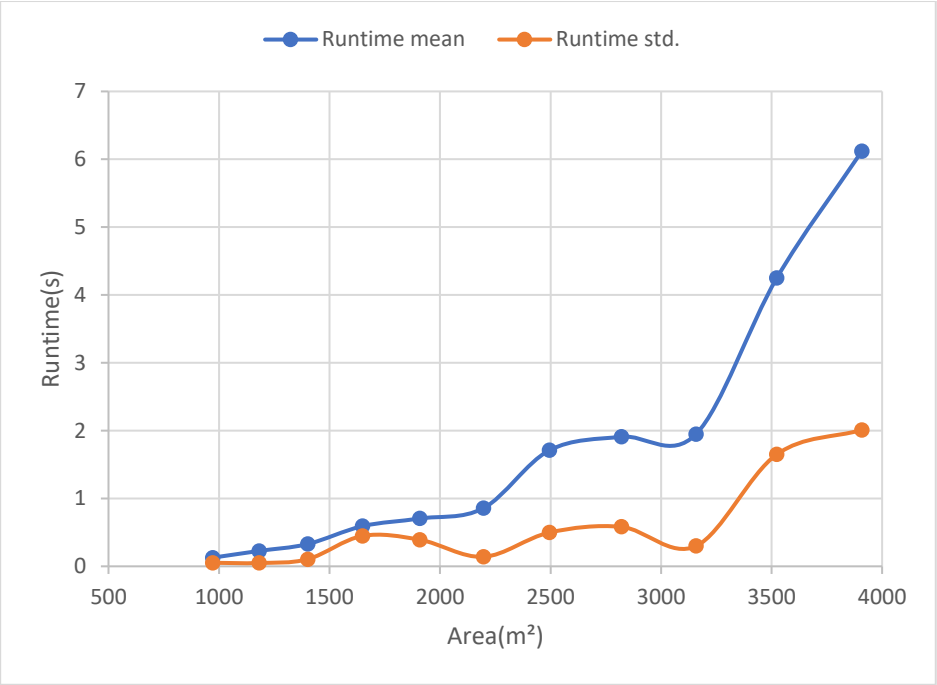

Figure 5.10 Efficiency test results for combined land parcel

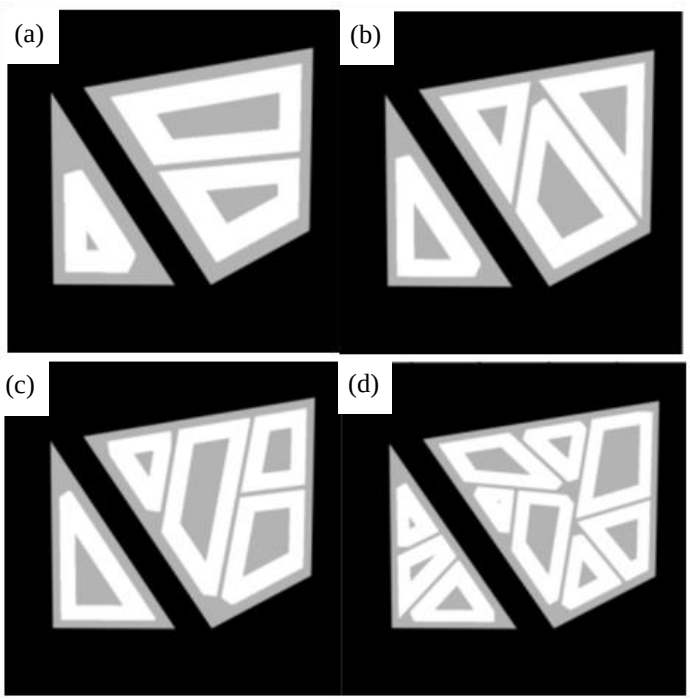

Figure 5.11 Some generated results for combined land parcel: (a) 1180.70m²; (b) 1908.90m²; (c) 2494.76m²; (d) 3908.63m²

Since the experiments for the ensemble contain two parcels, resulting in a significant reduction in the area of a single parcel for the same total area, the running time is also much reduced compared to the experimental group for a single parcel, as shown in Figure 5.10. It

can be seen that increasing the number of parcels does not increase the time complexity as significantly as increasing the area of a single parcel when the area of a single parcel is not very large. Since, in the real world, there does not exist such a single large parcel that needs to be built in a co-located building, the algorithm can handle more parcels at the same time when dealing with problems in reality. The results of design are shown in Figure 5.11. A point cloud model was generated for one of the design results with a land parcel area of 2494.76m², as shown in Figure 5.12.

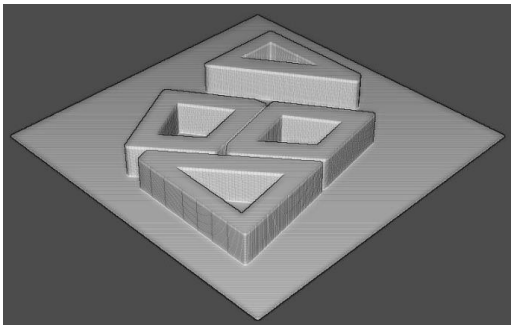

Figure 5.12 Point cloud model for combined land parcel

Second set of experiments:

A combination of quadrilateral and triangular parcels with areas ranging from 1210.91 to 2734.09 m² was used to test the effectiveness of the algorithm. The information input is in the form of a grayscale map containing the information of two quadrilateral and one combined triangle parcel, as shown in Figure 5.13. A total of six grayscale maps with dimensions of 500×500, 550×550, 600×600, 650×650,700×700, 750×750, and 800×800 were input to characterize the combined quadrilateral and triangular parcels with areas ranging from 1210.91 to 2734.09 m², respectively.

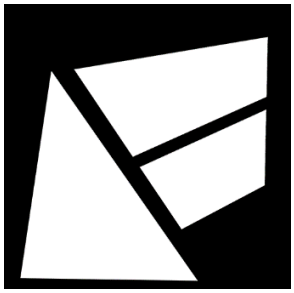

Figure 5.13 Combined land parcel

As mentioned earlier, the objective of the experiment is to design an automatic architectural design with the theme of courtyard buildings on multiple grayscale maps containing information on the combined quadrilateral and triangular parcels in order to obtain a de-sharpened architectural texture map (grayscale map) containing information of the original parcels. Each grayscale map is run seven times, and seven rounds of layout schemes are generated automatically. At the same time, the average running time and the standard

deviation of the seven runs with the design under different land parcel sizes are counted. The results are shown in Figure 5.14, 5.15, 5.16.

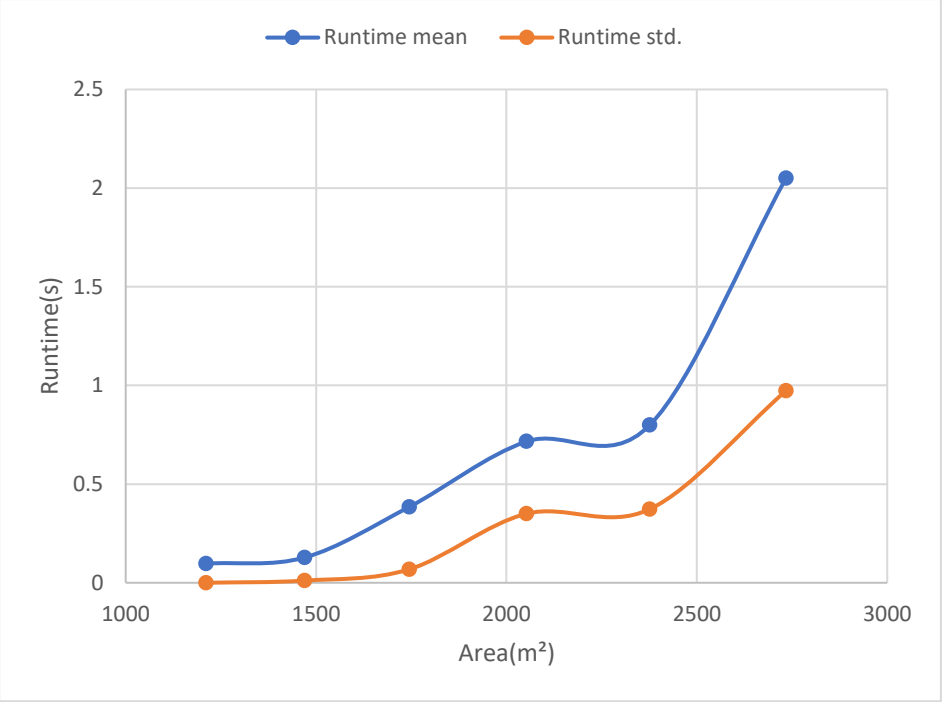

Figure 5.14 Efficiency test results for combined land parcel

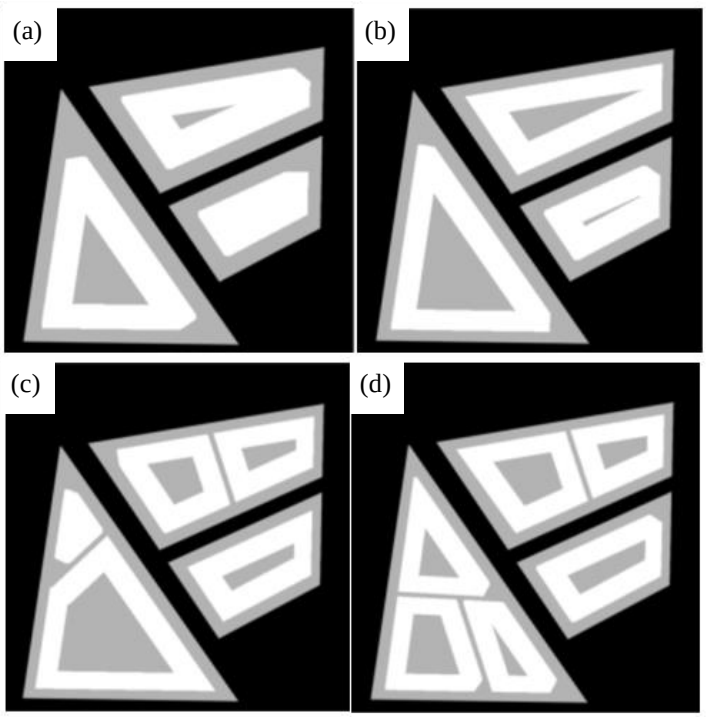

Figure 5.15 Some generated results for combined land parcel: (a) 1210.91m²; (b) 1744.49m²; (c) 2376.20m²; (d) 2734.09m²

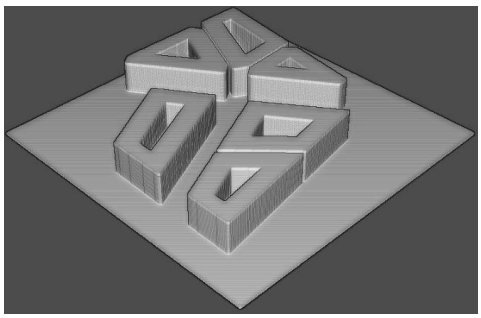

Figure 5.16 Point cloud model for combined land parcel

5.2.4 Layout generation for large combined land parcels

The algorithm was tested using a combination of land parcels containing a large number of polygons. A total of three grayscale maps with dimensions of 1200×1200 were input, characterizing quadrilateral parcels with areas of 7753.41m², 9384.52m² and 11224.37m² .

The objective of the experiment is to perform an automatic architectural design with the theme of courtyard-style buildings on multiple grayscale maps containing information of several polygonal parcels in order to obtain a de-sharpened architectural texture map (grayscale map) and a point cloud model containing information of the original parcel. Each grayscale map is run twice to show two different designs of building texture and point cloud models.

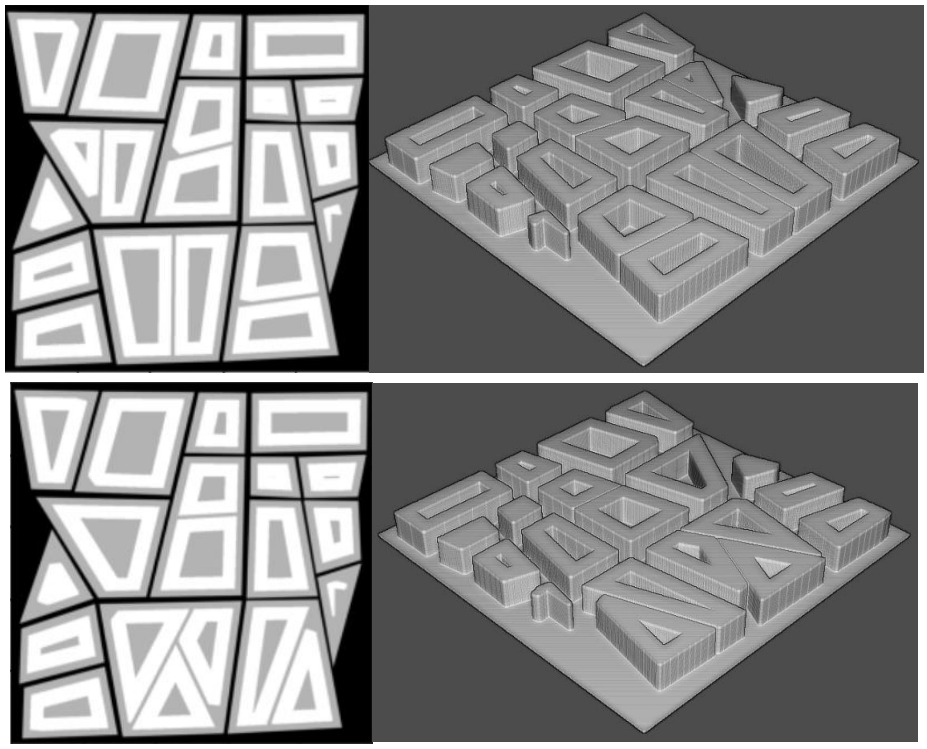

Figure 5.17 Generated results for combined land parcel (7753.41m²)

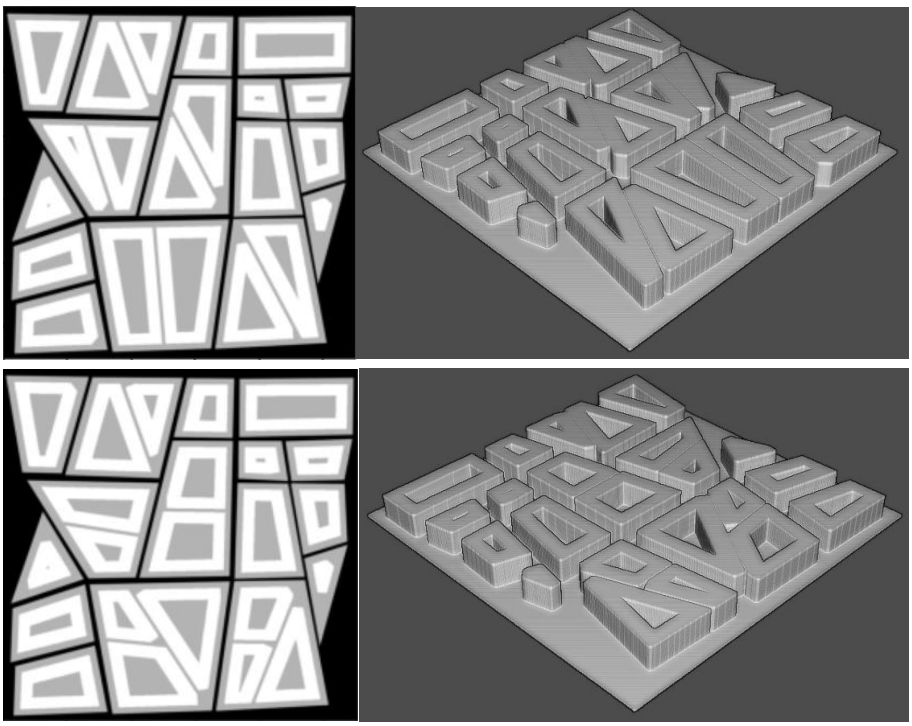

Figure 5.18 Generated results for combined land parcel (9384.52m²)

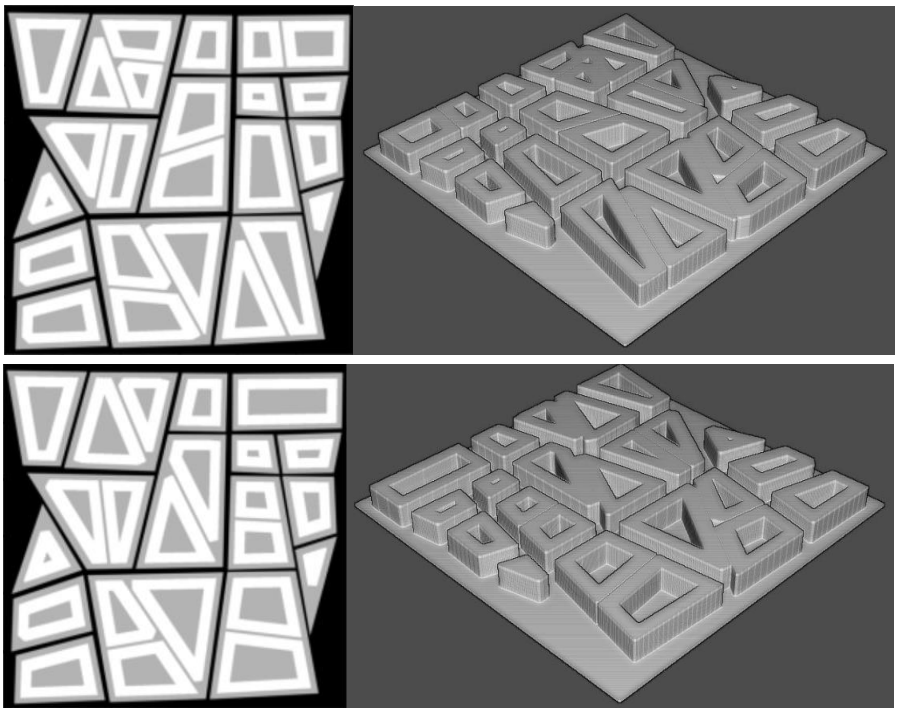

Figure 5.19 Generated results for combined land parcel (11224.37m²)

As shown in Figures 5.17, 5.18, and 5.19 in the input parcel information, if it contains many parcels, the algorithm can still generate diverse and decent design solutions stably and quickly.

5.2.5 Experiments on the efficiency of the generation algorithm

A quadrilateral parcel with an area of 3989.55m² was used to test the algorithm's efficiency. The information input is in the form of a grayscale map containing information about a quadrangular land parcel, as in Figure 5.20. A total of 1 grayscale map with dimensions of 900×900 is input to characterize a quadrangular land parcel with an area of 3989.55m².

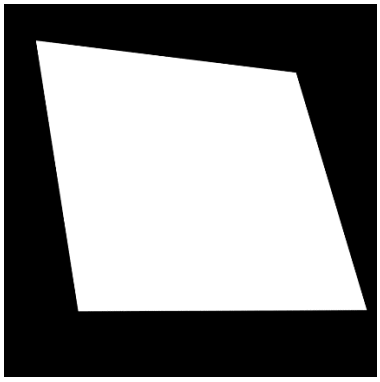

Figure 5.20 Quadrilateral land parcel (3989.55m²)

As mentioned earlier, the objective of the experiment is to perform an automatic architectural design with a Courtyard-style building theme on a grayscale map containing information about a single quadrangular land parcel to obtain a de-sharpened architectural texture map (grayscale map) with information about the original land parcel. Each gray map was run seven times. The average running time of each step (pre-processing, recognition & segmentation, texture map generation, 3D visualization) was also calculated for each of the seven design runs.

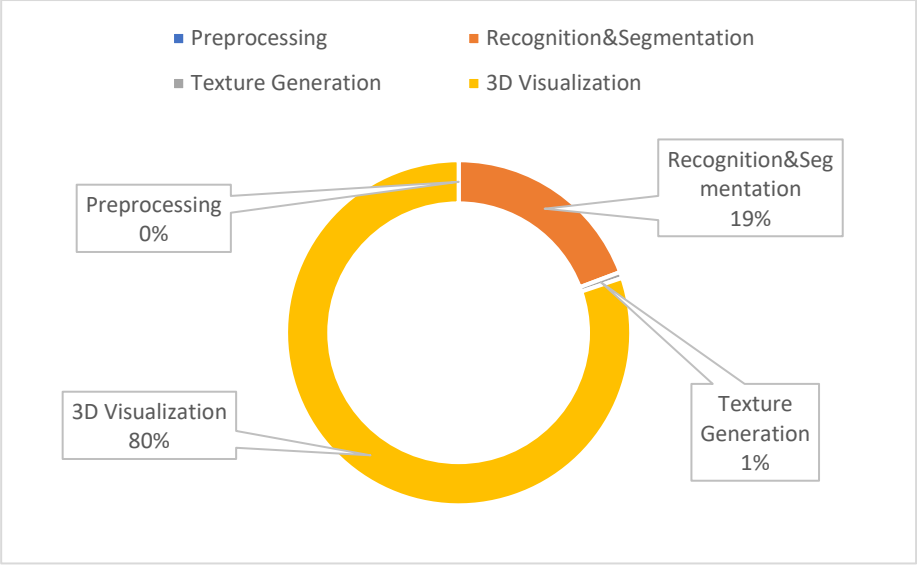

Figure 5.21 Percentage of running time in each phase

In the experiments, the average time for preprocessing was 0.006s, the average time for recognition & segmentation was 5.14s, the average time for generating the texture map was

0.154s, and the average time for 3D visualization was 21.4s, which accounted for 0%, 19%, 1%, and 20% of the total time, respectively (Figure 5.21). Among them, 3D visualization and recognition & segmentation took up most of the running time. Especially for 3D visualization, the average time spent generating point cloud data is 4.3 s, while the average time spent generating a 3D model from point cloud data is 17.1 s. Obtaining a 3D model from point cloud data is not the best means of 3D visualization. Moreover, there is still much room for efficiency improvement in the recognition & segmentation process.

5.3 Improvement plan

The algorithm uses the algorithm goodFeaturesToTrack from the OpenCV library to perform the corner point detection task in the recognition phase of recognizing the vertices of polygons. This function will not recognize the correct corner points in specific cases. It is difficult to detect whether the recognized corner points are correct. The coordinates of the corner points are the basis of the whole program, so the problem can lead to not generating the building texture map not or adequately removing the sharp corners properly. If this problem is to be solved, it is needed that the corner point detection algorithm can be further improved to enhance the stability of the program operation.

In addition, since Courtyard-style buildings are not all fully enclosed structures, many of them are semi-enclosed types. Therefore, if the distribution pattern of the gaps in the Courtyard -style buildings can be further explored and converted into the programming language to reflect in the generated design system, the diversity, authenticity, reference ability, and value of the results can be significantly improved.

Finally, the result generated by this algorithm is a building texture map, which already has all the information and conditions to generate a 3D structure map. If the way of generating point cloud data can be replaced and polygon modeling is used to define different shapes and surfaces using X, Y, and Z coordinates, and then the different surfaces are combined to form the shape of the building, the 3D visualization can be done with much higher efficiency. It is even possible to add different heights to the building to improve the aesthetics and realism of the result.

Conclusions

In this thesis, the Courtyard-style building, a building with distinctive style and morphological characteristics, is select as the research object to construct an automatic computer-generated design system for the layout of this kind of historical district building.

At the algorithm level of exploration, the results of the image conversion algorithm based on GAN networks do not meet the expected standards. There are too many noise points in the model and the buildings' prediction map of aerial images. Although the basic outline and shape can be seen, the overall form is fragmented or partially distorted, failing to achieve the effect of improving the reference ability. In the prediction map of the model to the building texture map, the general trend and orientation of the design can be seen, but it is not easy to obtain information with a strong structure. It is necessary to post-process its prediction map in depth to obtain the building structure for further study, but the complexity is too high. In conclusion, the GAN network-based model is of limited help in constructing the overall algorithm in this study.

Focusing on the automatic design algorithm based on OpenCV, a complete design generation system for Courtyard-style building layout is implemented by a Python program. After profoundly analyzing the characteristics of the architectural design level of the historic district buildings exemplified by the Courtyard-style buildings, the whole design process of designing the layout of the Courtyard-style building complex can be divided into the division of the land parcel where the Courtyard-style buildings need to be built, and the adjustment and visualization of the architectural form of each Courtyard-style building on the divided land parcel.

Therefore, the algorithm automates the design process by translating the vague architectural design language into a procedural computer language while ensuring its integrity and consistency as much as possible. Starting with recognizing detailed geometric morphological features of the land parcel, the algorithm translates the visualized image information into a large number of digitized graphical features. The algorithm enables the segmentation of arbitrarily shaped parcels using these features after several adjustments and process design. It has several different, probabilistic-based segmentation methods that conform to the architectural design, intending to obtain mixed results. After completing a segmentation, the algorithm can segment the segmented results again. The algorithm can segment a parcel of any size using a circular structure until the required result is obtained. The algorithm also has a validation mechanism to ensure the reasonableness of the segmented results. The segmentation results that do not meet the required indexes are asked to be re-segmented until a good result meets the requirements.

For the selection and setting of metrics, this study also conducts experiments and analyses to pursue a balance in program efficiency and rationality. After completing the segmentation, the algorithm generates a building texture map on the segmented parcel as the design result. For the building edges in the building texture map, the algorithm will modify and improve them based on the architectural design principles and probability, significantly improving the design results' authenticity and reference. Finally, the algorithm can use the generated building texture map to automatically generate point cloud data and obtain the 3D visualization results of the building texture map using pyvista. In this study, the procedure's efficiency for generating the design and visualization process is experimented with and analyzed. In conclusion, the algorithm can quickly process any shape, area, and the number of parcels and quickly generate diverse and reasonable basic architectural texture maps of ensemble buildings.

References

[1] He J, Song C. Evaluation of pedestrian winds in urban area by numerical approach[J]. Journal of Wind Engineering & Industrial Aerodynamics, 1999, 81(1-3):295-309.

[2] Overby, Jens, et al. "Automatic 3D building reconstruction from airborne laser scanning and cadastral data using Hough transform." International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences 34.01 (2004).

[3] 孙钊，吴志华，熊伟．基于三维数字技术的城市设计研究与应用[J]. 城市规划学刊. 7(2009):239-241.

[4] 田静，吴红梅，王子启．三维技术在数字城市辅助规划中的应用[J]．北京测绘，2014(1):3.

[5] 苏剑鸣，梅小妹．基于功能与环境关系评价的生成式计算机辅助建筑设计方法研究[J]. 南方建筑 1(2011):239-241.

[6] 李飚．建筑生成设计——基于复杂系统的建筑设计计算机生成方法研究[M]. 南京:东南大学出版社，2012.

[7] 刘宇波，林文强，邓巧明，等．基于神经网络的建筑形态生成探索——以柏林自由大学为例[C]// 2019 全国建筑院系建筑数字技术教学与研究学术研讨会.

[8] Martin J. Algorithmic beauty of buildings methods for procedural building generation[J]. Computer Science Honors Theses.4(2005):200-223.

[9] 张佳石．基于多智能体系统与整数规划算法的建筑形体与空间生成探索[D]. 东南大学.

[10] Merrell P, Schkufza E, Koltun V. Computer-generated residential building layouts[J]. ACM Transactions on Graphics (TOG), 2010, 29(6):1-12.

[11] Wang X Y, Yang Y, Zhang K. Customization and generation of floor plans based on graph transformations[J]. Automation in Construction,2018,94: 405-416.

[12] Arvin S A, House D H. Modeling architectural design objectives in physically based space planning[J]. Automation in Construction, 2002, 11(2):213-225.

[13] Sun Y, Qiu M, Taplin J. Achieving residential connectivity and density goals with computer-generated plans in a greenfield area[J]. Environment & Planning B Planning & Design, 2014, 41(3):430-449.

[14] Krawczyk, Robert J. Architectural interpretation of cellular automata[J]. Generative Art (2002).

[15] 何宛余，杨小荻．人工智能设计, 从研究到实践[J]．时代建筑，2018(1):38-43.

[16] 高婉君，毛超，刘贵文．城市居住区中建筑布局自动生成方法与方案评价研究[J]．城市住宅，2020, 27(3):85-88.

[17] Baker B S. Orthogonal packing in two dimensions[J]. SIAM Journal on Computing, 1980, 9：846-855.

[18] 季云竹．基于模式语言的建筑空间生成算法探索——以徽州民居为例[D]. 东南大学, 2016.

[19] 许杰青．浅谈中国生态建筑的模式语言的构建[J]．山西建筑，2008, 34(36):2.

[20] 石川．肌理在建筑立面设计中的应用研究[D]．合肥工业大学，2012.

[21] 何依，邓巍．历史街区建筑肌理的原型与类型研究[J]．城市规划，2014(8):6.

[22] 蔡文彬. 精神和文化的空间——古代中国合院式建筑的庭院[J]. 艺术与设计：理论版, 2008(12):3.
[23] Ayers, Andrew. The architecture of Paris: an architectural guide. Edition Axel Menges, 2004.
[24] 金华, 陆伟, 卜默默, 等. 历史街区保护与更新中建筑文化旅游的可开发性探讨——以大连东关街为例[J]. 城市建筑, 2019, 16(19):6.
[25] 许承程. 基于共生理论的大连东关街街区更新研究[D]. 沈阳建筑大学, 2015.
[26] 李炜. 谈楼盘设计中的围合式布局手法[J]. 房地产导刊, 2004(88):176-177.
[27] 胡文荟, 王舒, 赵宸. 场所营造与城市历史街区的微更新保护——以大连东关街为例[J]. 中国名城, 2018(1):5.
[28] 杨永悦. 如何利用信息技术提高建筑设计效率[J]. 建筑技术及设计, 2002(10):2.
[29] 沈萍. 街廓形态的几何分析[D]. 南京大学, 2011.
[30] 王金岩. 城市街廓模式研究[D]. 大连理工大学, 2006.
[31] Isola P, Zhu J Y, Zhou T, et al. Image-to-Image Translation with Conditional Adversarial Networks[J]. IEEE Conference on Computer Vision & Pattern Recognition, July 21-26, 2017, Honolulu, United states.
[32] Jia Kui, Xiaogang Wang, and Xiaoou Tang. Image transformation based on learning dictionaries across image spaces[J]. IEEE transactions on pattern analysis and machine intelligence 35.2 (2012): 367-380.

Appendices

Code(Function part):

```
import cv2
import math
import copy
import pyvista as pv
import numpy as np
import pandas as pd
import random as rd
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

MIN_AREA = 43000
MAX_AREA = 70000
APRATIO = 0.20
goodFeaturesToTrack_qualityLevel = 0.1055

def show_img(address):
    """获取文件地址并展示图片
    参数：文件地址（字符串）
    返回值：无
    """

    img = cv2.imread(address)
    plt.imshow(img)

def get_img_gray(address):
    """获取文件地址并返回其灰度图
    参数：文件地址（字符串）
    返回值：返回灰度图
    """

    img = cv2.imread(address)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img

def show_img_gray(address):
    """获取文件地址并展示其灰度图
    参数：文件地址（字符串）
```

```
    返回值：无
    """

    img = cv2.imread(address)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    plt.imshow(img,cmap='gray')

def save_arr(arr,address):
    """保存数组到文件
    参数：arr:数组；str:文件地址+/name.csv（字符串）
    返回值：无
    """

    data = pd.DataFrame(arr)
    data.to_csv(address)

def get_lines(img):
    """获取边线
    参数：0 和 255 的二值图像，255 为边缘
    返回值：列数为 4，行数为直线数的数组。每行包含两个点坐标 x1,y1,x2,y2。
    直线数量为 lines.shape[0]
    """

    lines = do_hough(img)
    lines = reconfirm_lines(lines,img)
    return lines

def do_canny(img):
    """Canny 边缘检测算子处理
    参数：8bit 灰度图
    返回值：0 和 255 的二值图像，255 为边缘
    """

#       img = cv2.GaussianBlur(img, (3, 3), 0)
    img = cv2.Canny(img, 100, 150)
    return img


def do_hough(img):
    """霍夫变换得到边上的一些线段（两点式）
```

```
    参数：0 和 255 的二值图像，255 为边缘
    返回值：列数为 4，行数为直线数的数组。每行包含两个点坐标 x1,y1,x2,y2。
    直线数量为 lines.shape[0]
    """

    img1 = do_canny(img)
    rho = 1                                 #距离分辨率
    theta = np.pi/180                       #角度分辨率
    threshold = 60                          #阈值：霍夫空间中多少个曲线相交表示一个正式的交点
    min_line_len = 60                       #最低线段的长度，小于该参数的线段不被显示。PRMT:最
     短边长
    max_line_gap = 50                       #两条线段端点小于这个参数则认为是同一条线段
    lines = cv2.HoughLinesP(img1, rho, theta, threshold, maxLineGap = max_line_gap)
    return lines

def draw_lines(lines):
    """绘制根据霍夫变换得到的边线
    参数：列数为 4，行数为边线数的数组。每行包含两个点坐标 x1,y1,x2,y2。
    返回值：无
    """

    newImageInfo = (1850,2000,1)
    dst = np.zeros(newImageInfo,np.uint8)
    for line in lines:
            for x1, y1, x2, y2 in line:
                dst = cv2.line(dst,(x1,y1),(x2,y2),255,5)
    plt.figure(figsize=(30,30))
    plt.imshow(dst,cmap="gray")

def get_centroid(img):
    """寻找几何图形质心
    参数：灰度图
    返回值：质心坐标(四舍五入到整数)
    """

    m = cv2.moments(img)
    x = m['m10']/m['m00']
    y = m['m01']/m['m00']
    centroid = [round(x),round(y)]
    return centroid
```

```
def get_area_from_pixel(img):
    """通过像素值求面积
    """

    area = len(img[img==255])
    return area

def get_points(img):
    """将 img 中所有灰度值不为零的点的坐标放在 img_list 中作为二维数组返回
    参数：img 灰度图
    返回值：img_list 数组
    """

    img_list = []
    for j in range(img.shape[0]):
        for i in range(img.shape[1]):
            if img[j,i] > 0:
                img_list.append([i,j])
    img_list=np.array(img_list)
    return img_list

def do_erode(img,kernel_size,iteration):
    """腐蚀操作
    参数：img:灰度图；kernel_size:核尺寸；iteration:迭代数
    返回值：腐蚀后的灰度图
    """

    kernel = np.ones((kernel_size, kernel_size), dtype=np.uint8)
    dst = cv2.erode(img, kernel, iteration)
    return dst

def do_dilate(img,kernel_size,iteration):
    """膨胀操作
    参数：img:灰度图；kernel_size:核尺寸；iteration:迭代数
    返回值：膨胀后的灰度图
    """

    kernel = np.ones((kernel_size, kernel_size), dtype=np.uint8)
    dst = cv2.dilate(img, kernel, iteration)
```

```
    return dst

def gen_basic_building(img):
    """膨胀处理背景生成基本合院建筑
    参数：灰度图
    返回值：结果图，合院建筑数值为 150
    """

    ds2 = do_erode(img,70,1)
    ds1 = cut_acute_angle(img)
#       ds1 = img
    ds3 = ds1-ds2
    ret,ds4 = cv2.threshold(ds3,50,255,0)
#       dst = img/255*30 + ds4
    dst = ds4
    return dst

def reconfirm_lines(lines,img):
    """改进霍夫变换缺陷，对线段进行聚类，重新确定边线
    参数：lines：霍夫变换得到的边线；img：原灰度图
    返回值：新确定的边线
    """

    pick_lines = []
    for i in range(lines.shape[0]):
        pick_lines.append([lines[i,0,0],lines[i,0,1],lines[i,0,2],lines[i,0,3]])
    pick_lines = np.array(pick_lines)

    #寻找在同一边线上的直线(通过斜率、横截距、纵截距三项特征进行聚类)
    lines_group = []
    for i in range(pick_lines.shape[0]):
        lines_group.append([pick_lines[i,0],pick_lines[i,1],pick_lines[i,2],pick_lines[i,3],0,0,0,0])
    lines_group = np.array(lines_group)

    #计算斜率
    for i in range(lines_group.shape[0]):
        delta_x = lines_group[i,0] - lines_group[i,2]
        delta_y = lines_group[i,1] - lines_group[i,3]
        if delta_x == 0:
            lines_group[i,4]=90
```

```
    elif delta_y == 0 :
        lines_group[i,4]=0
    else:
        k = -delta_y / delta_x * 1.0
        # 将得到的弧度转换为度
        lines_group[i,4] = round(np.arctan(k) * 57.29577)
#计算与过质心且平行于 x 轴的直线的横截距
for i in range(lines_group.shape[0]):
    delta_x = lines_group[i,0] - lines_group[i,2]
    delta_y = lines_group[i,1] - lines_group[i,3]
    if delta_y == 0:
        lines_group[i,5] = -12345678
    elif delta_x == 0:
        lines_group[i,5] = lines_group[i,0]
    else:
        lines_group[i,5] = lines_group[i,0] + ((get_centroid(img)[1] -
 lines_group[i,1])*delta_x/delta_y)
#计算与过质心且平行于 y 轴的直线的纵截距
for i in range(lines_group.shape[0]):
    delta_x = lines_group[i,2] - lines_group[i,0]
    delta_y = lines_group[i,3] - lines_group[i,1]
    if delta_y == 0:
        lines_group[i,6] = lines_group[i,1]
    elif delta_x == 0:
        lines_group[i,6] = -12345678
    else:
        lines_group[i,6] = lines_group[i,1] + ((get_centroid(img)[0] -
 lines_group[i,0])*delta_y/delta_x)
#进行聚类
#允许误差范围（图片的尺寸的 1/50）
margin_error_x = img.shape[0]/50
margin_error_y = img.shape[1]/50
group_number = -1
flag = True
for i in range(lines_group.shape[0]):
    for j in range(i):
        #必须满足：1.角度相差不大于 3°度；2.横截距或者纵截距至少有一个相差在允许
 误差范围内
        if (
        lines_group[j,4] - 3 < lines_group[i,4] and lines_group[i,4] < lines_group[j,4] + 3
        and
```

```
            (
                (lines_group[j,5] - margin_error_x < lines_group[i,5] and lines_group[i,5] < lines_group[j,5] + margin_error_x and lines_group[i,5] != -12345678)
                or
                (lines_group[j,6] - margin_error_y < lines_group[i,6] and lines_group[i,6] < lines_group[j,6] + margin_error_y and lines_group[i,6] != -12345678)
            )
            ):
                lines_group[i,7] = lines_group[j,7]
                flag = False
        if flag:
            group_number = group_number + 1
            lines_group[i,7] = group_number
        flag = True
#在同一聚类里找到距离最远的两个点作为两端
new_lines = []
temp_arr = lines_group[:,7]
temp = -1
for i in range(temp_arr.shape[0]):
    if temp_arr[i] > temp:
        temp = temp_arr[i]#确认有多少条边
for j in range(temp + 1):
    pts = []
    candidate = []
    dist = 10
    for k in range(lines_group.shape[0]):
        if lines_group[k,7] == j:
            pts.append(lines_group[k,0:2])
            pts.append(lines_group[k,2:4])#把同一条边上的线段的端点都放进去
    pts = np.array(pts)
    for m in range(pts.shape[0]-1):
        for n in range(m+1, pts.shape[0]):
            temp_length = get_length(pts[m],pts[n])
            if dist < temp_length:
                dist = temp_length
                candidate = [pts[m,0],pts[m,1],pts[n,0],pts[n,1]]
    candidate = np.array(candidate)
    new_lines.append(candidate)
new_lines = np.array(new_lines)
np.warnings.filterwarnings('ignore', category=np.VisibleDeprecationWarning)
return new_lines
```

```
def get_cross(line1, line2):
    """计算两直线交点
    参数：line1(arr[4]), line2(arr[4])
    返回值：point_is_exist(bool),[x, y](arr[2])
    """

    #判断是否存在交点
    point_is_exist=False
    x=0
    y=0
    x1 = line1[0]  # 取四点坐标
    y1 = line1[1]
    x2 = line1[2]
    y2 = line1[3]
    x3 = line2[0]
    y3 = line2[1]
    x4 = line2[2]
    y4 = line2[3]

    if (x2 - x1) == 0:
        k1 = None
    else:
        k1 = (y2 - y1) * 1.0 / (x2 - x1)  # 计算 k1,进行整型转浮点型
        b1 = y1 * 1.0 - x1 * k1 * 1.0
    if (x4 - x3) == 0:  # 如果 L2 直线斜率不存在
        k2 = None
        b2 = 0
    else:
        k2 = (y4 - y3) * 1.0 / (x4 - x3)  # 如果斜率存在
        b2 = y3 * 1.0 - x3 * k2 * 1.0

    if k1 is None:
        if not k2 is None:
            x = x1
            y = k2 * x1 + b2
            point_is_exist=True
    elif k2 is None:
        x=x3
        y=k1*x3+b1
```

```
            point_is_exist=True
        elif not k2==k1:
            x = (b2 - b1) * 1.0 / (k1 - k2)
            y = k1 * x * 1.0 + b1 * 1.0
            point_is_exist=True
        x = round(x)
        y = round(y)
        return point_is_exist,[x, y]

def if_inside_img_shape(point, img):
        """判断是否在图像内
        参数：point(arr[2]), img(灰度图)
        返回值：bool 值
        """

        if point[0] < img.shape[1] and point[1] < img.shape[0] and point[0] > 0 and point[1] > 0:
            return True
        return False


def adjust_pts_order(pts_2ds):
        """
        对顶点进行排序
        """

        cen_x, cen_y = np.mean(pts_2ds, axis=0)
        d2s = []
        for i in range(len(pts_2ds)):

            o_x = pts_2ds[i][0] - cen_x
            o_y = pts_2ds[i][1] - cen_y
            atan2 = np.arctan2(o_y, o_x)
            if atan2 < 0:
                atan2 += np.pi * 2
            d2s.append([pts_2ds[i], atan2])

        d2s = sorted(d2s, key=lambda x:x[1])
        order_2ds = np.array([x[0] for x in d2s])

        return order_2ds
```

```
def temp_get_corners(img):
    img_float = np.float32(img)
    corners_float = cv2.goodFeaturesToTrack(img_float, 4, goodFeaturesToTrack_qualityLevel, 5)
    corners = []
    for corner_float in corners_float:
        for x, y in corner_float:
            corners.append([round(x), round(y)])
    corners = np.array(corners)
    return corners

def get_corners(img):
    """获得四边形或者三角形的顶点
    参数：img 灰度图
    返回值：corners(arr[n,2])
    """

    corners = temp_get_corners(img)

    if corners.shape[0] == 3:
        return corners
    #若为四边形，则对顶点进行排序
    if corners.shape[0] == 4:
        corners = adjust_pts_order(corners)
        return corners
def get_angles(b,a,c):
    """通过三点求角度
    参数： a (arr[2])， b (arr[2])， c (arr[2])
    返回：在 a 点形成的角度 angle(角度制)
    """

    ab = ((b[0]*1.0-a[0])**2 + (b[1]-a[1])**2)**0.5
    ac = ((c[0]*1.0-a[0])**2 + (c[1]-a[1])**2)**0.5
    angles = math.acos(((b[0]-a[0])*(c[0]-a[0])+(b[1]-a[1])*(c[1]-a[1]))/ab/ac)
    #换算为角度制
    angles = math.degrees(angles)
    return round(angles)

def get_corners_n_angles(corners):
    """通过多边形顶点生成角度
    参数：corners(array[n,2])
```

```
    输出：array[n,3]最后一列为角度
    """

    corners_n_angles = []
    for i in range(corners.shape[0]):
        corners_n_angles.append([corners[i,0],corners[i,1],0])
    corners_n_angles.append([corners[0,0],corners[0,1],0])
    corners_n_angles.append([corners[1,0],corners[1,1],0])
    corners_n_angles = np.array(corners_n_angles)
    for i in range(1,corners.shape[0]+1):
        b = [corners_n_angles[i-1,0],corners_n_angles[i-1,1]]
        a = [corners_n_angles[i,0],corners_n_angles[i,1]]
        c = [corners_n_angles[i+1,0],corners_n_angles[i+1,1]]
        corners_n_angles[i,2] = get_angles(b,a,c)
    corners_n_angles = corners_n_angles[1:corners.shape[0]+1]
    return corners_n_angles

def get_length(p1, p2):
    """求两点之间长度
    参数：p1(arr); p2(arr)
    返回值：长度
    """

    length = 1.0*((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)**0.5
    return length

def get_distance_from_point_to_line(point, line_point1, line_point2):
    """
    求点到直线的距离
    参数：point, line_point1, line_point2
    返回值：distance
    """

    #计算直线的三个参数
    A = line_point2[1] - line_point1[1]
    B = line_point1[0] - line_point2[0]
    C = (line_point1[1] - line_point2[1]) * line_point1[0] + \
        (line_point2[0] - line_point1[0]) * line_point1[1]
    #根据点到直线的距离公式计算距离
    distance = np.abs(A * point[0] + B * point[1] + C) / (np.sqrt(A**2 + B**2))
```

```
    return distance

def get_foot(point_a, start_point, end_point):
    """
    获取直线 与 点的垂足
    """

    start_x, start_y = start_point
    end_x, end_y = end_point
    pa_x, pa_y = point_a

    p_foot = [0, 0]
    if start_point[0] == end_point[0]:
        p_foot[0] = start_point[0]
        p_foot[1] = point_a[1]
        return p_foot

    k = (end_y - start_y) * 1.0 / (end_x - start_x)
    a = k
    b = -1.0
    c = start_y - k * start_x
    p_foot[0] = int((b * b * pa_x - a * b * pa_y - a * c) / (a * a + b * b))
    p_foot[1] = int((a * a * pa_y - a * b * pa_x - b * c) / (a * a + b * b))

    return p_foot


def get_area(corners):
    """通过多边形顶点求面积
    参数：corners
    返回：面积
    """

    #海伦一秦九韶公式:
    if corners.shape[0] == 3:
        a = get_length(corners[0],corners[1])
        b = get_length(corners[1],corners[2])
        c = get_length(corners[2],corners[0])
        p = (a + b + c)/2
        area = (p * (p - a) * (p - b) * (p - c))**0.5
    if corners.shape[0] == 4:
```

```
            a = get_length(corners[0],corners[1])
            b = get_length(corners[1],corners[2])
            c = get_length(corners[2],corners[3])
            d = get_length(corners[3],corners[0])
            e = get_length(corners[0],corners[2])
            p1 = (a + b + e)/2
            p2 = (c + d + e)/2
            area1 = (p1 * (p1 - a) * (p1 - b) * (p1 - e))**0.5
            area2 = (p2 * (p2 - c) * (p2 - d) * (p2 - e))**0.5
            area = area1 + area2
        return round(area)

def get_perimeter(corners):
        """获取周长
        参数：corners
        返回：周长
        """

        if corners.shape[0] == 3:
            a = get_length(corners[0],corners[1])
            b = get_length(corners[1],corners[2])
            c = get_length(corners[2],corners[0])
            perimeter = a + b + c
        if corners.shape[0] == 4:
            a = get_length(corners[0],corners[1])
            b = get_length(corners[1],corners[2])
            c = get_length(corners[2],corners[3])
            d = get_length(corners[3],corners[0])
            perimeter = a + b + c + d
        return round(perimeter)

def get_apratio(img):
        """近圆率的二次方根结果
        参数：corners
        返回：比值
        """

        corners = get_corners(img)
        area = get_area_from_pixel(img)
        perimeter = get_perimeter(corners)
        circularity = area/(perimeter**2)
```

```
    apratio = circularity**0.5
    return apratio

def choose_corner0_or_edge1(possibility_of_egde):
    """掷骰子：决定选择边/顶点
    参数：possibility_of_egde 边的概率（百分比，如 30）
    返回：0：选择边；1：选择顶点
    """

    karma = rd.randint(0,99)
    if karma < possibility_of_egde:
        return 1
    else:
        return 0

def get_design_img(chosen_line, img):
    """根据抽中的边，去掉靠近端点的一部分（每段各占总长度的 1/k），以及其他设计上的
     优化
    参数：chosen_line：选中的边, img：原图，用于获取尺寸
    返回值：design_img：与 img 尺寸相同的灰度图
    """

    k = 5
    center_1 = [chosen_line[0], chosen_line[1]]
    center_2 = [chosen_line[2], chosen_line[3]]
    center_3 = [round((chosen_line[0] + chosen_line[2]) / 2), round((chosen_line[1] + chosen_line[3])
     / 2)]
#     center_1 = [chosen_line[1], chosen_line[0]]
#     center_2 = [chosen_line[3], chosen_line[2]]
#         center_3 = [round((chosen_line[1] + chosen_line[3]) / 2), round((chosen_line[0] +
     chosen_line[2]) / 2)]

    newImageInfo = (img.shape[0],img.shape[1])
    design_img = np.zeros(newImageInfo,np.uint8)
    optimize_img = np.zeros(newImageInfo,np.uint8)
    #基础图像#去除两端
    cv2.line(design_img, center_1, center_2, 15, 1)
    length = get_length(center_1, center_2)
    radius = round(length/k)
    cv2.circle(design_img, center_1, radius, 0, -1)
```

```
    cv2.circle(design_img, center_2, radius, 0, -1)
    #优化图像#中点加强
    cv2.circle(optimize_img, center_3, round(length/10), 255, -1)

    #结果为优化图像和基础图像相乘得到的图，再加基础图像
    optimize_img = cv2.multiply(design_img, optimize_img)
    design_img = cv2.add(design_img, optimize_img)

    return design_img

def get_points_n_possibility(design_img,x1,x2,y1,y2):
    """扫描图片，以灰度值作为每个像素被选中的概率
    参数：img 灰度图；x1,x2,y1,y2 为扫描范围
    返回值：pts_n_possibility（n*3 的数组，每行分别为横纵坐标和概率）
    """

    #i 为横坐标，j 为纵坐标
    pts_n_possibility = []
    for j in range(y1,y2+1):
        for i in range(x1,x2+1):
            if design_img[j,i] > 0:
                pts_n_possibility.append([i,j,design_img[j,i]])
    pts_n_possibility = np.array(pts_n_possibility)
    return pts_n_possibility

def get_corners_n_possibility(corners_n_angles):
    """根据角度大小，判断每个顶点被抽中的概率
    参数：array[n,3]最后一列为角度
    返回值：corners_n_possibility(array[n,3]最后一列为概率)
    """

    min_angle_can_be_split = 80
    how_slow_possibility_grows_as_angle_increases = 8

    corners_n_possibility = []
    for i in range(corners_n_angles.shape[0]):
        if corners_n_angles[i,2] >= min_angle_can_be_split:
            temp_possibility = 1 + (((corners_n_angles[i,2]-
  min_angle_can_be_split)/10)**2)/how_slow_possibility_grows_as_angle_increases
```

```
        corners_n_possibility.append([corners_n_angles[i,0],corners_n_angles[i,1],round(temp_possibi
        lity*100)])
    corners_n_possibility = np.array(corners_n_possibility)
    return corners_n_possibility

# def get_edges_n_possibility()

def get_edges(corners):
    """根据顶点获得多边形的边
    参数：corners（array[n,2]）
    返回值：edge（array[n,4]）
    """

    edge = []
    for i in range(corners.shape[0]):
        if i == corners.shape[0] - 1:
            edge.append([corners[i,0],corners[i,1],corners[0,0],corners[0,1]])
        else:
            edge.append([corners[i,0],corners[i,1],corners[i+1,0],corners[i+1,1]])
    edge = np.array(edge)
    return edge

def get_edges_n_lengths(edges):
    """根据多边形的边获得长度
    参数：edge（array[n,4]）
    返回值：edges_n_lengths（array[n,5]）
    """

    get_edges_n_length = []
    for i in range(edges.shape[0]):
        x1, y1, x2, y2 = edges[i]
        length = round(get_length([x1, y1],[x2, y2]))
        get_edges_n_length.append([x1, y1, x2, y2, length])
    get_edges_n_length = np.array(get_edges_n_length)
    return get_edges_n_length

def get_edges_n_possibility(edges):
    """根据多边形的边获得每条边被抽中的概率
    参数：edges（array[n,4]）
```

```
    返回值：edges_n_possibility（array[n,5]）
    """

    RADIX = 9

    edges_n_lengths = get_edges_n_lengths(edges)
    temp_sum = np.sum(edges_n_lengths,axis = 0)
    avg_edge_length = temp_sum[4]/edges_n_lengths.shape[0]
    edges_n_possibility = []
    for i in range(edges_n_lengths.shape[0]):
        #使用指数函数
        possibility = RADIX**((edges_n_lengths[i,4] - avg_edge_length)/avg_edge_length)
        edges_n_possibility.append([edges[i,0], edges[i,1], edges[i,2], edges[i,3],
  round(possibility*100)])
    edges_n_possibility = np.array(edges_n_possibility)
    return edges_n_possibility

def roulette(array):
    """轮盘赌抽签
    参数：array（最后一列为概率（需整数，最好大于两位数））
    返回值：result（抽中的序列号（int 型 0~n）），若没有可参与抽签的对象，则返回-1
    """

    roulette = []
    number = 0
    result = -1
    if array.shape[0] == 0:
        return result
    for i in range(array.shape[0]):
        number = number + array[i,-1]
        roulette.append(number)
    roulette = np.array(roulette)
    pointer = rd.randint(1,number)
    for j in range(roulette.shape[0]):
        if roulette[j] >= pointer:
            result = j
            break
    return result

def random_perp_or_par(split_img, edges, centroid, i):
    x0, y0 = centroid
```

```
if i == 0:
    x1, y1, x2, y2 = edges[0]
    dice = rd.randint(0, 2)
    if dice == 0:
        x3, y3, x4, y4 = edges[1]
        x00 = x0 + x3 - x4
        y00 = y0 + y3 - y4
        point_is_exist, cross_point = get_cross([x0, y0, x00, y00],[x1, y1, x2, y2])
    elif dice == 1:
        x3, y3, x4, y4 = edges[2]
        x00 = x0 + x3 - x4
        y00 = y0 + y3 - y4
        point_is_exist, cross_point = get_cross([x0, y0, x00, y00],[x1, y1, x2, y2])
    else:
        cross_point = get_foot([x0, y0],[x1, y1],[x2, y2])
    cv2.line(split_img, centroid, cross_point, 0, 10)
elif i == 1:
    x1, y1, x2, y2 = edges[1]
    dice = rd.randint(0, 2)
    if dice == 0:
        x3, y3, x4, y4 = edges[2]
        x00 = x0 + x3 - x4
        y00 = y0 + y3 - y4
        point_is_exist, cross_point = get_cross([x0, y0, x00, y00],[x1, y1, x2, y2])
    elif dice == 1:
        x3, y3, x4, y4 = edges[0]
        x00 = x0 + x3 - x4
        y00 = y0 + y3 - y4
        point_is_exist, cross_point = get_cross([x0, y0, x00, y00],[x1, y1, x2, y2])
    else:
        cross_point = get_foot([x0, y0],[x1, y1],[x2, y2])
    cv2.line(split_img, centroid, cross_point, 0, 10)
elif i == 2:
    x1, y1, x2, y2 = edges[2]
    dice = rd.randint(0, 2)
    if dice == 0:
        x3, y3, x4, y4 = edges[1]
        x00 = x0 + x3 - x4
        y00 = y0 + y3 - y4
        point_is_exist, cross_point = get_cross([x0, y0, x00, y00],[x1, y1, x2, y2])
    elif dice == 1:
```

```
            x3, y3, x4, y4 = edges[0]
            x00 = x0 + x3 - x4
            y00 = y0 + y3 - y4
            point_is_exist, cross_point = get_cross([x0, y0, x00, y00],[x1, y1, x2, y2])
        else:
            cross_point = get_foot([x0, y0],[x1, y1],[x2, y2])
        cv2.line(split_img, centroid, cross_point, 0, 10)


def get_split_img_triangle_0(corners, img):
    """三角形分割_0:两边分割
    参数：corners（array[n,2]）;img 原图，用于获取尺寸
    返回值：切割后的灰度图，包含两个连通域（255），其余皆为黑色背景（0）
    """

    edges = get_edges(corners)
    edges_n_lengths = get_edges_n_lengths(edges)
    edges_n_possibility = get_edges_n_possibility(edges)

    chosen_edge_1 = roulette(edges_n_possibility)
    temp_img_1 = get_design_img(edges[chosen_edge_1], img)
    x1_1 = min(edges[chosen_edge_1,0],edges[chosen_edge_1,2])
    x2_1 = max(edges[chosen_edge_1,0],edges[chosen_edge_1,2])
    y1_1 = min(edges[chosen_edge_1,1],edges[chosen_edge_1,3])
    y2_1 = max(edges[chosen_edge_1,1],edges[chosen_edge_1,3])
    points_n_possibility_1 = get_points_n_possibility(temp_img_1, x1_1, x2_1, y1_1, y2_1)
    points_n_possibility_1
    result_index_1 = roulette(points_n_possibility_1)
    chosen_point_1                                                                    =
     (points_n_possibility_1[result_index_1,0],points_n_possibility_1[result_index_1,1])

    chosen_edge_2 = roulette(edges_n_possibility)
    while(chosen_edge_2 == chosen_edge_1):
        chosen_edge_2 = roulette(edges_n_possibility)
    temp_img_2 = get_design_img(edges[chosen_edge_2], img)

    x1_2 = min(edges[chosen_edge_2,0],edges[chosen_edge_2,2])
    x2_2 = max(edges[chosen_edge_2,0],edges[chosen_edge_2,2])
    y1_2 = min(edges[chosen_edge_2,1],edges[chosen_edge_2,3])
    y2_2 = max(edges[chosen_edge_2,1],edges[chosen_edge_2,3])
    points_n_possibility_2 = get_points_n_possibility(temp_img_2, x1_2, x2_2, y1_2, y2_2)
```

```
    result_index_2 = roulette(points_n_possibility_2)
    chosen_point_2 =
      (points_n_possibility_2[result_index_2,0],points_n_possibility_2[result_index_2,1])

    newImageInfo = (img.shape[0],img.shape[1])
    split_img = np.zeros(newImageInfo,np.uint8)
    split_corners = np.array([corners])
    cv2.fillPoly(split_img, split_corners, 255)
    cv2.line(split_img, chosen_point_1, chosen_point_2, 0, 10)
    return split_img

def get_split_img_triangle_1(corners, img):
    #当三角形为直角三角形或者钝角三角形时，最长边的切法不能是平行切法
    split_img = np.copy(img)
    centroid = get_centroid(split_img)
    edges = get_edges(corners)
    corners_n_angles = get_corners_n_angles(corners)
    if_angle_over_90 = any(corners_n_angles[:,2] >= 90)
    if if_angle_over_90:
        #当三角形为直角三角形或者钝角三角形时
        edges_n_lengths = get_edges_n_lengths(edges)
        longest_index = 0
        for i in range(3):
            if edges_n_lengths[i,4] == max(edges_n_lengths[:,4]):
                longest_index = i
                break
        if longest_index == 0:
            x1, y1, x2, y2 = edges[0]
            cross_point = get_foot(centroid,[x1, y1],[x2, y2])
            cv2.line(split_img, centroid, cross_point, 0, 10)
            random_perp_or_par(split_img, edges, centroid, 1)
            random_perp_or_par(split_img, edges, centroid, 2)
        if longest_index == 1:
            x1, y1, x2, y2 = edges[1]
            cross_point = get_foot(centroid,[x1, y1],[x2, y2])
            cv2.line(split_img, centroid, cross_point, 0, 10)
            random_perp_or_par(split_img, edges, centroid, 0)
            random_perp_or_par(split_img, edges, centroid, 2)
        if longest_index == 2:
            x1, y1, x2, y2 = edges[2]
            cross_point = get_foot(centroid,[x1, y1],[x2, y2])
```

```
                cv2.line(split_img, centroid, cross_point, 0, 10)
                random_perp_or_par(split_img, edges, centroid, 1)
                random_perp_or_par(split_img, edges, centroid, 0)

        else:
            #当三角形为锐角三角形时
            #第一条边
            random_perp_or_par(split_img, edges, centroid, 0)
            #第二条边
            random_perp_or_par(split_img, edges, centroid, 1)
            #第三条边
            random_perp_or_par(split_img, edges, centroid, 2)
        return split_img

def get_split_img_quadrangle_0(corners, img):
    """四边形分割_0:对边分割
    参数：corners（array[n,2]）;img 原图，用于获取尺寸
    返回值：切割后的灰度图，包含两个连通域（255），其余皆为黑色背景（0）
    """

    edges = get_edges(corners)
    edges_n_lengths = get_edges_n_lengths(edges)
    edges_n_possibility = get_edges_n_possibility(edges)

    chosen_edge_1 = roulette(edges_n_possibility)
    chosen_edge_2 = chosen_edge_1 + 2
    if chosen_edge_2 > 3:
        chosen_edge_2 = chosen_edge_2 - 4
    temp_img_1 = get_design_img(edges[chosen_edge_1], img)

    x1_1 = min(edges[chosen_edge_1,0],edges[chosen_edge_1,2])
    x2_1 = max(edges[chosen_edge_1,0],edges[chosen_edge_1,2])
    y1_1 = min(edges[chosen_edge_1,1],edges[chosen_edge_1,3])
    y2_1 = max(edges[chosen_edge_1,1],edges[chosen_edge_1,3])
    points_n_possibility_1 = get_points_n_possibility(temp_img_1, x1_1, x2_1, y1_1, y2_1)
    points_n_possibility_1
    result_index_1 = roulette(points_n_possibility_1)
    chosen_point_1                                                                    =
     (points_n_possibility_1[result_index_1,0],points_n_possibility_1[result_index_1,1])
    temp_img_2 = get_design_img(edges[chosen_edge_2], img)
```

```
    x1_2 = min(edges[chosen_edge_2,0],edges[chosen_edge_2,2])
    x2_2 = max(edges[chosen_edge_2,0],edges[chosen_edge_2,2])
    y1_2 = min(edges[chosen_edge_2,1],edges[chosen_edge_2,3])
    y2_2 = max(edges[chosen_edge_2,1],edges[chosen_edge_2,3])
    points_n_possibility_2 = get_points_n_possibility(temp_img_2, x1_2, x2_2, y1_2, y2_2)
    result_index_2 = roulette(points_n_possibility_2)
    chosen_point_2                                                                    =
      (points_n_possibility_2[result_index_2,0],points_n_possibility_2[result_index_2,1])

    newImageInfo = (img.shape[0],img.shape[1])
    split_img = np.zeros(newImageInfo, np.uint8)
    split_corners = np.array([corners])
    cv2.fillPoly(split_img, split_corners, 255)
    cv2.line(split_img, chosen_point_1, chosen_point_2, 0, 10)
    return split_img

def get_split_img_quadrangle_1(corners, img):
    """四边形分割_1:角边分割
    参数：corners（array[n,2]）;img 原图，用于获取尺寸
    返回值：切割后的灰度图，包含两个连通域（255），其余皆为黑色背景（0）
    """

    ANGLE_0 = 90
    PARA_1 = 5
    PARA_2 = 12

    corners_n_angles = get_corners_n_angles(corners)
    chosen_corners = []
    for i in range(corners_n_angles.shape[0]):
        if corners_n_angles[i,2] >= ANGLE_0:
            chosen_corners.append([i,corners_n_angles[i,2]])
    chosen_corners = np.array(chosen_corners)
    corner_index = chosen_corners[roulette(chosen_corners),0]

    a_index = corner_index
    b_index = corner_index + 1
    c_index = corner_index + 2
    d_index = corner_index + 3

    if b_index > 3:
        b_index = b_index - 4
```

```
if c_index > 3:
    c_index = c_index - 4
if d_index > 3:
    d_index = d_index - 4

a = np.array(corners_n_angles[a_index,:2])
b = np.array(corners_n_angles[b_index,:2])
c = np.array(corners_n_angles[c_index,:2])
d = np.array(corners_n_angles[d_index,:2])

L1 = np.append(b,d)
L2 = np.append(a,c)
L4 = np.append(b,c)
L5 = np.append(d,c)
point_is_exist, e = get_cross(L1, L2)

e1 = (d-e)/PARA_1 + e
e1 = e1.astype(int)
e2 = (d-e)/PARA_2 + e
e2 = e2.astype(int)
e3 = (b-e)/PARA_2 + e
e3 = e3.astype(int)
e4 = (b-e)/PARA_1 + e
e4 = e4.astype(int)

length_e1e2 = get_length(e1,e2)
length_e3e4 = get_length(e3,e4)
newImageInfo = (img.shape[0],img.shape[1])
design_img = np.zeros(newImageInfo,np.uint8)
if length_e3e4 > length_e1e2:
    cv2.line(design_img,e3,e4,100,1)
    x_list=[e3[0],e4[0]]
    y_list=[e3[1],e4[1]]
    L = L4
if length_e3e4 <= length_e1e2:
    cv2.line(design_img,e1,e2,100,1)
    x_list=[e1[0],e2[0]]
    y_list=[e1[1],e2[1]]
    L = L5
```

```
    points_n_possibility =
      get_points_n_possibility(design_img,min(x_list),max(x_list),min(y_list),max(y_list))
    f = points_n_possibility[roulette(points_n_possibility),:2]
    L3 = np.append(a,f)
    point_is_exist, g = get_cross(L3, L)
    g = np.array(g)

    chosen_point_1 = a
    chosen_point_2 = g

    split_img = np.copy(img)
    cv2.line(split_img, chosen_point_1, chosen_point_2, 0, 10)
    return split_img

def if_min_area(area):
    """判断是否还需要切分
    参数：area 面积
    返回值：布尔值，需要切分返回 True
    """

    if area > MAX_AREA:
        return True
    elif area >= MIN_AREA and area <= MAX_AREA:
        random_int = rd.randint(MIN_AREA, MAX_AREA)
        if random_int <= area:
            return True
    else:
        return False

def do_seperate_unconnected_components(split_img):
    """分离非连通域
    参数：split_img 一个包含多个非连通域的二值图像
    返回值：sub_img 一个列表，包含了多个与原图像相同尺寸的、包含单一连通域的二值图
      像
    """

    #获得标记图
    num_objects, labels_img = cv2.connectedComponents(split_img)
    newImageInfo = (labels_img.shape[0],labels_img.shape[1])
```

```
    #创建一个列表，用来存放分离后的图像
    sub_img= {}
    for n in range(num_objects - 1):
        sub_img[n] = np.zeros(newImageInfo,np.uint8)

    #遍历标记图
    for j in range(labels_img.shape[0]):
        for i in range(labels_img.shape[1]):
            label = labels_img[j,i]
            if label > 0:
                sub_img[label - 1][j,i] = 255
    return sub_img

def cut_acute_angle(img):
    """以一定概率切除锐角
    参数：img
    返回值：processing_img 切除后的图片"""
    processing_img = np.copy(img)
    corners = get_corners(processing_img)
    corners_n_angles = get_corners_n_angles(corners)
    corners_num = corners_n_angles.shape[0]
    processing_index = []
    for i in range(corners_num):
        if corners_n_angles[i,2] < 30:
            processing_index.append(i)
        elif corners_n_angles[i,2] >= 30 and corners_n_angles[i,2] < 90:
            if rd.randint(30,90) > corners_n_angles[i,2]:
                processing_index.append(i)

    for i in range(len(processing_index)):
        index = processing_index[i]
        x = corners_n_angles[index,0]
        y = corners_n_angles[index,1]
        if index == 0:
            x1 = corners_n_angles[corners_num-1,0]
            y1 = corners_n_angles[corners_num-1,1]
        else:
            x1 = corners_n_angles[index-1,0]
            y1 = corners_n_angles[index-1,1]
        if index == corners_num-1:
            x2 = corners_n_angles[0,0]
```

```
        y2 = corners_n_angles[0,1]
    else:
        x2 = corners_n_angles[index+1,0]
        y2 = corners_n_angles[index+1,1]

    length_1 = ((x1-x)**2 + (y1-y)**2)**0.5
    length_2 = ((x2-x)**2 + (y2-y)**2)**0.5

    CUT_LENGTH = 30

    dice = rd.randint(1,8)

    #1.做垂线
    if dice < 3:
        L1 = CUT_LENGTH/np.sin(math.radians(corners_n_angles[index,2]))
        L2 = CUT_LENGTH/np.tan(math.radians(corners_n_angles[index,2]))
        x_p1 = round(x + (x1 - x) * L1 / length_1)
        y_p1 = round(y + (y1 - y) * L1 / length_1)
        x_p2 = round(x + (x2 - x) * L2 / length_2)
        y_p2 = round(y + (y2 - y) * L2 / length_2)
        pts = []
        pts.append([x,y])
        pts.append([x_p1,y_p1])
        pts.append([x_p2,y_p2])
        pts = np.array(pts)
        cv2.fillConvexPoly(processing_img, pts, 0)
        processing_img = do_erode(processing_img, 7, 1)
        processing_img = do_dilate(processing_img, 7, 1)

    #2.做另一条边的垂线
    elif dice >=3 and dice < 5:
        L1 = CUT_LENGTH/np.sin(math.radians(corners_n_angles[index,2]))
        L2 = CUT_LENGTH/np.tan(math.radians(corners_n_angles[index,2]))
        x_p1 = round(x + (x1 - x) * L2 / length_1)
        y_p1 = round(y + (y1 - y) * L2 / length_1)
        x_p2 = round(x + (x2 - x) * L1 / length_2)
        y_p2 = round(y + (y2 - y) * L1 / length_2)
        pts = []
        pts.append([x,y])
        pts.append([x_p1,y_p1])
        pts.append([x_p2,y_p2])
```

```
                pts = np.array(pts)
                cv2.fillConvexPoly(processing_img, pts, 0)
                processing_img = do_erode(processing_img, 7, 1)
                processing_img = do_dilate(processing_img, 7, 1)

        #3.做等腰三角形
            elif dice >= 5:
                L = CUT_LENGTH/2/np.sin(math.radians(corners_n_angles[index,2]/2.0))
                x_p1 = round(x + (x1 - x) * L / length_1)
                y_p1 = round(y + (y1 - y) * L / length_1)
                x_p2 = round(x + (x2 - x) * L / length_2)
                y_p2 = round(y + (y2 - y) * L / length_2)
                pts = []
                pts.append([x,y])
                pts.append([x_p1,y_p1])
                pts.append([x_p2,y_p2])
                pts = np.array(pts)
                cv2.fillConvexPoly(processing_img, pts, 0)
                processing_img = do_erode(processing_img, 7, 1)
                processing_img = do_dilate(processing_img, 7, 1)

        return processing_img

def do_split(spliting_img):
        waiting_list = []
        completed_list = []
        temp_list = []

        sub_imgs = do_seperate_unconnected_components(spliting_img)
        for i in range (len(sub_imgs)):#若面积小于 MIN_AREA 则视为已经完成，不需要切割
            area = get_area_from_pixel(sub_imgs[i])
            if if_min_area(area):
                waiting_list.append(sub_imgs[i])
            else:
                completed_list.append(sub_imgs[i])

        while waiting_list:#只要 waiting_list 不为空，就一直进行
            for spliting_img in waiting_list:

                flag = 1
                while flag:
```

```
                corners = get_corners(spliting_img)
                if corners.shape[0] == 4:
                    if rd.randint(0,1) < 1:
                        split_img = get_split_img_quadrangle_0(corners, spliting_img)
                    else:
                        split_img = get_split_img_quadrangle_1(corners, spliting_img)
                if corners.shape[0] == 3:
                    if rd.randint(0,1) < 1:
                        split_img = get_split_img_triangle_0(corners, spliting_img)
                    else:
                        split_img = get_split_img_triangle_1(corners, spliting_img)
                sub2_imgs = do_seperate_unconnected_components(split_img)

                #若 AP 比小于 0.75 则重新切分
                flag2 = 0
                for k in range(len(sub2_imgs)):
                    if get_area_from_pixel(sub2_imgs[k]) < 7000 or get_apratio(sub2_imgs[k])
    < APRATIO:
                        flag2 = 1
                if flag2:
                    continue

                #若切分后面积小于 MIN_AREA 则视为完成，否则继续切割
                for j in range(len(sub2_imgs)):
                    area = get_area_from_pixel(sub2_imgs[j])
                    if if_min_area(area):
                        temp_list.append(sub2_imgs[j])
                    else:
                        completed_list.append(sub2_imgs[j])
                flag = 0


            waiting_list.clear()
            waiting_list = copy.deepcopy(temp_list)
            temp_list.clear()

        return completed_list

def gen_design(img):
    """生成设计图
    参数：img 原图
```

```
    返回值：design 总设计灰度图, result_img 建筑设计灰度图, completed_list
    """

    img2 = do_erode(img, 40, 1)
    completed_list = do_split(img2)
    adding_img_number = len(completed_list)
    newImageInfo = (img.shape[0],img.shape[1])
    result_img = np.zeros(newImageInfo,np.uint8)
    for i in range(adding_img_number):
        result_img = result_img + gen_basic_building(completed_list[i])

    design = img/255*180 + result_img/255*75
    plt.imshow(design, cmap="gray")
    return design, result_img, completed_list

def gen_points_cloud_array(img, basic_building):
    """获得点云坐标的数组
    参数：img, basic_building
    返回值：points 数组(array:n*3)
    """

    newImageInfo = (img.shape[0],img.shape[1])
    all_white = np.ones(newImageInfo,np.uint8)
    all_white = all_white*255

    building_contour = cv2.Canny(basic_building,80,255)
    background_minus_building = img - basic_building

    all_white_minus_building = all_white - basic_building

    building_list = get_points(basic_building)
    building_contour_list = get_points(building_contour)
    background_minus_building_list = get_points(background_minus_building)
    all_white_minus_building_list = get_points(all_white_minus_building)

    points = []

    for j in range(len(all_white_minus_building_list)):
        if rd.randint(0,12) < 1:
            points.append([all_white_minus_building_list[j,0], all_white_minus_building_list[j,1],
  0])
```

```
    for i in range(1,80):
        if not(i % 3):#每三层添加一层点云
            for j in range(len(building_contour_list)):
                if rd.randint(0,5) < 1:
                    points.append([building_contour_list[j,0], building_contour_list[j,1], i])

    for j in range(len(building_list)):
        if rd.randint(0,12) < 1:
            points.append([building_list[j,0], building_list[j,1], 80])

    points=np.array(points)
    return points

def if_intersect(l1, l2):
    v1 = (l1[0] - l2[0], l1[1] - l2[1])
    v2 = (l1[0] - l2[2], l1[1] - l2[3])
    v0 = (l1[0] - l1[2], l1[1] - l1[3])
    a = v0[0] * v1[1] - v0[1] * v1[0]
    b = v0[0] * v2[1] - v0[1] * v2[0]

    temp = l1
    l1 = l2
    l2 = temp
    v1 = (l1[0] - l2[0], l1[1] - l2[1])
    v2 = (l1[0] - l2[2], l1[1] - l2[3])
    v0 = (l1[0] - l1[2], l1[1] - l1[3])
    c = v0[0] * v1[1] - v0[1] * v1[0]
    d = v0[0] * v2[1] - v0[1] * v2[0]

    if a*b < 0 and c*d < 0:
        return True
    else:
        return False
```

Code(Main part):

```
img = get_img_gray("C:/Users/iswzh/Documents/opencv/t/t (1).png")
design,result_img,completed_list = gen_design(img)
plt.imshow(design, cmap="gray")
```

修改记录

第一次修改记录：

原题目：Historical Block Design Generation Algorithm Based on GANs

修稿后题目：Historical Block Design Generation Algorithm Based on OpenCV

第二次修改记录：

统一修改所有过短和不准确的标题。

调整第三章的部分至第四章的 4.1.1 中。

修改第四章部分介绍。

增加多项实验 5.2.3，5.2.4，5.2.5 于 5.2 实验中。

修改部分流程图，使文字部分完全处于框图之内。

第三次修改记录：

原题目：Historical Block Design Generation Algorithm Based on OpenCV

修稿后题目：Design Generation Algorithm of Historic Building with Courtyard-style

修改了参考文献格式。

将所有乘号“*”改为了“×”。

修改了日文概要的文体。

修改了图标的文字环绕格式，去除了空太多行的问题。

修改了相关工作部分的引用格式。

记录人（签字）：王子浩

指导教师（签字）：刘

Acknowledgement

Time flies, and when I look back on my four years of university life again, I feel moved. When I am about to finish this thesis, I also felt relief.

Firstly, I would like to thank my advisor, Prof. LIU Bin, sincerely. His rigorous attitude and spirit of excellence have deeply educated me and infected me. He has been patiently guiding me in my research and study, providing me with numerous inspirations and opportunities. As a beginner in scientific research, I gradually gained experience, finding the direction and correcting my attitude under his enthusiastic guidance. For this graduation design, Prof. Liu chose the intersection direction of the architecture field and computer field for me so that I could accept the challenge in the unfamiliarity and continuously enhance my self-learning ability. I truly appreciate his trust and patience in guiding me along the way. In addition, Prof. Liu also helped me in my academic planning, treating me like his own son and showing me the way forward.

I want to thank Prof. GUO Fei and PhD. ZHAO Jun from the School of Architecture and Art for their professional and detailed knowledge in the field of architecture. During the thesis research process, they kept communicating with me. They patiently answered my questions as a layman in architectural design. In addition, PhD. Guo Fei enthusiastically collected lectures and study resources for me, providing me with precious learning opportunities; PhD. Zhao Jun worked tirelessly to prepare many worldwide datasets for the training model, ensuring the conduct of the experiment.

Thank you to all the teachers on the campus at the foot of Dahei Mountain for the past four years! Thank you to my roommates for spending these unforgettable years together.

Special thanks to my parents, who are far away from me at home. During my four years in college, I have had my ups and downs, and they have been behind me all the time. Their consistent love and support kept me from giving up hope when faced with choices, fears, and disappointments. They were the ones who accompanied me in my difficult and tough times and were the ones who cheered with me in my moments of joy and happiness. During these four years, they have given me the courage and reflection to become a strong and courageous person truly.

Finally, I would like to express my heartfelt gratitude to the authors of the works cited in this thesis and the expert teachers who took time out of their busy schedules to review this thesis.