applied sciences

MDPI

Article

Residential Buildings Complex Boundaries Generation Based on Spatial Grid System

Marko Lazić [1], Ana Perišić [1,*] and Branko Perišić [2]

[1] Faculty of Technical Sciences, University of Novi Sad, 21000 Novi Sad, Serbia; lazic.m@uns.ac.rs
[2] Center Novi Sad, Singidunum University, 11000 Belgrade, Serbia; bperisic@singidunum.ac.rs
* Correspondence: anaperisic@uns.ac.rs

Abstract: The automatic generation of building boundaries in contemporary research and engineering projects and practices is dominantly characterized by interior functional constraints. As a basis for the automated generation of various building boundaries, the solution presented in this paper is a novel approach that ignores the internal (functional) and focuses only on the external (non-functional) impacts. The primary orientation on external impacts may be, at any instance, extended by suitable complementary traditional methodology. The applied research methodology and presented method rely on a developed extendible rule-based system that simplifies floor plan creation by the recursive application of a formulated spatial grid generation algorithm. Based on starting parameter values (mainly the lot and building area spaces) the algorithm tends to create a set of grids that satisfy initial constraints by marking the individual grid cells as a part of the building or empty. The presented conceptual framework model served as a foundation for creating a prototype software application that supports the experimental generation of grid arrays that are transformed into readable images of residential building boundaries. For the initial validation of the developed methodology, method, and algorithm, the concrete parametric resolution is set to 1 m. The comparative analysis has shown that the presented approach overcomes some of the limitations of previous related research that generate building boundaries in simple rectangular form or with limited variability. The proposed method, in its current stage, outperforms discussed existing methods concerning complex shape boundary building plan generation. Besides that, there is a broad space for further enhancement directions concerning the interoperability with other, independently developed, frameworks, and software tools.

Keywords: computer graphics image generation; floor plan; rule-based algorithm; complex shape boundary of buildings

Citation: Lazić, M.; Perišić, A.; Perišić, B. Residential Buildings Complex Boundaries Generation Based on Spatial Grid System. *Appl. Sci.* **2022**, *12*, 165. https://doi.org/10.3390/app12010165

Academic Editor: Jürgen Reichardt

Received: 7 November 2021
Accepted: 20 December 2021
Published: 24 December 2021

1. Introduction

Designing a floor plan is one of the common tasks performed by architects. For thousands of years, this process was manually conducted and supported by different methods that consider the environment around the building, room types and sizes, room connections, as well as climate, culture, and other influencing rules. In recent years there is a great need for automation of this process considering the digitalization of this industry. This trend applies not just in the field of architecture, but also in the media and video game industries where flexible rule-based rapid model generation is essential.

Rapid generation of city environments consists of repetitive tasks which can be achieved more efficiently using procedural modeling techniques. These techniques are based on automatic or semi-automatic systems for procedural modeling such as shape grammar, cellular automata, L-systems, fractals, etc. These systems are often highly specialized for segmented design procedures using a narrow framework for a specific task.

The general specification of any system includes functional and non-functional requirements, where the latter addresses the constraints under which the former have to be

applied. The implementation of non-functional requirements is generally more demanding. The majority of scientific boundary generators are focused on the functional usage of the interior spaces usually resulting in simple building boundaries. The form of the building acts as a container for the interior space but it is not solely defined or influenced by it.

Functionality is not a key component in the digital and video game industry where residential building simulations mainly create a recognizable style. The contemporary design in real-world architecture directly influenced the research approach presented in this article.

In the presented approach, the focus is on the restrictions implied by external non-functional influencers, like contemporary building design, wind comfort, building-lot design, or even aesthetic factors and so on, as the driving force for shaping the designed residential building. This was the main reason why the adopted approach ignores the interior functionality in order to enable the rapid generation of building boundary variations that are compliant with the external restrictions, to which the interior functionality impact, as the other side of the same coin, may be added afterward (on demand).

This paper is focused on residential-type buildings which usually have more complex borders than other types such as office buildings, factories, supermarkets, schools, etc. In the verification phase of our methodology, we have used real examples of complex building boundaries.

The prototyped solution to the problem of automatic residential building border design by observing its floor plans as a grid system has been achieved by using a simple rule-based algorithm that accepts the area of the building as an input and generates complex shapes that represent different versions of residential floor plan border. The developed algorithm generates floor plans by observing buildings in the form of a two-dimensional array (grid) with respect to lot area and fixed building boundary size, through successive transformation steps leading to the completely connected cells model (see Figure 1).

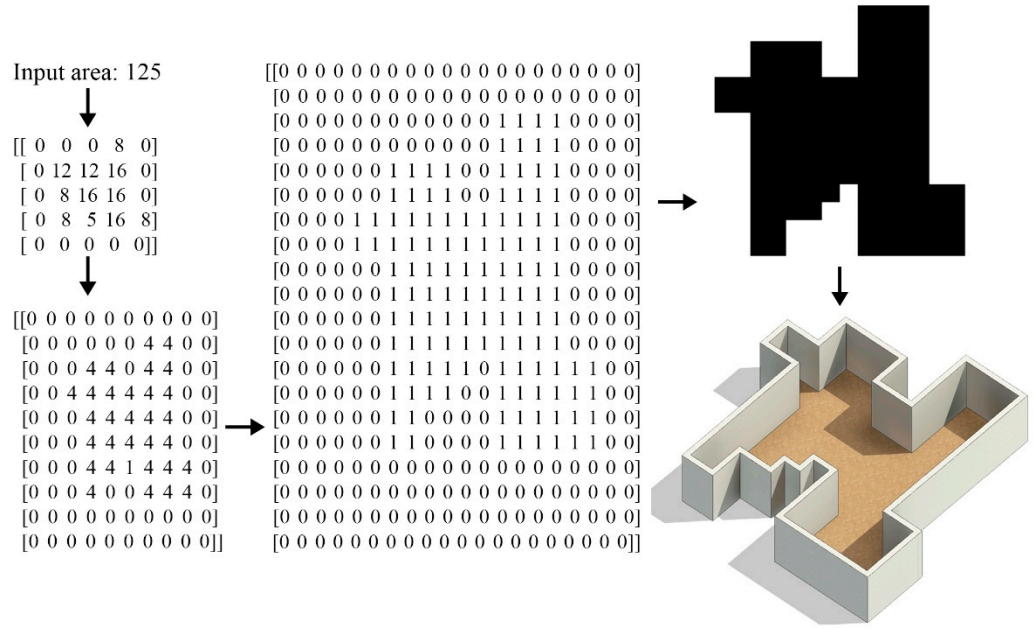


Figure 1. The illustration of an algorithm for one particular transformation starting with a 5 by 5 array (matrix), transformed to a 10 by 10 and finally a 20 by 20 array resulting in boundary rendered in 2D and also illustrated in 3D visual form.

The algorithm validity is checked via the application of prototyped software on a selected set of real residential building examples. The derived results have been used for algorithm adjustments and code refactoring that were used in order to raise the level of abstraction and enable the development of a software framework conceptual model that may

be used in further method, algorithm, and software tool refinements. The rest of the paper is organized as follows. Section 2 presents a brief survey of different approaches related to the research topic. In Section 3, a brief overview of methods and goals is presented together with the conceptual model of extendible software framework intended to serve as an umbrella for future improvements of methodology implementation refinements. Section 4 contains a detailed description of the algorithm and several representative examples that illustrate the way the algorithm is developed and refined. In Section 5 the cross-reference analysis of the proposed method and algorithm performance compared to related work references are presented. Section 6 addresses the discussion, and the possible directions of future research are presented in Section 7. The list of referenced literature is presented in the finishing part of the article.

2. Related Work

In the current literature, there are a number of proposed systems suitable for automatic building generation based on different geometry concepts. The layout of rectangular floor plans was introduced by Shekhawat [1]. Kozminski et al. [2] presented an algorithm for architectural floor plan generation using rectangular planar graphs. Bhasker et al. [3] proposed a similar approach using triangular planar graphs. This system was further developed by Wang et al. [4], who introduced an approach to generate floor plans from room adjacencies obtained from existing plans. All of these methods only generate rectangular rooms based on predefined (existing) graph structures.

Martin [5] has developed an algorithm to generate residential floor plans using the Monte Carlo method for room distribution. Flemming et al. [6] have applied an expert system for solving building layouts, introducing many serious disadvantages. As a consequence of the fact that the completeness and consistency of the system are not known to the user, it takes an intolerably lot of time to converge resulting in bad response time. Additionally, real building boundaries usually vary in shape and these variations are ignored in both Martin's [5] and Flemming et al.'s [6] approach. The boundaries are rectangular because they are derived from the internal constraints where all of the rooms have s rectangular shape, (which is not very common in contemporary architectural practice).

Del Rio-Cidoncha et al. [7] based the research approach on an expert system where optimization is performed through the use of artificial intelligence. Nauata et al. [8] developed a method for house layout generation using relational generative adversarial networks. The presented solution satisfies the majority of addressed problem aspects but generated building boundaries are dominantly simple, due to the fact that rectangular forms are derived from rectangular rooms in most discussed cases. For the generation of building layouts within a known boundary, Peng et al. [9] used a linear programming system of deformable tile templates. The exterior building boundary is the input while the interior layout is divided into tiles that are labeled as rooms. This is one of the research works that use the building boundary as an input that drives function within the object that, in a broader sense, shares a similar idea as the approach presented in this paper. Wu et al. [10] proposed a solution that automatically generates layout designs based on a MIQP (mixed integer quadratic programming) formulation.

Nonlinear programming techniques have been used for floor layout problems by Li et al. [11]. However, the solutions are derived for rectangular shape buildings only. Harada et al. [12] developed a model for interactive manipulation of layouts by the use of shape grammar. The results are within the simple form of building boundary and constant interactive input from a user is required. Wang et al. [13] proposed a framework for the automatic generation of floor plans using graph grammar formalism. The system can reproduce good results, but for each deviation from a rectangular building, interactive user input is required. Duarte [14] presented an interactive system for layout generation based on discursive grammar incorporating programming grammar and designing grammar. The described solutions include the 3D aspect and layout of the design space, but, due to the recognizable architectural style that has been analyzed, only produce rectangular

forms. Some of the researchers found the solution for house layout automation through the use of genetic algorithms [15–17]. Wu et al. [18] proposed a data-driven method for generating floor plans for residential buildings. They utilized a dataset of floor plans of real buildings in order to construct a neural network for layout generation. This research shows that, with extensive layout database and adequate neural network training, the problem of automatically generation of building layouts can be solved using existing boundaries as an input.

Merrel et al. [19] used a Bayesian network trained on real-world data to design an architectural program that generates floor plans using stochastic optimization. The described solution enables the generation of simple and complex building forms, but with low levels of complexity. The basic approach, described in this paper, is focused on the creation of very complex forms instead. That is why the former related work reference is used for the comparative analysis of derived results, presented in detail in Section 5.

Hua [20] proposed a method for automatic construction of irregular floor plans from the graphical patterns as an input where the accuracy of the input area of the rooms in the described results is not of great importance. The boundary is derived from the input image of an object shape. Al Omani [21] used natural graphic images to generate a building layout plan. Bao et al. [22] developed a method for exploring different building layouts based on simulated annealing.

Procedurally generating building shapes with grid-based constraints based on the cellular automata approach is widely adopted in the field of architectural and urban design based on the cellular automata approach. Patterns generated by cellular automata systems are used in architectural design for form exploration [23–26]. Several methods have been developed based on cellular automata systems. Anzalone and Clarke [27] developed a system for the generation of 3D buildings based on Conway's Game of Life system [28]. Araghi and Stouffs [29] use cellular automata for the generation of high-density residential buildings, addressing density, accessibility, and natural light. Some researchers explored cellular automata generation results as the first step and developed architecture form enveloping some cells or the entire form [30,31].

However, the application of cellular automata systems in the domain of architectural design is faced with several limitations. According to [26,32,33], the degree of complexity for even simple architectural outcomes suggests that generating a complete architectural design in this manner is resource-demanding and, in general, probably even not possible. The main challenge is to find the solution that preserves an acceptable balance between complexity in architectural design and formal rules of cellular automata systems.

Summarizing the current state-of-the-art analysis it is possible to conclude that there are several approaches used as a foundation to specification, modeling, and design of tools for automated generation of building boundaries, with emphasis on 2D or 3D building geometry layout inclusion.

The comparative analysis of related work relevant to this article topics, with respect to selected key factors (dimensionality, interior layout inclusion, and boundary type) is presented in Table 1.

The analyzed results, however, show that the prevalent trend mainly focuses on the simple boundary type forms because either complex forms appeared challenging for the particular approach or they were not considered at all.

Table 1. The comparative analysis of related work relevant to this article.

Research	Method/Problem That Is Addressed	Dimensionality	Interior Layout	Boundary Type
Shekhawat [1]	Spiral based layout generation	2D	Yes	Rectangular
Kozminski et al. [2]	Algorithm for finding a rectangular dual of a planar triangulated graph	2D	Yes	Rectangular
Bhasker et al. [3]	Algorithm to construct a rectangular dual of an n-vertex planar triangulated graph	2D	Yes	Rectangular
Wang et al. [4]	Algorithm for floor plan generation from existing floor plans based on graph transformations	2D	Yes	Rectangular
Martin [5]	Algorithm for procedural house generation based on the Monte Carlo method (one example without the code or detailed method description)	2D	Only as a predefined rectangular shape	Complex, derived from rectangular rooms
Flemming et al. [6]	Expert system for solving building layouts based on design grammars	2D	Only as a predefined rectangular shape	Complex, derived from rectangular rooms
Del Rio-Cidoncha et al. [7]	Expert system based on route search as a problem of AI optimization	2D	Yes	Rectangular
Nauata et al. [8]	Relational generative adversarial neural network for graph-constrained house generation	2D	Yes	Simple, derived from mostly rectangular rooms
Peng et al. [9]	Layout computation using deformable templates	2D	Yes	Boundary is an input
Wu et al. [10]	Algorithm based on mixed-integer quadratic programming formulation	2D	Yes	Simple, but also Boundary can be an input
Li et al. [11]	Nonlinear programming techniques for floor layout generation	2D	Yes	Rectangular
Harada et al. [12]	Mixed continuous/discrete models where user can manipulate objects within constrains	2D	Yes	Simple, derived from mostly rectangular rooms
Wang et al. [13]	Algorithm that uses reserved graph grammar formalism	2D	Yes	Simple with required input for each deviation from rectangle form
Duarte [14]	Discursive grammar that consists of programming and a designing grammar	3D	Yes	Rectangular, because of the architectural style
Wu et al. [18]	Data-driven method for generating floor plans where building boundary is an input	2D	Yes	Boundary is an input
Merrel et al. [19]	Bayesian network trained on real-world data	3D	Yes	Mostly simple, with limited complexity
Hua [20], AlOmani [21]	Irregular architectural layout synthesis with graphical inputs	2D	Yes, with errors in the output area	Irregular, from pattern
Bao et al. [22]				
Anzalone and Clarke [27]	Complex adaptive systems applied to architecture form	3D	No	Mostly simple
Araghi and Stouffs [29]	Cellular automata-based algorithm for 3D building generation	3D	No	Complex, but with low resolution (10×10 m cells)

3. Overview

From the methodology point of view, the starting point of the research, presented in this paper, was to derive the foundations of an extendible rule-based framework model that may be used to derive either formal procedures or a family of software tools addressing the problem of building boundary generation.

There are two main framework features that have to be explicitly addressed, extendibility and persistency.

Extendibility is the most important characteristic of any formal procedure or software system. It means that new concepts may be introduced at any instance of time without

impact on any previously added (existing) concepts. It is especially important in software engineering where it is known as an Open/Close quality design principle [34]. In our case, it is essential to support the extendibility of applied strategy (Inner-Bounded, Outer-Bounded, or even the combination of them), method (the collections of rules), and algorithm (rule-based limited set of steps needed to reach an arbitrary problem solution).

The initial conceptual framework model, in the form of an object-oriented diagram, is presented in Figure 2.

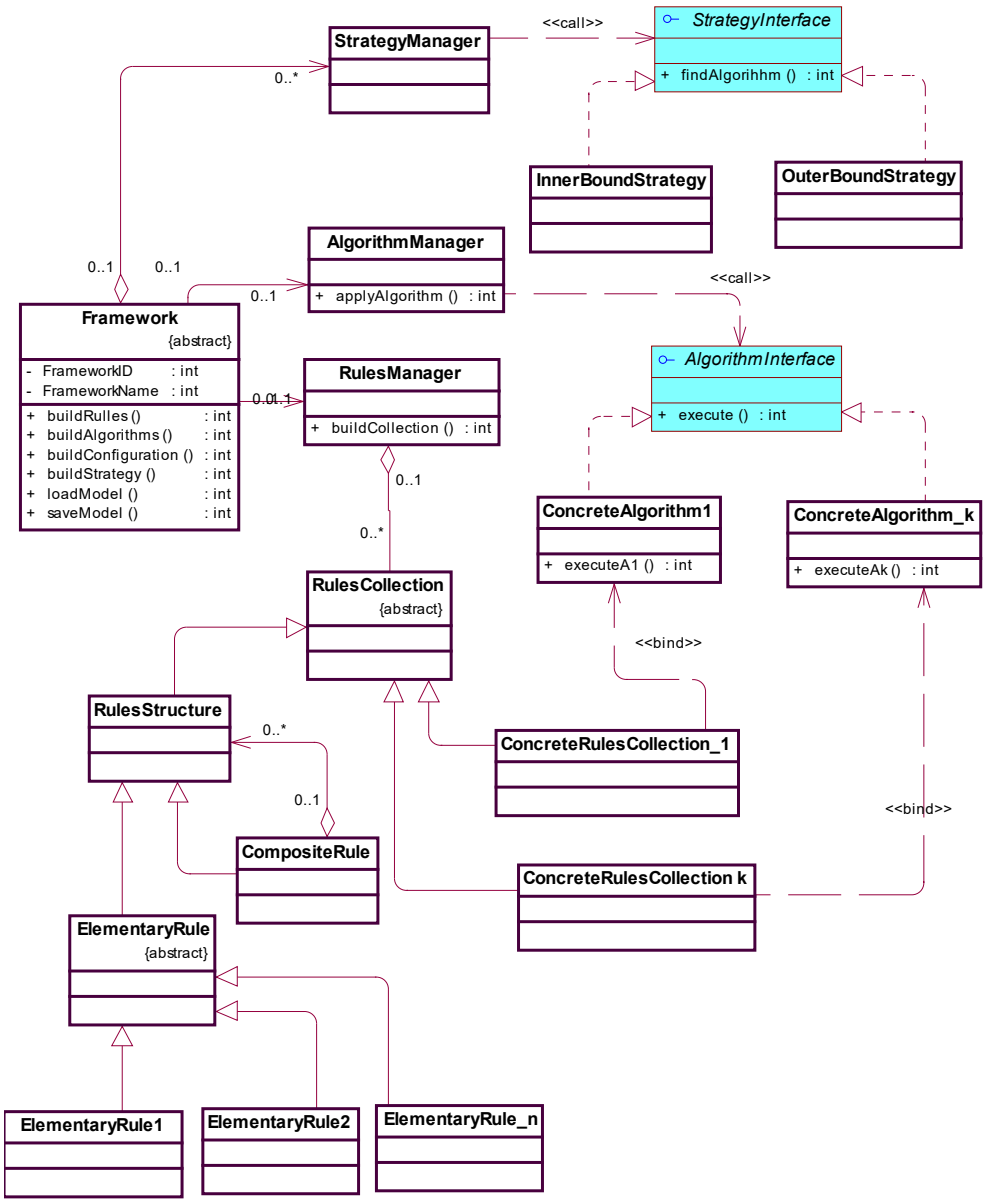


Figure 2. The initial framework conceptual model.

The Framework is a concept to which the interoperability concerns are allocated. It delegates the extendible point handling to StrategyManager, AlgorithmManager, and Rules-Manager components with the extending points: Strategy Interface, Algorithm Interface, and ElementaryRule fabric respectively (see Figure 2).

The explicit dependency between each ConcreteAlgorithm and corresponding Concrete-RulesCollection has to be maintained in order to enable the analysis of algorithm-rules correlation (see Figure 2).

The other essential point of framework specification is the way of preserving concrete products (generated set of building boundaries) joined with a particular instance of Strategy, Algorithm, and Parameter combination data.

The initial conceptual framework model from the persistency aspect is presented in Figure 3 in the form of an object-oriented diagram.

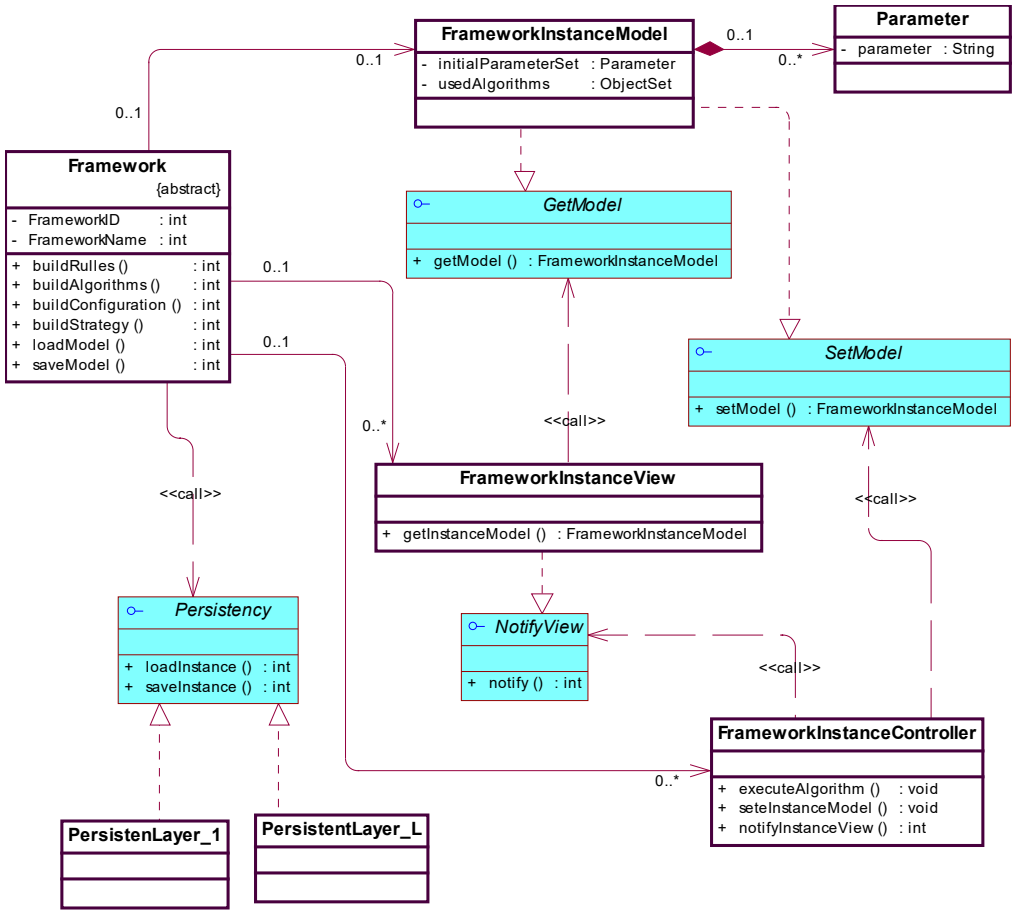


Figure 3. The initial framework conceptual model—persistency aspect.

The role of the Framework concept is the same.

The Persistency interface concept enables the extendibility of vide variety of persistent forms and structure handlers used to store a large volume of data usually created in the particular processing. It isolates the persistent layer from its dynamic management.

The Framework delegates the internal data structure concerns to the FrameworkInstance-Model concept. It is loaded and saved through the Persistency interface methods implementation.

The Framework delegates the visualizations concerns of internal data structure to the FrameworkInstanceView concept and data structure transformation actions concerns to the FrameworkInstanceControler concept.

The role of GetModel, SetModel, and NotifyView interfaces is to relax dependences of FrameworkInstanceModel, FrameworkInstanceView, and FrameworkInstanceCon-troler concepts.

The method used in the proposed solution receives the area of a building as an input in the form of two-dimensional grid representation (particularly 1×1 m cells) and varies available algorithms in order to, if possible, generate corresponding building boundaries that satisfy initial constraints (lot and building size) preserving the grid cells interconnection principle. This approach is chosen to initially simplify the overall shape thereby generating particular solutions that:

- can be, for verification purposes, easily compared with other methods

- are suitable for further enhancements and development in the domain of building's layout design.

Boundary shape can be either rudimentary with a small number of variations, or quite complex just as is the case in the real examples examination. The primary focus of the proposed method is the focus on complex building boundaries. As the boundary complexity directly influences algorithm complexity and the time for analytical calculation, the method is exercised through the application of different spatial grid transformation algorithms, described in the algorithm section of the article. These variations had a direct impact on criteria that restricts the number of lines composing the particular floor plan. This calculated number is used as a metric for comparative analysis and the evaluation of the proposed method with regards to other published solutions.

The resolution of 1 m of the final image was chosen because in related work research smaller cells of the system caused the generation of buildings that are too complex compared to real buildings and larger cells composed results that are simple and similar to related work findings.

This paper method is applied to building boundary generation for a 20 × 20 m lot size because typical residential buildings are in this category. Expanding the boundaries of the lot did not improve the algorithm in the experimental phase of this research.

The rule-based system is chosen because it allows the writing of algorithms that are reasonably fast for a rapid generation. They are also adjustable to better suit the needs for the task of creating images that represent building boundaries. Through an extensive trial and error process, the rules for the proposed method are formulated and described in the algorithm section of this paper.

The method described in this paper is derived in a way that it can be compared with the real buildings of complex boundary shape to confirm that there is a need for such a generator. Next, the method is compared with different results of other methods described in the related work section and the discussion section of this article.

4. Algorithm

The general algorithm used to illustrate and refine the proposed method is presented in Figure 4.

In the rest of the section, there is a detailed explanation of the steps that have been used in the algorithm and method refinement process.

The shape of a building layout L is defined by the label of the array with 20 columns and 20 rows. Each cell of the array represents space of 1 m by 1 m that can be either part of the building—B or part of the empty space around it—E. The number of cells that represent built space N is the value that corresponds to the desired area of the house. Value N is the area of the building in square meters. A valid shape should satisfy two constraints first, all B labels are connected into a single shape; second, there are no E labels inside layout L.

$$\text{Area}(L) = N \tag{1}$$

The algorithm starts from the simplest case where the floor plan is represented as an array of five columns and five rows where random values describe the overall shape. Next, the transformation of the results into a 10 by 10 matrix with specific rules that are applied. Finally, from previous results, a 20 by 20 array is formed. In this final step sets of rules are applied.

The algorithm can be applied for generating buildings that are square-shaped and have a maximum area of 400 square meters. It can give results for houses that are in the range of 40 to 360 square meters.

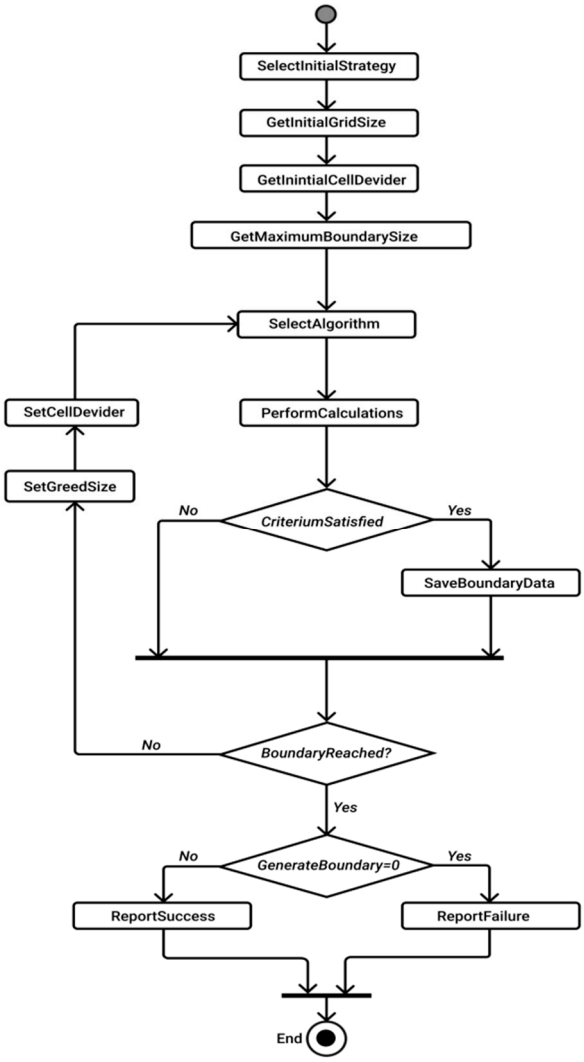


Figure 4. The general algorithm.

4.1. Generation of 5 by 5 Array

The first stage of the method generates a random array using a random number generator. The only input into the algorithm is the desired area of the building. In the first stage, that number is divided into a 5 by 5 array.

In order to get valid results, new rules are applied. Every cell of the 5 by 5 array will be transformed into 16 cells in the final 20 by 20 solution. The first rule applied states that any cell cannot have a value greater than 16 or less than 0. If one cell of this array is $c_{(x,y)}$ then we can use the expression:

$$0 \leq \forall c_{(x,y)} \leq 16 \tag{2}$$

The next constraint is that the middle cell of the array $Lf_{(2,2)}$ is always 16. This means that in the final array the middle of the layout is always labeled as part of the building (expression (3)).

$$\begin{bmatrix} c_{(0,0)} & c_{(1,0)} & c_{(2,0)} & c_{(3,0)} & c_{(4,0)} \\ c_{(0,1)} & c_{(1,1)} & c_{(2,1)} & c_{(3,1)} & c_{(4,1)} \\ c_{(0,2)} & c_{(1,2)} & \mathbf{16} & c_{(3,2)} & c_{(4,2)} \\ c_{(0,3)} & c_{(1,3)} & c_{(2,3)} & c_{(3,3)} & c_{(4,3)} \\ c_{(0,4)} & c_{(1,4)} & c_{(2,4)} & c_{(3,4)} & c_{(4,4)} \end{bmatrix} \tag{3}$$

As the consequence, in the final 20 by 20 array, not all corners will be labeled as part of the building. In the 5 by 5 array the corner cells are as $C = \{C_1, C_2, C_3, C_4\}$ and 8 middle cells as $M = \{M_1, M_2, M_3, M_4\}$ as in showed in expression (2).

$$\left[\begin{array}{c|cc|c|c|cc|c} C_1 & \cdots & \cdots & M_1 & \cdots & C_2 & \cdots & \cdots \\ & \cdots & \cdots & & \cdots & & \cdots & \cdots \\ \hline M_2 & \cdots & \cdots & \multicolumn{3}{c|}{\mathbf{16}} & M_3 & \cdots & \cdots \\ \hline C_3 & \cdots & \cdots & M_4 & \cdots & C_4 & \cdots & \cdots \\ & \cdots & \cdots & & \cdots & & \cdots & \cdots \end{array}\right] \tag{4}$$

Their corresponding values are C_v for the sum of C_1, C_2, C_3, and C_4 and M_v for the sum of M_1, M_2, M_3, and M_4 = Relationship of values of all corner cells C_v and middle cells M_v is defined by the formulas:

$$C_v = \lfloor N * 0.6 - 15 \rfloor \tag{5}$$

$$M_v = N - 16 - C_v \tag{6}$$

where N represents the area of the building. The next step was to get unique random solutions for the layout. This is solved by the generation of four random real numbers r_1, r_2, r_3, and r_4 where $\{r_n \in \mathbb{R} \mid 0 \leq r_n \leq 1\}$ which is applied in order to get values of cells in four corners of the array. C_{v1}, C_{v2}, C_{v3}, and C_{v4} are values of cells C_1, C_2, C_3, and C_4 respectively. Their value is calculated by the formula:

$$C_{vn} = \left\lfloor C_v * \frac{r_n}{\sum_{i=1}^{4} r_i} \right\rfloor \mid n = \{1, 2, 3, 4\} \tag{7}$$

This will determine where the most cells labeled as built are in the final solution. If C_{vn} is bigger than 49, then the value is 49 and the difference is added to the other cells. The next step is to fill C_{vn} values into corner cells of a 5 by 5 array. This is accomplished by forming a set of 4 numbers $\{c_{vn1}, c_{vn2}, c_{vn3}, c_{vn4}\}$ which sum is the value of the cell C_n. They are calculated by the rules in the equation:

$$[c_{vn1},\ c_{vn2},\ c_{vn3},\ c_{vn4}] = \begin{cases} [C_{vn}, 0,\ 0,\ 0]\ if\ C_{vn} \leq 16 \\ [16,\ C_{vn} - 16,\ 0,\ 0]\ if\ C_{vn} \leq 32 \\ [16,\ 16,\ C_{vn} - 16,\ 0]\ if\ C_{vn} \leq 48 \\ [16,\ 16,\ 16,\ C_{vn} - 16] otherwise \end{cases} \tag{8}$$

This list is then transformed so the biggest values are closest to the middle of the array, and the lowest is at the corners. The other two cells are randomly placed in the other two places for C_n values. This is illustrated by the next formula:

$$\begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \mathbf{16} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \rightarrow \begin{bmatrix} c_{v14} & (c_{v12}\ \mathrm{V}\ c_{v13}) & \cdots & (c_{v22}\ \mathrm{V}\ c_{v23}) & c_{v24} \\ (c_{v12}\ \mathrm{V}\ c_{v13}) & c_{v11} & \cdots & c_{v21} & (c_{v22}\ \mathrm{V}\ c_{v23}) \\ \cdots & \cdots & \mathbf{16} & \cdots & \cdots \\ (c_{v32}\ \mathrm{V}\ c_{v33}) & c_{v31} & \cdots & c_{v41} & (c_{v42}\ \mathrm{V}\ c_{v43}) \\ c_{v34} & (c_{v32}\ \mathrm{V}\ c_{v33}) & \cdots & (c_{v42}\ \mathrm{V}\ c_{v43}) & c_{v44} \end{bmatrix} \tag{9}$$

By these steps, only the middle cells of M are left to be filled. The algorithm was formulated to firstly fill the middle cells that are surrounded by at least two 16 cells. In that case, the cell is filled with value 16. Secondly, if the values of the cells orthogonally 1 cell away are bigger than 32, if other conditions are met, then the cell value is 12. The rest of the middle cells are filled with the number 8 until the sum of the array is the same as the input number of built cells N.

There is a possibility that the sum of the array is less than N. The difference is calculated and represented as D. In that case, the rest of the sum is split into values depending on the value D and the number of possible cells to be filled. These values are filled in cells

which value is zero. Finally, if the values of the middle cells M that are in the first or the last column or row $m_{15\text{-}i}$ not zero, and the values of the middle cells M that are connected to them $m_{34\text{-}i}$ are not 16, then values of these cells are recalculated by the rule:

$$m_{15-i} > m_{34-i} \rightarrow \begin{cases} m_{34-i} = 16 \; and \; m_{15-i} = 16 - m_{15-i} + m_{34-i} \; if \; m_{15-i} + m_{34-i} \geq 16 \\ m_{34-i} = m_{15-i} + m_{34-i} \; and \; m_{15-i} = 0 \; if \; m_{fl-i} + m_{sf-i} < 16 \end{cases} \tag{10}$$

An example is shown in expression (11). Value $m_{15\text{-}i}$ = 12 and value $m_{34\text{-}i}$ = 5. In this case, $m_{15\text{-}i} > m_{34\text{-}i}$, and therefore new values are recalculated. Without this step, empty cells would appear inside the borders of the building.

$$\begin{bmatrix} \cdots & \cdots & 12 & \cdots & \cdots \\ \cdots & \cdots & 5 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \rightarrow \begin{bmatrix} \cdots & \cdots & 1 & \cdots & \cdots \\ \cdots & \cdots & 16 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \tag{11}$$

By these rules, all problems are solved and the next step is transforming the 5 by 5 array to a 10 by 10 array using different rules.

4.2. Generation of 10 by 10 and 20 by 20 Arrays

This array is the middle step between generating the first sketch and the final array. Every cell value is split into four values as shown in the array:

$$\begin{bmatrix} c_{(0,0)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(1,0)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(2,0)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(3,0)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(4,0)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} \\ c_{(0,1)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(1,1)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(2,1)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(3,1)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(4,1)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} \\ c_{(0,2)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(1,2)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(2,2)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(3,2)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(4,2)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} \\ c_{(0,3)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(1,3)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(2,3)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(3,3)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(4,3)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} \\ c_{(0,4)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(1,4)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(2,4)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(3,4)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} & c_{(4,4)} \rightarrow \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} \end{bmatrix} \tag{12}$$

Isolating one cell from the 5 by 5 array transformation can be presented as:

$$c_{(0,0)} \rightarrow \begin{pmatrix} c_{x(0,0)} & c_{x(1,0)} \\ c_{x(0,1)} & c_{x(1,1)} \end{pmatrix} \tag{13}$$

In this case value of the cell $c_{(0,0)}$ is equal to the sum of values $c_{x(0,0)}$, $c_{x(1,0)}$, $c_{x(0,1)}$, and $c_{x(1,1)}$. If the value of the 5 by 5 cell is zero then four values corresponding to this value in a 10 by 10 array are 0. Similarly, if the same value is 16, then all four values are equal to 4. When the value is between 0 and 16 first step is to split the number with the rule then no 10 by 10 cell can have a value greater than 4. In order to have a solution without empty cells in the middle of the grid, cells values are preferably 4 or 0. Only one cell out of four per transformation can have a value that is different than 0 or 4.

A list of four values is then applied in the 10 by 10 array based on the surrounding cells. If the values of cells around the observed cells are 16 and 0, then minimal values from the list are positioned close to other 0 values, and maximum values of the list are positioned close to cells with value 16. An example of this method is presented in Figure 5.

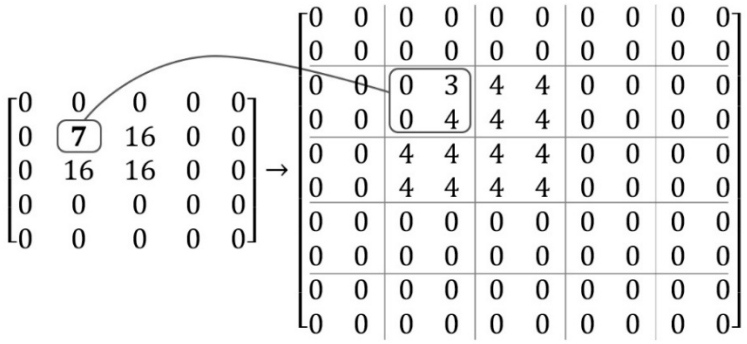


Figure 5. Transformation of the 5 by 5 array into a 10 by 10 array. In this example position of number 7 in the left array dictates the position of a list of numbers [0, 0, 3, 4] in the right array. The position of number 4 is conditioned by the surrounding cells to be next to the highest surrounding cell.

The next step is the transformation of a 10 by 10 array into a 20 by 20 array by simple transformation rules. This method for generating the final array has rules similar to the rules from the previous chapter. Every cell of 10 by 10 array is transformed into 4 cells of 20 by 20 array. A value of 0 means all cells are zero. A value of 4 means that all four cells have a value of 1. Values between 0 and 4 are written as a list of four numbers that are either 0 or 1. The sum of the list is equal to the value of the cell from the 10 by 10 array. Similar to the solution shown in Figure 5, minimum values are placed near adjacent cells with a value of 0, and maximum values are placed near cells with a value of 4.

The algorithm was designed to validate the final array where solutions that have empty cells in the middle of the building, solutions that are split in more than one building, and solutions that have a different sum of the array from the input number are excluded. The last step is labeling cells with a value of 1 as built and others, with a value of 0, as empty. Solutions are represented as an orthogonal grid of pixels and the image is created as presented in Figure 6.

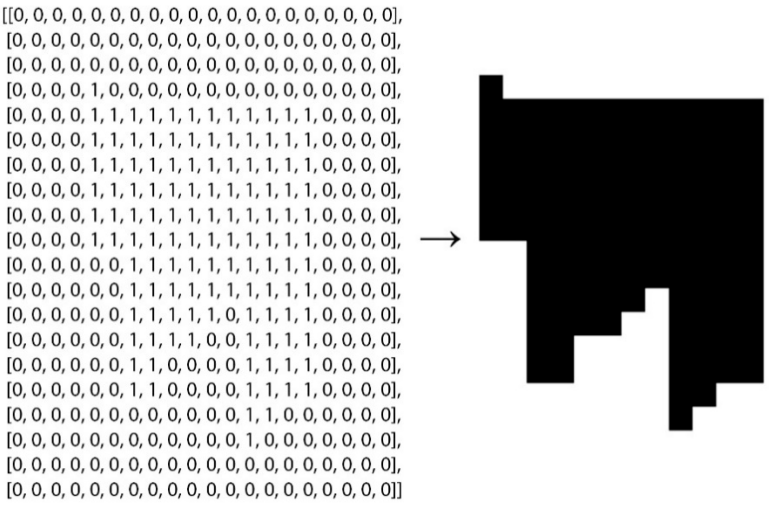


Figure 6. Forming the shape of the building from the array. Cells that are labeled 1 are presented as black squares and cells that are labeled 0 are white squares.

5. Results

The prototype implementation of the algorithm is in Python language. All tests are done on a 3.1 GHz Intel Core I5 with 8 GB RAM. The inputs to the algorithm are numbers that represent the area of the building and the number of different generated solutions. Outputs of the algorithm are images of the layout and arrays exported in text form. In Figure 7, the results are illustrated by randomly generated layouts from using the method described in the previous chapter.

Figure 7. Examples of random shapes of buildings generated by our method. Buildings in this example have their floor plan area from 50 square meters (first) to 350 square meters (last).

This method is evaluated by producing 100 different building layout shapes for each area input. Input numbers that represent the area of the building in square meters are values from 50 to 350 by increasing the number by 25 for overall 13 different inputs. The average time for this task is calculated and the results are presented in Figure 8. For most of the input values, N calculations of 100 valid layouts takes between 2 and 3 s. For the bordering values of N, the time for calculation is higher.

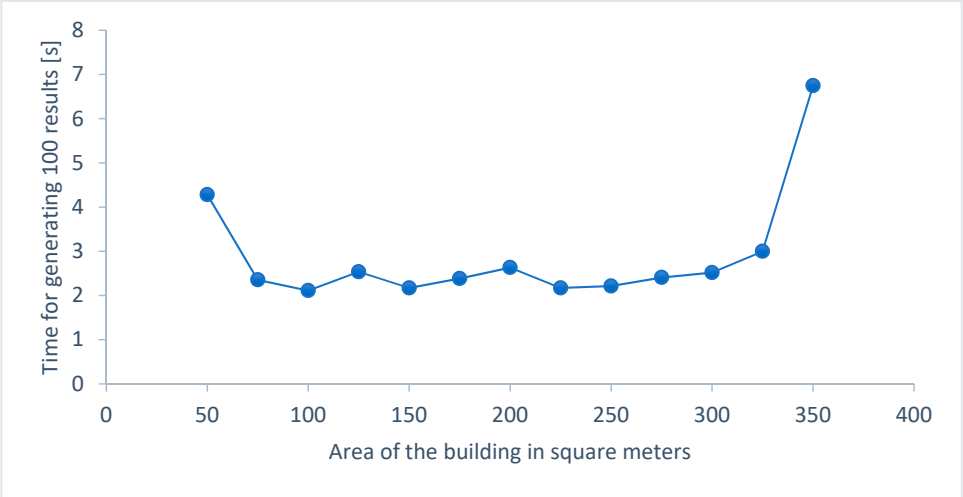


Figure 8. Performance of the method. The graph presents the correlation between the input parameter that represents the area of the building and the average time for generating 100 results. Performance is worse for areas that are lesser than 75 or greater than 325.

Results are measured by the number of edges necessary to draw the border of the building. Borders of the buildings generated by our method show that the number of edges can be a minimum of 12 and a maximum of 54. The distribution of the results is presented in Figure 9. The average values vary corresponding to the input parameter of the building area.

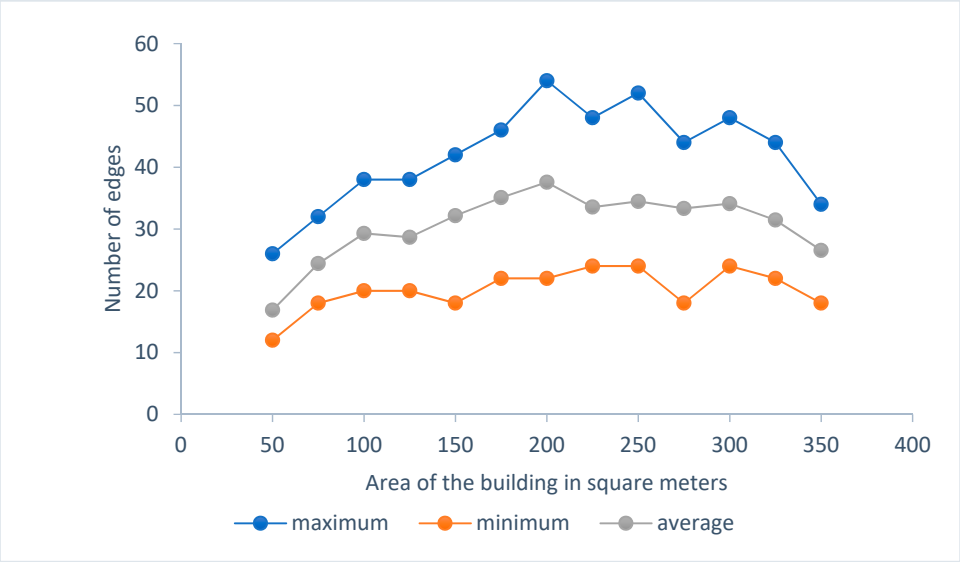


Figure 9. Efficiency of our method based on the number of edges that are necessary to draw the shape of the building boundary. The minimum value that we generated was for a 50 square meter building with a value of 12. The maximum value for a 200 square meter building was 54.

The results are compared to the results of other methods [10,13,18,19] and real buildings with complex borders by the same criteria. Building floor plans are converted into a grid of the 1 by 1 m squares through the simple algorithm that is developed in order to have uniform results. The number of edges before and after conversion is the same except in some cases the area of the buildings changed no more than 5%, which we considered tolerable.

First, the described method is compared to the research by Merrell et al. [19]. Only complete computer-generated solutions are considered. The maximum value of edges is 18 for the house with an area of 152 square meters. Figure 10 shows the distribution of the results.

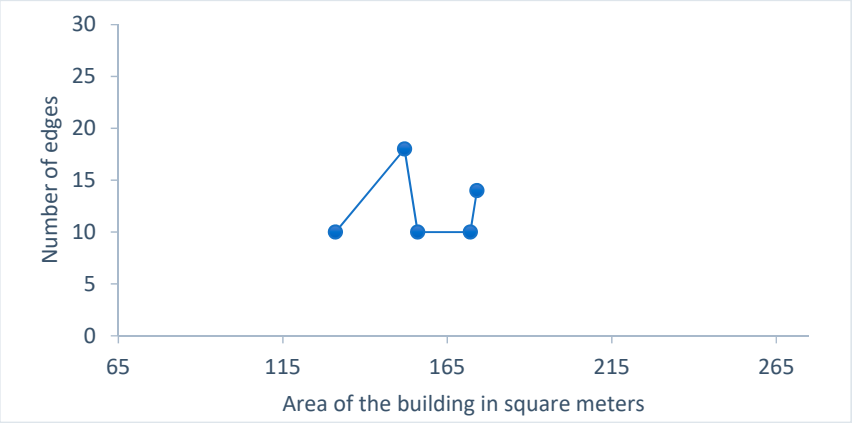


Figure 10. Efficiency of the method by Merrell et al. [19], observed based on the number of edges that are necessary to draw the shape of the building boundary. Values in this research range from 10 to 18.

Next, the results from the research of Wang et al. [13] are evaluated area of buildings in this research is between 85 and 155 square meters. The highest number of edges in this paper is 14 and the average value is 10. Results are presented in Figure 11.

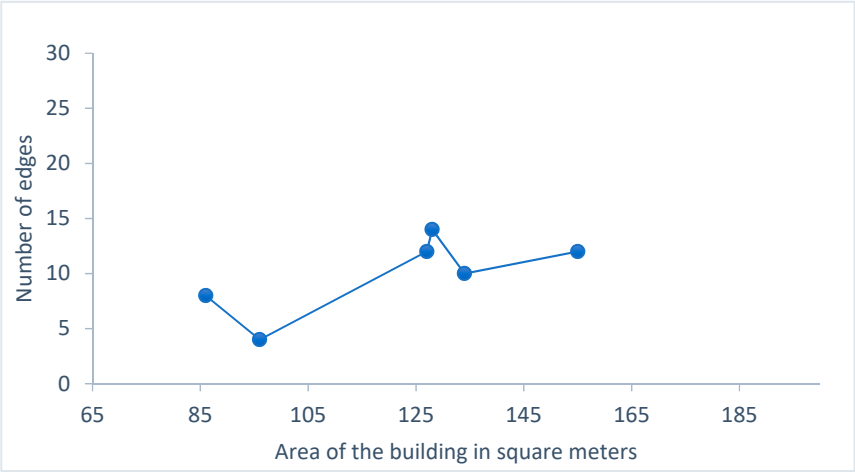
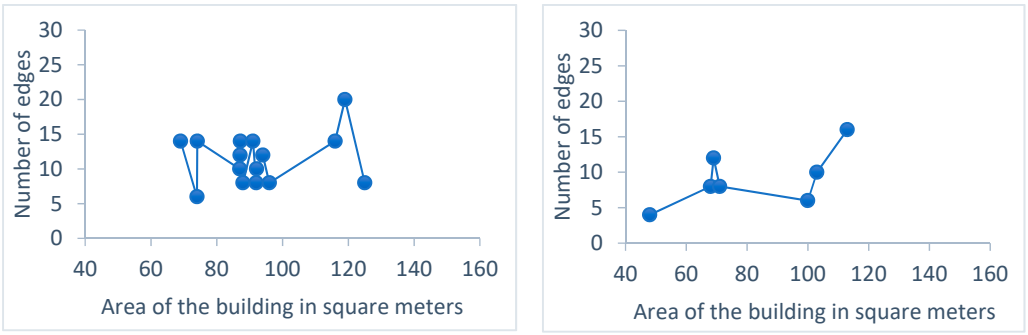


Figure 11. Efficiency of the method by Wang et al. [13], observed based on the number of edges that are necessary to draw the shape of the building boundary. Values in this research range from 4 to 14.

The research of Wu et al. [10,18] can be observed from two aspects. The first aspect is the analysis of the floor plans from traced from the real world [18] where examples are shown in area range of 75 to 125 square meters. The maximum value is 20, and the average is 11.46 edges per building. The second aspect is the analysis of the MIQP-based computer-generated layout [10]. This research presented a similar area range as previously mentioned, but the maximum value of edges is 16, and the average is 7. Results are shown in Figure 12.



(**a**) (**b**)

Figure 12. Efficiency observed based on the number of edges that are necessary to draw the shape of the building boundary. (**a**) Results of real-life buildings in the research paper by Wu et al. [13] show high variety with 11.46 as the average number of edges. (**b**) Results of the method by Wu et al. [13] have a value of 7 for the average number of edges.

Finally, the Internet databases were searched for the real buildings that have complex border shapes in order to make the comparison with the described algorithm. In this process floor plans of buildings with a simple form that can easily be found in different databases were eliminated. The focus was on the buildings in the area range similar to our research results from better comparison. In Figure 13 examples of real building floor plans are shown. The range of an area is from 50 to 350 square meters. The maximum value is 34 and the average value is 24.73.

Figure 13. Border shapes of our selection of complex real-life buildings.

All the results are compared and presented in Figure 14. Differences in the results of different method applications can be observed based on the parameter of the number of edge counts. The overall average of edges per building is 24.73 for selected real buildings, 28.88 for the method described in the paper, and 10.33 for other methods.

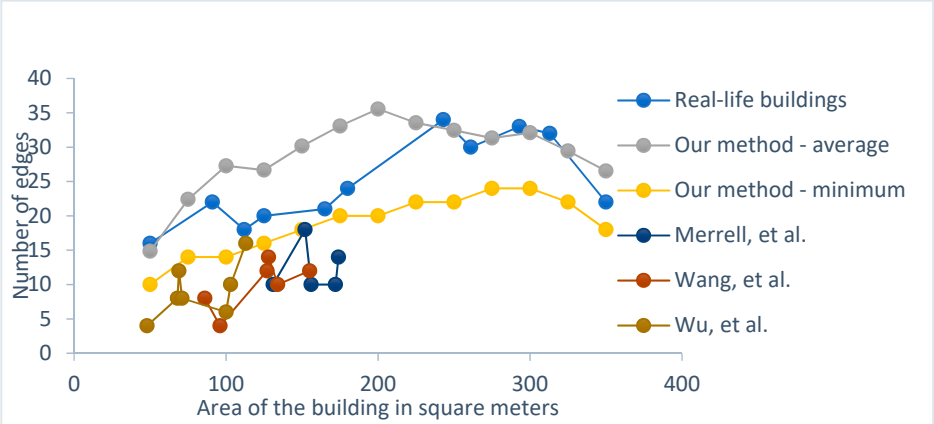


Figure 14. Comparison of our method to all methods selected from research [10,13,18,19] and to our selection of real-life buildings with complex building floor plan borders.

6. Discussion

The problem of building boundary generation is widely addressed in contemporary research and publications. In the referenced publications discussed methodologies, methods, algorithms, and more or less automated frameworks and software tools tend to cluster into coherent groups that use the related principles but with different impacts on the abstraction level and generality of proposed solutions.

One cluster is mainly oriented to simple regular building boundaries development that solely serves as the container of internal. They are usually constrained by the internal functionality disposition. For this group, the main focus appears to be the building floor plan layout or the architectural style they analyzed observed is not contemporary or visually challenging by the form. A few of those researches argue the possibility of combining simple building boundary shapes to get more complex variations or offer low-resolution results, but most of them are significantly limited in the number of variations achieved in a specific time frame. The external constraints are more or less ignored and do not represent a driving force of the automatic building boundary generation process.

The other cluster, with a significantly lower number of research works, focuses on relatively complex building boundaries generation that makes them suitable for cross-reference performance comparison with the approach described in this article. The results of three research works from this group are compared with regard to the presented novel approach. They were selected because their algorithms produced results that are more complex compared to others, listed in Section 2 making them an ideal validation benchmark.

The smallest research cluster uses external boundary constraints as the driving force of building boundaries generation but different methods used (knowledge-based system) were the main obstacle in strict comparisons.

There were no research works seriously focusing on the methodology, methods, and algorithms that support the production of complex boundary shapes in large quantities within a reasonable time frame.

The results of the research methods of the papers used as benchmarks for validation [10,13,19] are derived from a maximum of a 174 square meter building area. The results for larger buildings could not be obtained from the current examples. The algorithm presented in this paper produces results for buildings up to 350 square meters. The possibility of other methodologies generating similar results is currently highly unlikely but may be possible in further development.

The main obstacle of the methodology presented in research papers [10,13,19] was the heavy internal functionality interrelations and dependencies that are computationally demanding. That is why the inherent limitation of variations is understandable.

The goal of this paper is to create the building boundaries in the form of constraints influenced by outer forces but not to completely exclude the premises of interior functionality either. There are research works [9,18] that can produce building layouts if the border constraint of the building is provided. The expanded version of the methodology applying grid systems, used in this paper, is also planned to be upgraded for the internal layout generation in the future.

Figure 11 summarizes the advantages and disadvantages of the benchmark set with regard to the proposed method and algorithm. Almost all of the compared research methods produce results that are below the minimum value of the algorithm presented in this paper. The results clearly show that the novel method, proposed by this paper, fills in the current research gap in the referenced scientific field. The area of interest is a specific type of building boundary generator with a capacity for creating numerous design variations. The uses for these results are numerous in the specific fields of architecture, gaming, generative design, etc.

For example, the most common of the real-life complex building boundary has borders with the number of edges that are mostly between the minimal and average results of our algorithm. The analyzed buildings are not average but rather common examples of the real-life set mainly because the basic rectangular and simple shape boundaries are more commonly seen than the complex ones. Although not such common, buildings with complex boundary shapes are the mainstream of the presented approach.

The internal function-based approach usually results in basic and simple shapes of boundary layers. On the contrary, contemporary architecture is sometimes strongly influenced by outer factors like adapting to the force of the wind or the increase of daylight

insulation or even dominant visual message making the building a recognizable environmental marker.

Concerning system sciences, the general specification of any system includes functional and non-functional requirements, where the latter addresses the constraints under which the former have to be applied. The implementation of non-functional requirements is generally much more demanding. The trend of automatic generation of residential buildings applies not just in the field of architecture, but also in the media and video game industries where flexible rule-based rapid model generation is essential. The is that the functionality is not a key component in the digital and video game industry residential building simulations. The emphasis is on the recognizable visual style and game narrative. The form of the building acts as a container for the interior space but it is not solely defined or influenced by it.

This was the main reason to ignore the interior functionality in the proposed approach in order to enable the rapid generation of building boundary variations into which the interior functionality may be included afterward.

7. Conclusions

The automatic generation of complex building boundaries, in contemporary research and engineering projects and practices, is heavily processing and data volume bounded. Even with the available processing power and storage capacity of commercial computer systems nowadays the interactive and reactive design is rather limited. The main challenge is the rapid generation of a large number of complex building boundaries in a reasonable time frame.

The other dominant obstacle is that the majority of methods rely on the interior functional constraints as a driving force dominantly resulting in a low level of variability and simple rectangular geometry of generated building boundaries.

As a basis for the automated generation of various building boundaries, the solution presented in this paper introduces a novel approach that ignores the internal (functional) and focuses only on the external (non-functional) impacts. The primary orientation on external impacts may be, at any instance, extended by suitable complementary traditional methodology. The applied research methodology and presented method rely on a developed extendible rule-based system that simplifies building boundaries creation by recursive application of formulated spatial grid generation algorithm. Based on starting parameter values (mainly the lot and building area spaces) the algorithm tends to create a set of grids that satisfy initial constraints by marking the individual grid cells as part of the building or empty.

The presented conceptual framework model served as a foundation for creating a prototype software application that supports the experimental generation of grid arrays that are transformed into readable images of residential building boundaries. For the initial validation of the developed methodology, method, and algorithm, the concrete parametric resolution is set to 1 m.

The comparative analysis has shown that the presented approach overcomes some of the limitations of previous related research that generate building boundaries in simple rectangular form or with limited variability. The proposed method, in its current stage, outperforms discussed existing methods concerning complex building boundary shape generation.

Author Contributions: Conceptualization, M.L.; methodology, A.P. and B.P.; software, M.L., A.P. and B.P.; validation, M.L.; investigation, resources, data curation, and visualization, M.L., A.P. and B.P.; writing—original draft preparation, M.L.; formal analysis, B.P.; writing—review and editing, A.P. and B.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Our Python application is open-source and can be publicly accessed [35].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shekhawat, K. Algorithm for Constructing an Optimally Connected Rectangular Floor Plan. *Front. Archit. Res.* **2014**, *3*, 324–330. [CrossRef]
2. Kozminski, K.; Kinnen, E. An Algorithm for Finding a Rectangular Dual of a Planar Graph for Use in Area Planning for VLSI Integrated Circuits. In Proceedings of the 21st Design Automation Conference Proceedings, Albuquerque, NM, USA, 25–27 June 1984; pp. 655–656.
3. Bhasker, J.; Sahni, S. A Linear Algorithm to Find a Rectangular Dual of a Planar Triangulated Graph. *Algorithmica* **1988**, *3*, 247–278. [CrossRef]
4. Wang, X.-Y.; Yang, Y.; Zhang, K. Customization and Generation of Floor Plans Based on Graph Transformations. *Autom. Constr.* **2018**, *94*, 405–416. [CrossRef]
5. Martin, J. Procedural House Generation: A Method for Dynamically Generating Floor Plans. In Proceedings of the Symposium on Interactive 3D Graphics and Games, Redwood City, CA, USA, 14–17 March 2006; Volume 2.
6. Flemming, U.; Coyne, R.; Glavin, T.; Rychener, M. A generative expert system for the design of building layouts. In *Applications of Artificial Intelligence in Engineering Problems*; Springer: Berlin/Heidelberg, Germany, 1986; pp. 811–821.
7. Del Rio-Cidoncha, G.; Martínez-Palacios, J.; Iglesias, J.E. A Multidisciplinary Model for Floorplan Design. *Int. J. Prod. Res.* **2007**, *45*, 3457–3476. [CrossRef]
8. Nauata, N.; Chang, K.-H.; Cheng, C.-Y.; Mori, G.; Furukawa, Y. House-Gan: Relational Generative Adversarial Networks for Graph-Constrained House Layout Generation. In Proceedings of the European Conference on Computer Vision, Glasgow, UK, 23–28 August 2020; pp. 162–177.
9. Peng, C.-H.; Yang, Y.-L.; Wonka, P. Computing Layouts with Deformable Templates. *ACM Trans. Graph.* **2014**, *33*, 99:1–99:11. [CrossRef]
10. Wu, W.; Fan, L.; Liu, L.; Wonka, P. MIQP-based Layout Design for Building Interiors. *Comput. Graph. Forum* **2018**, *37*, 511–521. [CrossRef]
11. Li, S.-P.; Frazer, J.H.; Tang, M.-X. A Constraint Based Generative System for Floor Layouts. 2000. Available online: http://papers.cumincad.org/data/works/att/5b5d.content.pdf (accessed on 6 November 2021).
12. Harada, M.; Witkin, A.; Baraff, D. Interactive Physically-Based Manipulation of Discrete/Continuous Models. In Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, Los Angeles, CA, USA, 6–11 August 1995; pp. 199–208.
13. Wang, X.-Y.; Zhang, K. Generating Layout Designs from High-Level Specifications. *Autom. Constr.* **2020**, *119*, 103288. [CrossRef]
14. Duarte, J.P. A Discursive Grammar for Customizing Mass Housing: The Case of Siza's Houses at Malagueira. *Autom. Constr.* **2005**, *14*, 265–275. [CrossRef]
15. Nilkaew, P. Assistant Tool for Architectural Layout Design by Genetic Algorithm. 2006. Available online: http://papers.cumincad.org/data/works/att/caadria2006_641.content.pdf (accessed on 4 November 2021).
16. Narahara, T.; Terzidis, K. Multiple-Constraint Genetic Algorithm in Housing Design. 2006. Available online: http://papers.cumincad.org/data/works/att/acadia06_418.content.pdf (accessed on 4 November 2021).
17. Bahrehmand, A.; Batard, T.; Marques, R.; Evans, A.; Blat, J. Optimizing Layout Using Spatial Quality Metrics and User Preferences. *Graph. Models* **2017**, *93*, 25–38. [CrossRef]
18. Wu, W.; Fu, X.-M.; Tang, R.; Wang, Y.; Qi, Y.-H.; Liu, L. Data-Driven Interior Plan Generation for Residential Buildings. *ACM Trans. Graph. TOG* **2019**, *38*, 1–12. [CrossRef]
19. Merrell, P.; Schkufza, E.; Koltun, V. Computer-Generated Residential Building Layouts. *ACM Trans. Graph. TOG* **2010**, *29*, 1–12. [CrossRef]
20. Hua, H. Irregular Architectural Layout Synthesis with Graphical Inputs. *Autom. Constr.* **2016**, *72*, 388–396. [CrossRef]
21. AlOmani, A.; El-Rayes, K. Automated Generation of Optimal Thematic Architectural Layouts Using Image Processing. *Autom. Constr.* **2020**, *117*, 103255. [CrossRef]
22. Bao, F.; Yan, D.-M.; Mitra, N.J.; Wonka, P. Generating and Exploring Good Building Layouts. *ACM Trans. Graph. TOG* **2013**, *32*, 122. [CrossRef]
23. Cruz, C.; Karakiewicz, J.; Kirley, M. Towards the Implementation of a Composite Cellular Automata Model for the Exploration of Design Space. 2016. Available online: https://www.researchgate.net/publication/299597241_TOWARDS_THE_IMPLEMENTATION_OF_A_COMPOSITE_CELLULAR_AUTOMATA_MODEL_FOR_THE_EXPLORATION_OF_DESIGN_SPACE (accessed on 4 November 2021).
24. Krawczyk, R.J. Cellular Automata: Dying to Live Again, Architecture, Art, Design. In *Designing Beauty: The Art of Cellular Automata*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 39–52.
25. Petruševski, L.; Devetaković, M.; Mitrović, B. Self-Replicating Systems in Spatial Form Generation: The Concept of Cellular Automata. *Spatium* **2009**, 8–14. [CrossRef]

26. Herr, C.M.; Kvan, T. Adapting Cellular Automata to Support the Architectural Design Process. *Autom. Constr.* **2007**, *16*, 61–69. [CrossRef]
27. Anzalone, P.; Clarke, C. Architectural Applications of Complex Adaptive Systems. 2003. Available online: http://papers.cumincad.org/data/works/att/acadia03_042.content.09646.pdf (accessed on 4 November 2021).
28. Gardner, M. Mathematical Games. *Sci. Am.* **1970**, *222*, 132–140. [CrossRef]
29. Araghi, S.K.; Stouffs, R. Exploring Cellular Automata for High Density Residential Building Form Generation. *Autom. Constr.* **2015**, *49*, 152–162. [CrossRef]
30. Krawczyk, R.J. Architectural Interpretation of Cellular Automata. 2002. Available online: https://mypages.iit.edu/~{}krawczyk/rjkga02.pdf (accessed on 4 November 2021).
31. Coates, P.; Healy, N.; Lamb, C.; Voon, W.L. The Use of Cellular Automata to Explore Bottom up Architectonic Rules. 1996. Available online: https://repository.uel.ac.uk/download/d3aedf91a9fe865d5d4863f29c921ad4b4a43b77838fb3ba785cfaa4db97b5b9/507983/Coates%2C%20P%20%281996%29%20Eurographics.pdf (accessed on 4 November 2021).
32. Ford, R.C. Think Like Ants, Not Like Gods: A Study of Cellular Automata and Its Validity Within the Architectural Design Process. Master's Thesis, Unitec Institute of Technology, Auckland, New Zealand, 2013.
33. Herr, C.M.; Ford, R.C. Cellular Automata in Architectural Design: From Generic Systems to Specific Design Tools. *Autom. Constr.* **2016**, *72*, 39–45. [CrossRef]
34. Martin, R.C. *Agile Software Development, Principles, Patterns and Practices*; Pearson Education Limited: London, UK, 2014.
35. FPgenerator. 2021. Available online: https://github.com/ArchitectureMarko/FPgenerator (accessed on 4 November 2021).