BARC0141: Built Environment Dissertation

"Gameplay with encoded architectural tilesets: A computational framework for building massing design using the Wave Function Collapse algorithm"

by

Eleni Chasioti

13/09/2020

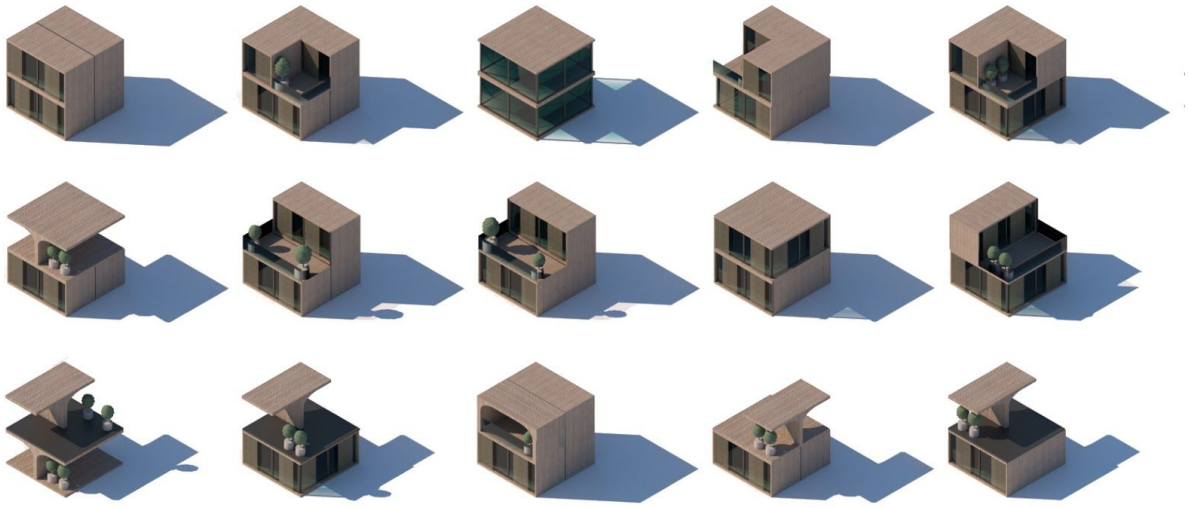
Dissertation submitted in part fulfilment of the Degree of Master MSc Architectural Computation

Bartlett School Architecture
University College London

Gameplay with encoded architectural tilesets:
A computational framework for building massing design using the Wave Function Collapse algorithm

Eleni Chasioti

MSc Architectural Computation



Bartlett School of Architecture, UCL

September 2020

This dissertation is submitted in partial fulfillment of the requirements of the degree of Master of Science in Architectural Computation from University College London.I, Eleni Chasioti, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Automating repetitive and laborious design tasks is an essential objective in both the architectural and the game design industry. Often, it is needed to produce extensive designs or a wide range of variations, a process that affects the time efficiency and the cost of a project. Additionally, most traditional design methodologies exclude non-experts and do not facilitate communication among different parties.

Procedural Content Generation (PCG) is an algorithmic method capable of automating the production of virtual content with limited human intervention. It is an emerging technique becoming particularly popular in the gaming industry where there is often the need for extensive and detailed models. Constraint solving is a novel approach for Procedural Content Generation. The Wave Function Collapse algorithm (WFC) is an example-driven image generation algorithm that uses constraint solving to produce a variety of output images based on an input image.

This dissertation implements the WFC in 3 dimensions, using mesh geometries and voxels. It explores the implementation of the algorithm in the domain of early-stage architectural design and adapts the algorithmic decision making to better facilitate architectural design goals. In addition, it proposes a computational framework where the WFC is combined with the use of architectural tilesets. The workflow aims to provide a friendly user interface where the designer has both control and a basic understanding of the algorithmic process. The proposed workflow is also efficient in computational time, provides results in real-time, and it is flexible in terms of input model type and size.

Acknowledgments

I would like to express my sincere appreciation to my academic supervisor, Vishu Bhooshan, for all his guidance, critical viewpoint, and support and to my technical tutor Vlad Tenu for his useful insight on my project. I would also like to take this opportunity to express my gratitude to the Hellenic Petroleum Company for honoring me with a scholarship to pursue this Master's degree. They actively supported me throughout my studies and made possible their successful completion.

Keywords: WFC algorithm, WFC, PCG, digital tool, computational framework, procedural generation, architectural tilesets, building massing design

Table of Contents

Table of Figures

Pseudocode

1. Introduction

1.1 Thesis Overview

The use of digital tools to assist the design process is increasingly becoming a vital factor in the AEC industry as a response to the emerging complexity and need for adaptability. Generative design, automation, analysis, and optimization are some of the main purposes of the design tools that are domains of continuous research and improvement. Often, architects, digital designers, and developers research other industries for inspiration, creative ideas, or tools with interdisciplinary value.

There are cases where the game industry tends to respond more quickly to algorithmic approaches and technological advances while the AEC industry adapts later to them. This dissertation explores the shared space between the algorithmic processes of the gaming and AEC industry. It seeks to render a texture generation gaming algorithm effective in an architectural design methodology and integrate it into a computational framework.

When the Wave Function Collapse algorithm (WFC) was published in 2015, it was welcomed with great enthusiasm by both digital designers and gamers. Despite the interest, the main algorithm is not widely understood and not yet adapted to architectural design purposes.

1.2 Objectives, Contributions, and Outline

The main objective of this dissertation is to investigate the possibilities of adaptation of the WFC algorithm into an architectural design methodology. The research conducted provides insight into the algorithmic process, comparison between the different known implementations, and proposes two variations of the existing algorithm with a focus on architectural objectives. The current dissertation is organized as follows:
In Chapter 2, a review of previous work upon which this research draws is presented. The chapter briefly presents texture synthesis, which is the basis of the WFC algorithm closing with a preview of three-dimensional implementations and tile-based approaches applied in architectural design. In Chapter 3, the computational framework and the parametric space explorations are explained. Chapter 4 focuses on the documentation of the algorithmic performance by evaluating both the inputs and outputs. The evaluation is based on a set of parameters such as the dimensions of the models, the periodicity of the input, and the use of probabilities. The chapter includes the results of two implemented variations of the algorithm. Finally, Chapter 5 discusses the conclusions of this research, and Chapter 6 discusses possible future work.

The WFC is implemented based on the original algorithm (Gumin, 2015) and the Kazimir project developed by Matvey Khokhlov, focusing on the generation of voxel models (Khokhlov, 2017). The implementation is not using the minimum entropy heuristic, and it is also non-backtracking for time efficiency purposes. This dissertation is also based on Paul C. Merrell's Ph.D. dissertation on Model Synthesis (Merell, 2009). The original algorithm was implemented in the C# programming language for Unity. For this dissertation, the algorithm was reimplemented in the Rhinoceros 3D (Robert McNeel & Associates,1998) environment as a compiled component for the visual programming application Grasshopper (Rutten, 2007).

2. Literature Review

2.1 Gaming Algorithms in Architecture

Architecture and video games have a codependent relationship. As the built environment constructs the majority of what we perceive as the real world, it is an essential part of our everyday experiences. Similarly, the virtual world of a video game is driven by many “real-world” principles, it simulates its limitations and often defines the experience of the gameplay itself.

An essential application of architecture in terms of gameplay is that it forms the background for action and is the scaffolding for all the characters’ movement. In many cases, the environment is designed as Minotaur’s Labyrinth, where the player wanders. In games like *Minecraft*, the role of architecture is enabling the creation of a user-generated story in a world-building environment. The users can construct their own narrative using a simple set of blocks and rules, choosing between different vegetation, material, and building types. Minecraft has expanded beyond its entertainment origins — design studios like London-based *Blockworks* now act as pseudo-virtual architecture firms that create bespoke Minecraft environments for major clients like Disney. There are also cases of games like *Myst*, where the world disobeys the laws of nature, and the player is the one to discover the many architectural allusions. (Artemel,2017)

The relationship between video games and architecture is not a new one. There are numerous ways architecture serves video games, and video games inspire architecture and promote novelty. There are many examples of architects working on the design of virtual worlds such as the 2016 adventure puzzle game *The Witness*. There are also cases where the gameplay is used to help underserved communities in developing countries to design their own public spaces like *The Block by Block* initiative, run by the United Nation’s housing arm, UN-Habitat, in partnership with the makers of *Minecraft* (Yar, 2018).

2.2 Procedural Content Generation

Procedural Content Generation (PCG) is the automated production of different types of media. This media can be anything humans would usually produce, such as poetry, paintings, music, architectural drawings, or films. PCG for games is the use of algorithms to produce game content that a designer would typically create, such as textures, sound effects, maps, levels, characters, weapons, quests, or even game mechanics and rules. Content generation for video games demands a lot of manual labor, and it is considered one of the main costs in video game development. In many cases, the content needed has to be highly detailed and of great size. Procedural Content Generation (PCG) is a technique used to reduce that cost by generating content algorithmically, demanding little human contribution (Barriga, 2019).

A key point when referring to procedural content generation is *randomness*. There are games that implement a deterministic version of PCG, but the more usual case is the use of a decision-making algorithm based on a random seed. Game designer Dan Kline argues that a key to good game design with procedural systems is to replace uniform randomness with directedness. Directing random content creation can occur through carefully designing the individual pieces of content that are being assembled or by altering the algorithm itself. (Smith, 2015)

The main difference between procedural design and parametric design can be perceived as the following: A procedural designer creates a procedure that produces a desirable result. Once the system is created, it can be repeated or altered with little human effort to get different results. On the other hand, a parametric designer handles measurable characteristics (parameters) to describe an outcome out of them. A parametric object has parameters that can be adjusted, while procedural is a process that combines objects with parameters. (“Procedural vs. Parametric,” 2020)

2.3. Texture Synthesis

Claude Shannon was an American mathematician, electrical engineer, and cryptographer known as *"the father of information theory."* In 1948, he wrote an article titled "A Mathematical Theory of Communication," mentioning an interesting way of producing English-sounding written text using *n-grams*. The idea is to produce text using a generalized Markov chain. A set of consecutive letters or words create an n-gram and define the probability distribution of the next word or letter. When a large sample of language is used, such as a book, probability tables for each n-gram can be built. Following that, it is possible to produce English-sounding text using the built Markov chain to sample from it .(Efros & Leung, 1999).

The above idea was popularized by an early computer program called *Mark V. Shanley*. Shanley was a *Usenet* user whose postings on the *net.singles* newsgroups were generated by Markov chain techniques, based on text from other postings. The username is an anagrammatism of the words *"Markov chain".* A Markov chain represents statistical data about how past information was followed by successor information. Such data can be utilized to make predictions under the assumption that future behavior will be similar to past behavior. It can also be used to generate future data probabilistically. ("Elements of Computer Science", 2013)

Machine Learning professor at California Institute of Technology (Caltech), Yisong YueA, developed a chatbot named Mark V. Shanley, based on the original chatbot. The chatbot can be found online, and it was tested with the following phrase: *"A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. A countably infinite sequence, in which the chain moves state at discrete time steps, gives a discrete-time Markov chain. A continuous-time process is called a continuous-time Markov chain."* (Gagniuc, 2017) Some of the produced results are presented below:

1. *A countably infinite sequence, in which the chain moves state at discrete time steps, gives a discrete-time Markov chain.*

2. *A Markov chain is a stochastic model describing a sequence of possible events in which the chain moves state at discrete time steps, gives a discrete-time Markov chain.*
3. *A continuous-time process is called a continuous-time Markov chain.*
4. *A continuous-time process is called a continuous-time Markov chain.*
5. *A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.*

In this case, an abstract was passed to the algorithm in order to produce English sounding text. A bigger, cohesive abstract can result in a wider variety of output texts with good chances of them making sense logically. The algorithm doesn't have knowledge of the meaning of the words but predicts what word or phrase should follow a previous one. This is, essentially, the idea behind most of the texture synthesis and model synthesis algorithms that were developed after this research was published. The main difference lies in the dimensions of the different implementations. Texture synthesis algorithms work mostly in 2D using pixels to generate or complete images, while model synthesis algorithms work mostly in 3D. There is also the possibility of a texture synthesis algorithm to work with time as the third dimension to produce video outputs.

There is some obscurity on the term texture and, consequently, on what is the objective of texture synthesis. Textures are features used to partition images into regions of interest and to classify those regions. They provide information in the spatial arrangement of colors or intensities in an image and are characterized by the spatial distribution of intensity levels in a neighborhood. A texture consists of texture primitives called *texels* and can be classified as deterministic (regular) or statistical (irregular). Deterministic textures are created by repeating a fixed geometric shape such as a circle or square. "*Examples of deterministic textures are patterned wallpaper and bricks. Statistical textures are created by changing patterns with fixed statistical properties. Most natural textures, like wood or stone, are statistical* " (Wirth, 2004).Texture synthesis is a relatively new and very active research area in computer graphics. The goal of texture synthesis can be stated as follows: *Given a texture sample, synthesize a*

new texture that, when perceived by a human observer, appears to be generated by the same underlying stochastic process (Wei & Levoy, 2000).

The origins of the texture synthesis research can be found in the '80s, but it is believed that the seminal papers by (Efros & Leung,1999) and (Wei & Levoy,2000) inspired extensive further research. Both methods are elegant, easy to implement, and work better than any other previously proposed method.

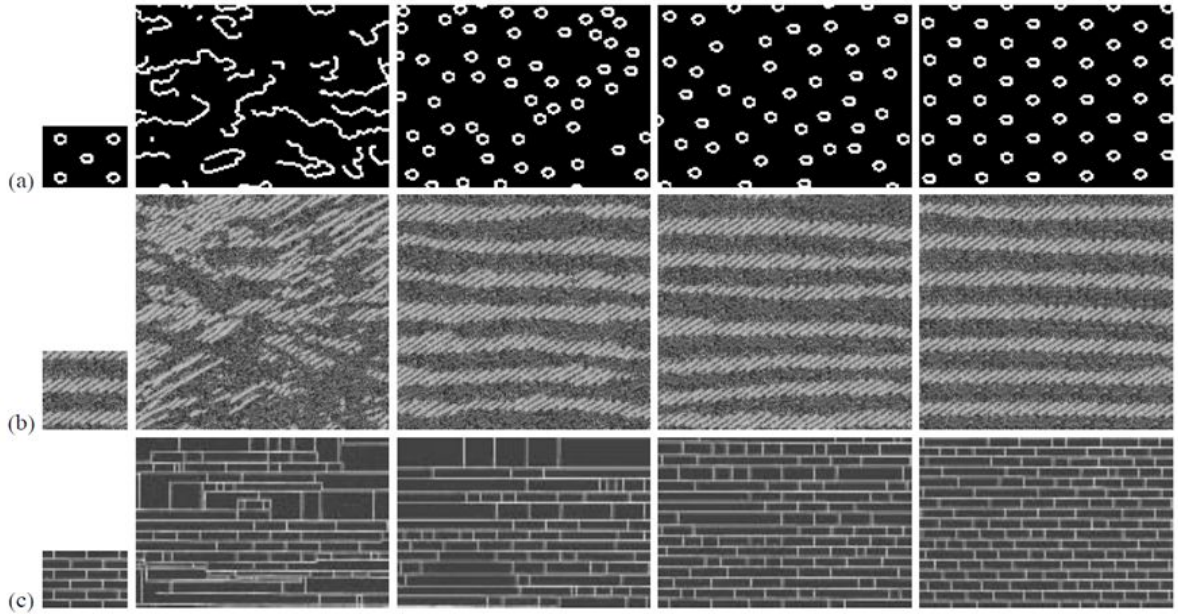


Figure 1: Input textures a),b) and c) and their outputs with the Efros and Leung method.

The Efros and Leung method demonstrated in Figure 1 grows the output image, pixel by pixel, starting from an initial seed and moving outwards. Pixels are chosen as the initial units the algorithm works with. The final texture is modeled as a Markov Random Field (MRF). "*The algorithm works assuming that the probability distribution of brightness values for a pixel given the brightness values of its spatial neighborhood is independent of the rest of the image. The neighborhood of a pixel is modeled as a square window around that pixel. Finally, the size of the window is a free parameter that specifies how stochastic the user believes this texture to be*." (Efros & Leung, 1999).

2.4. Introduction to the WFC algorithm

The WFC algorithm is a procedural content generation algorithm developed by Max Gumin (Gumin,2015). The name of the algorithm refers to the quantum mechanic process of changing the superposition of the wave function caused by the presence of the observer. Originally, an unobserved state has high entropy that is decreasing proportionally when particles are observed. Once the state is observed, the entropy becomes equal to zero, and the wave function collapses.

The algorithm was originally developed to work in 2D for pixel-based image generation, but it can easily be adapted to work in 3D producing voxelized models or models based on a 3d tileset. Marian Kleineberg adapted the WFC to create an infinite city assembled from 3D blocks like bridges, buildings, and streets (Kleineberg, 2018). The content is continuously generated towards the direction the player is moving to create the impression of an infinite virtual city. One of the first games generated with the use of *WFC* is *Proc Skater 2016* *(Parker, Jones and Morante, 2016)*. It is a skateboarding game in which a player can enjoy numerous procedurally generated skate parks and even save their preferred configuration.

The work of Oskar Stålberg is probably the most well-known example coming from the gaming community. The real-time strategy video game *Bad North* (Stålberg and Meredith,2018) implements the WFC in three-dimensional space to generate islands that should be a navigable game space. Currently, users can also try the early release of *Townscaper* (Stålberg, 2020), a procedural generated city-building toy with an oceanic, medieval aesthetic, and minimalist design.

Maxim Gumin's WFC algorithm is an example-driven image generation algorithm. The goal of the algorithm is to generate new images in the style of a given example image based on preserving local similarities. The algorithm ensures that every local window in the input image will exist somewhere in the output image. Operationally, WFC implements a non-backtracking, greedy search method.

At the top level, it performs four key tasks that are presented in Figure 2:

1. It extracts all the local patterns of a defined size from the input image.
2. It converts the inferred patterns into an index to make constraint checking faster.
3. It starts from a random location and incrementally generates the output image.
4. It renders the final image.

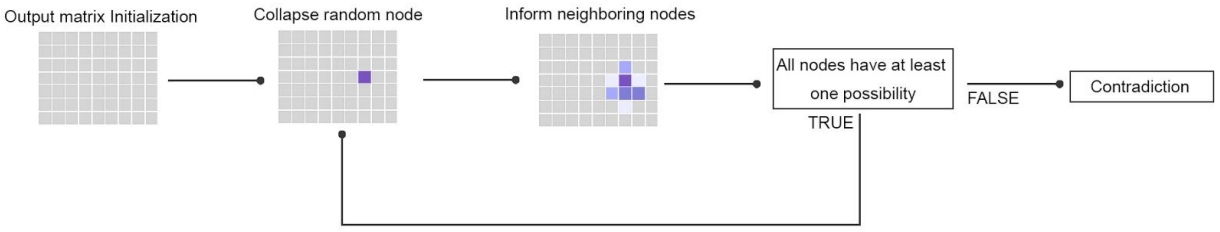


Figure 2: The main steps of the WFC

There are two versions of the algorithm included in the original implementation: *"a tiled model"* and *"an overlapping model."*The main difference between the two is the method followed to infer the patterns from the input image. In the tiled model, the image is cut into sub-images that are considered as patterns, while in the overlapping model, a *sliding window* is created inferring overlapping patterns and thus a greater variety of them. A brief explanation is included in Figure 3, and a more detailed comparison is presented in section 3.3.1.

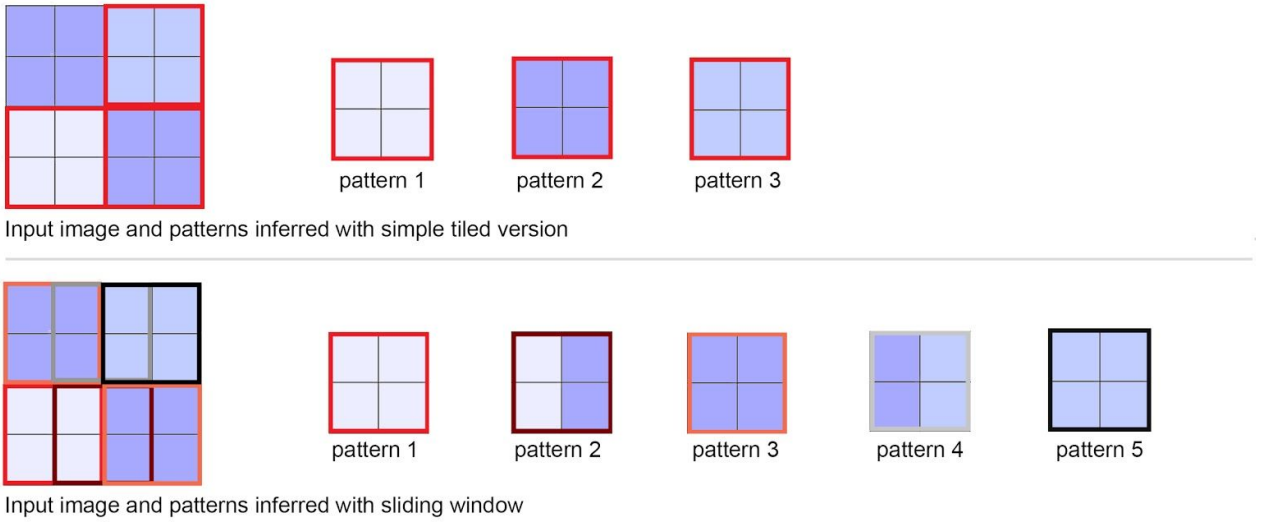


Figure 3: Tiled and overlapping inference of patterns in an input image

2.5 Constraint Solving Algorithms

According to Tsang (1996), a constraint problem is defined as a problem composed of a finite set of variables. Each variable is associated with a finite domain and a set of constraints that form the restrictions on the allowed values for each variable at a certain point in time. The purpose of a constraint solving algorithm is to assign a value to every variable and at the same time, satisfy all the constraints.

A constraint on a set of variables is a limitation on the values they can have at the same time. Conceptually, Constraints can be represented as sets of all the legal labels for a set of variables, but they can also get the form of equations, inequalities, matrices, etc. Figure 4 by Tsang (1996) presents a formal definition of constraint satisfaction problems.

A **constraint satisfaction problem** is a triple:

$$(Z, D, C)$$

where Z = a finite set of **variables** $\{x_1, x_2, ..., x_n\}$;

D = a function which maps every variable in Z to a set of objects of arbitrary type:

D: $Z \rightarrow$ finite set of objects (of any type)

We shall take D_{x_i} as the set of objects mapped from x_i by D. We call these objects possible **values** of x_i and the set D_{x_i} the **domain** of x_i;

C = a finite (possibly empty) set of **constraints** on an arbitrary subset of variables in Z. In other words, C is a set of sets of compound labels.

We use *csp*(P) to denote that P is a constraint satisfaction problem. ■

Figure 4: The formal definition of constraint satisfaction problems

There are two main categories of algorithms that deal with constraint satisfaction problems (CSP): inference and search algorithms. Inference methodologies aim to simplify a problem so as to provide a solution that preserves its semantics. In a simplification method case, the set of variables and constraints might be transformed, and invalid parts might be discarded from the search space. Exploration of the search space is required in the general case where it is impossible to solve a given CSP instance purely by inference. This exploration is called search. (Lecoutre, 2009).

2.6 Digital tools for building project development

The proposed computational workflow of this dissertation is organized based on the logic of a digital tool for the early stages of design. The idea of a digital tool that can support the architects, designers, and developers in the first steps of a project is becoming increasingly favored. Informed decisions based on analysis and simulation can contribute to time-saving and reducing costs at the beginning of a project. A generative tool can also contribute to the exploration of the design space, the production of design variations, and thus to faster and efficient communication and decision making in a design team. In this section, four examples of online digital platforms that aim to assist the design process are briefly presented.

Archistar is a property technology and generative design company aiming to solve property and design problems. The company attempts to respond to the needs of well-designed living and working spaces worldwide. Following the growing demand for buildings, a significant challenge of managing costs and efficiencies emerges. When design modifications need to be applied in the middle of a property development process, that has an immediate negative effect, increasing the cost of the project. Through technology and Artificial Intelligence, Archistar assists property professionals to reduce the time and risk that is inherent in the property development process.

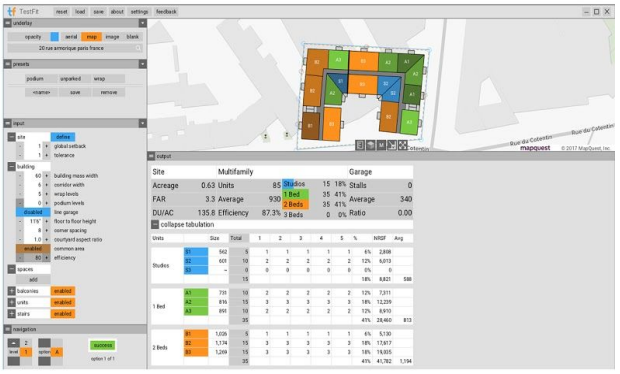
Testfit.io is another example of a company providing a building configurator specialized in urban design. This platform provides geolocation, different building types, unit styles, and building schemes. There is a manual and generative design mode available as well as sun path data, parking lot design, and the ability to export the generated building massing to widely used architectural design software.

Hypar is a cloud platform for generating buildings through an interface that offers computation, visualization, delivery, and interoperability. It aims to support building design through generative methods called functions. For example, geolocation of the design building is an available function that can be chosen and results in other functions related to it. Another function offers the option to create a building massing from an outline curve; in this case, the function is connected to certain parameters called

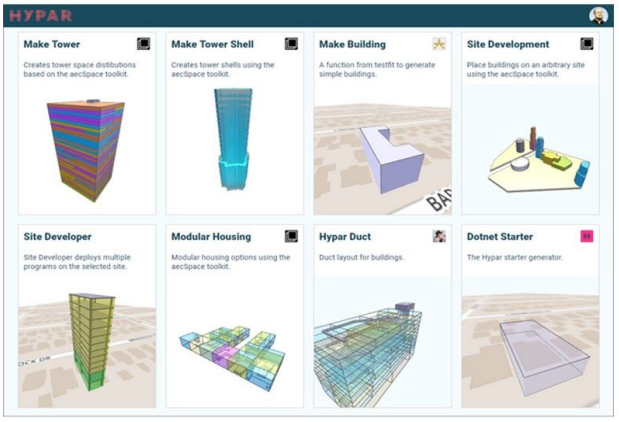
parameter settings. Parameters settings can be the desired building height, the setback interval, the foundation depth, etc.

Prism by Brydenwood is an open-source application that focuses on precision-manufactured housing (PMH) for London. It is an online platform that combines the Mayor of London's spatial planning rules with precision manufacturing expertise to help determine viable PMH development options. Some of the features provided are building design on a geolocated map, exploration of different layout options, project metrics, analysis as well as final design export. Figure 5 presents the interface of the digital tools described in this section.

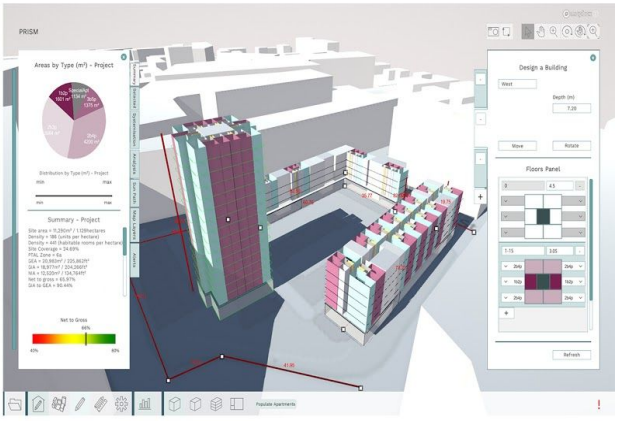
a) archistar interface



b) testfit interface



c) hypar interface



c) prism interface

Figure 5: The interface of the four digital tools presented in this section: archistar, testfit, hypar and prism

3. Computational Framework

The proposed computational framework consists of four main steps as presented in Figure 6:

1. The segmentation of the model to infer a tileset.
2. The encoding of the tileset into digits.
3. The WFC that reads the encoded input, and produces an encoded output.
4. The decoding step that deserializes the output and converts it back to mesh tiles.

Figure 6: The main steps of the computational framework

In case the input model is already divided into tiles, the first step of the Segmentation is not implemented, and the tiled input is directly used in the encoding process. The segmentation process includes the creation of a voxel grid around the input model. The input model gets segmented to individual mesh models, which are equal to the number of voxels in the 3d grid. After the segmentation, the resulting mesh geometries and their equivalent voxels can be used in the next step. The encoding process identifies the unique mesh tiles in the input.

After all the voxels and their enclosed geometries are serialized, the serialization numbers are used as the input for the WFC. Internally, the algorithm constructs a 3d input matrix where the voxels are used as place holders indicating where a node exists or not in a specific x,y.z position. The value of each voxel is its identity, a number associated with a position, and related to all the other numbers and their positions. The WFC infers all the patterns of a specific pattern size and creates an output 3d matrix using those patterns, placing them only in the adjacencies that are allowed by the input matrix.

The stochastic nature of the algorithm can lead to both the successful production of an output model or to a contradiction state. The user-defined parameters that will be further discussed in section 3.3.1 can also affect the outcome of the algorithm. After an output model is provided, the decoding process converts the values of the output matrix back to their equivalent mesh tiles and places them in the correct x,y,z coordinates using the output voxels also taken as an output from the WFC. With the final placement of the tiles in their positions, the representation of the output model is completed. With just one input model, several output models of different sizes can be produced.

3.1. Model Segmentation to create a tileset

The segmentation step is the starting point of the proposed workflow. The input mesh surface could be a rough idea for a facade or the outer shell of a building in the early stages of design. The process provides the option to be thickened to ensure the creation of valid mesh tiles. The user can segment a surface with no thickness or test different offset values if it is necessary. As seen in Pseudocode 1, the inputs to the algorithm are a list of boxes, a mesh geometry, and an offset value.

Algorithm : Segmentation

Data: (boxes, Mesh, offset)

```
for (Every box in boxes) do
    Weld the box
end
OffsetMesh = Mesh Geometry thickened using the offset value/;
for (every box in boxes) do
    Perform Boolean Intersection between the box and the OffsetMesh;
    Store the resulting Geometry to the OutputMeshes;
end
```

Result: (OutputMeshes)

Pseudocode 1: Segmentation Process

The total number of boxes should contain the geometry to be segmented in order to place one segment per box. If a box does not contain a part of the input geometry, the final output Meshes List will have a null value placed in the specific index. In Figure 7, the segmentation process of a flat surface is presented as well as the inferred tileset.

The recognition of unique tiles will be further explained in section 3.2 but is included here as a proof of validity for the produced tileset.

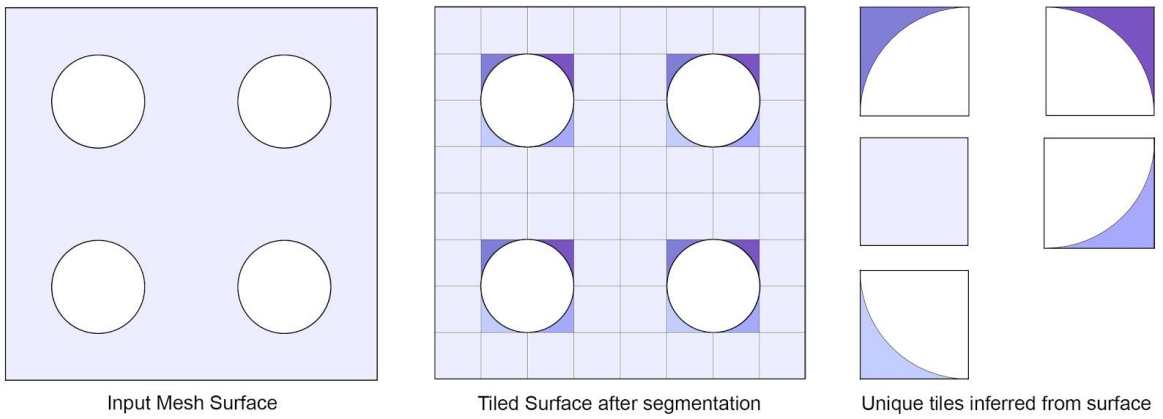


Figure 7: The segmentation of flat surface without offset and with an 8x8 voxels grid

The segmentation process is not limited to flat surfaces. It can also work with 3D ones with less repetition or uniformity. The steps executed are the same, but in this case, different offset values should be tested to ensure all the tiles created are valid meshes. In Figure 8, a curved mesh surface with no thickness is offsetted, segmented, and a tileset of 44 unique tiles is inferred. The input surface is created by lofting two similar but not identical curves resulting in a surface with a gradually changing topology from left to right. This is indicative of the level of resolution the segmentation process can capture.

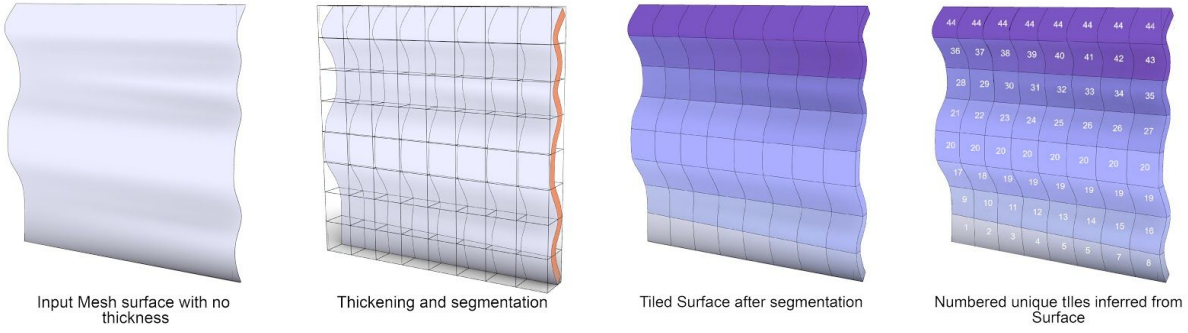


Figure 8: The segmentation of curved surface with 0.05 offset and an 8x8 voxels grid

Indeed, the numbering of unique tiles demonstrates that every row of tiles in the segmented model presents a different variety of tiles. For example, the fourth row is created with only one tile because the two lofted curves are identical in this position. On the other hand, the fifth row is created from 8 unique tiles because of the big difference between the two lofted curves in that position. In the following sections, the design

workflow used will not include the segmentation stage since the present dissertation mainly focuses on the design of tilesets and their use in a design methodology. However, the segmentation process could be further explored, improved, and fully integrated into the proposed workflow.

3.2. Tileset input encoding process

Previous implementations of the WFC that include a tileset input follow a naming process of the different tiles, which is automated or manual. Naming a tileset can be laborious and limiting. The proposed workflow includes an encoding process where the different tiles are automatically recognized based on their shape.The encoding step of the workflow is essential for the performance of the proposed workflow. The WFC is an agnostic algorithm that is not aware of the input type. In that sense, it can work with both images or 3D models without any significant alterations in the core algorithm. The encoding process converts the tiled input into a list of integer values, and that is all the necessary information to produce an output. When two or more voxels enclose the same geometry, the same value is assigned to them. As presented in Pseudocode 2, the process starts by dividing the voxels into subdivision points.

```
Algorithm : Encoding
   Data: Boxes, Meshes,div
1  for (Every box in Boxes and every mesh in Meshes) do
2  Divide every face into points with a div*div resolution;
3  divisionInterval = face domain / div;
4  for (Every division point) do
5  |  Store the distance from its closest point on mesh
6  end
7  Create a list of integers named Bools;
8  if (distance <= (divisionInterval*0.5)) then
9  |  add 1 to Bools List;
10 else
11 |  add 1 to Bools List;
12 end
13 for (every box in boxes) do
14 |  Parse the Bools list to create a string;
15 |  Store the string to a List of Codes;
16 |  Find the unique items in Codes;
17 |  Add the unique items to the Unique Items List;
18 end
19 for (every item in Unique items) do
20 |  Find the index of the item in the Codes List;
21 |  Add the index to the Encoded List;
22 end
   Result: (Codes, Unique Items, Encoded List)
```

Pseudocode 2: Encoding Process

For every experiment conducted in this dissertation, the resolution of the subdivision was 10, meaning that every voxel face was subdivided into a ten by ten grid. Other resolutions might perform better for different types of tiles. After the subdivision points are created, the mesh geometry enclosed in every voxel must be checked. The encoding process focuses on the way the mesh geometry intersects the sides of the voxel. The intersection is examined for a certain threshold *(division_interval*05),* and then the intersection points are stored. The algorithm converts the 100 points per face into 0 and 1 digits based on whether they intersect the face or not.

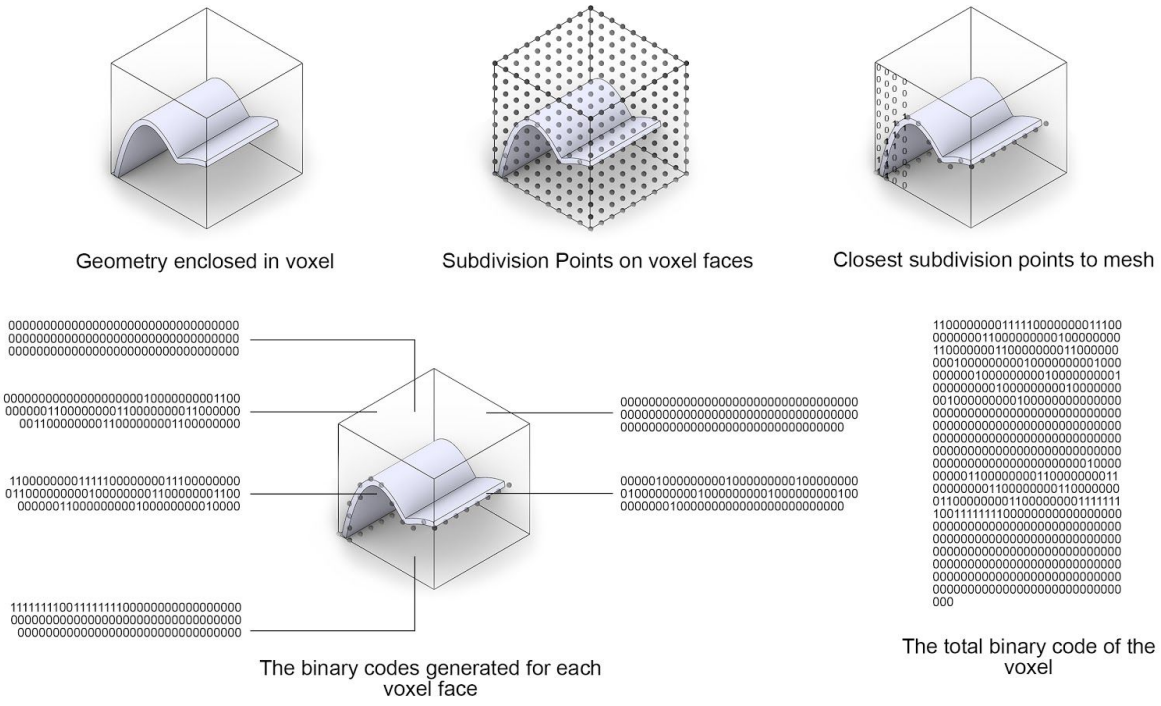


Figure 9: The basic steps of a mesh geometry serialization

This *identity* is a binary code that, as seen in Figure 9, is constructed with the combination of the codes generated for every face of the voxel. For a ten by ten resolution, each face has a 100 digit code, and the total identity is a string of 600 digits. The encoding process is based on the idea of Boolean descriptions of buildings, formulas that encode building configurations into hexadecimal numbers. An example of such a formula is the boolean description of the Seagram Building discovered by Lionel March (Steadman, 2016).

3.3. The WFC algorithm

The basis of this dissertation is a non-backtracking implementation of the WFC algorithm with random initialization. The algorithm is adapted to work with the information provided by the encoding step, which is the serialized numbers of the input meshes. The output of the algorithm is also serialized and converted back to mesh geometries by the decoding step that will be presented in the following section. The core algorithm is described in Pseudocode 3.

```
Algorithm : Wave Function Collapse
  Data: (Boxes,Identities,PatternSize,InputSize,OutputSize,Probabilistic,Periodic, Generate)
1 Convert boxes to voxels;
2 Store the voxel identities;
3 Initialize the input matrix;
4 if (Generate) then
5     while (Generations < 500 and output = unfinished) do
6         Observe
7         Store the input allowed adjacencies ;
8         Initialize the output matrix;
9         if (Probabilistic) then
10            Pick random node in output matrix;
11            Collapse random node into a definite state based on probabilities of occurrence in the input;
12        else
13            Collapse random node into random definite state;
14        end
15        Inform surrounding nodes and limit their possible states;
16        if (Contradiction) then
17            throw error message and restart;
18        else if (output = finished) then
19            Convert voxels to boxes;
20            Return the OutputBoxes, the OutputIdentities,the FoundPatterns, the Probabilities;
21        end
22        Generations++;
23    end
24 end
  Result: (outputBoxes, outputValues, Patterns, Probabilities)
```

Pseudocode 3: WFC Algorithm

The algorithm reads a list of serialized values that are assigned to a 2d or 3d input matrix of nodes. Each node is a location that has a voxel, a set of coordinates, and an assigned value. The patterns in which the values appear in the input matrix are stored and indexed. Those patterns are used to create the output model which is also a 2d or 3d matrix.

The initialization of the process is random. When the output matrix is constructed, all the nodes can possibly be in the state of all the inferred patterns. After choosing a node randomly, the node is collapsed into a random state. *Collapsing* is the assignment of a pattern to a node and the discard of all the other possibilities for that specific node. The next step is the *propagation* when all the surrounding nodes are informed, and their possibilities are limited based on the collapsed node. Following the described process, the WFC creates an output matrix incrementally or reaches to the point of *contradiction* and restarts. A limit of 500 generations was set as the maximum number of attempts that the algorithm can make to complete an output. Some implementations include a *backtracking technique* to handle contradictions, but others opt for restarting the whole process as it is less time-consuming. The present implementation also excludes the backtracking technique.

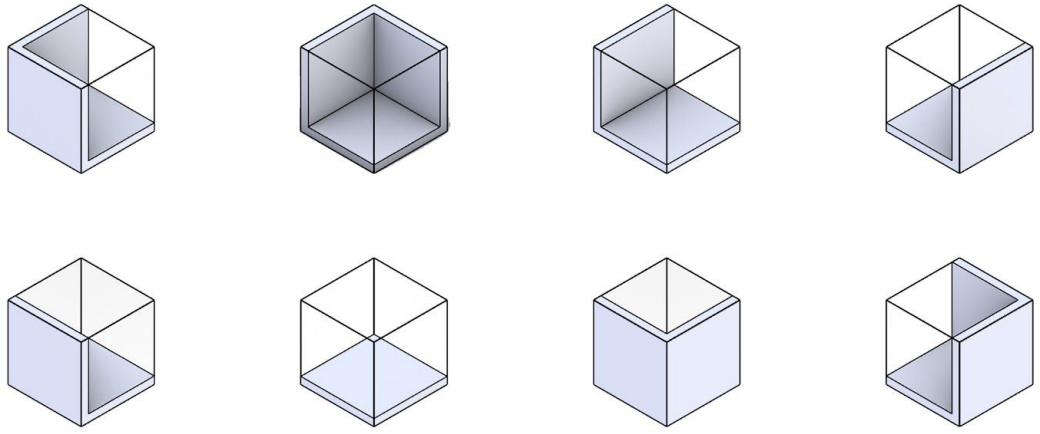


Figure 10: TIleset for space layout design

In Figure 11, the algorithm is tested with the simple tileset illustrated in the Figure 10 for a space layout design. An input of 12 tiles in the x-direction and 12 tiles in the y-direction is designed, and outputs of multiple sizes are taken. The outputs are created with periodic adjacencies allowed resulting in the creation of “open” layouts, layouts without a closed, surrounding wall. Additional constraints like forcing the collapse of specific nodes into the desired state can be a remedy to this issue.

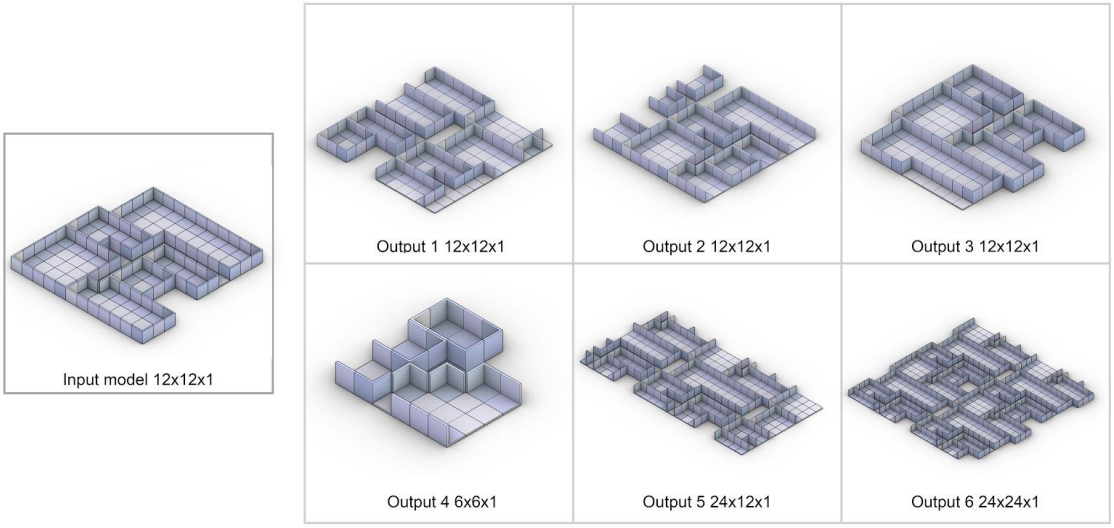


Figure 11: Simple tiled WFC example of input and outputs of different sizes

In Figure 12, the progress of the algorithm is captured. In the initial state, every position in the output model has the potential to be any tile existing in the input. All the positions are represented with the first tile “seen” in the input. The following step is the collapse of a random position into a random state.That has an effect on the possibilities of the neighboring positions. As the number of iterations increases, the output model is gradually constructed until the point after the fourth iteration when the contradiction is reached. The contradiction exception means that no other tile can be collapsed without violating the legal adjacencies or that a position has no possibilities left.

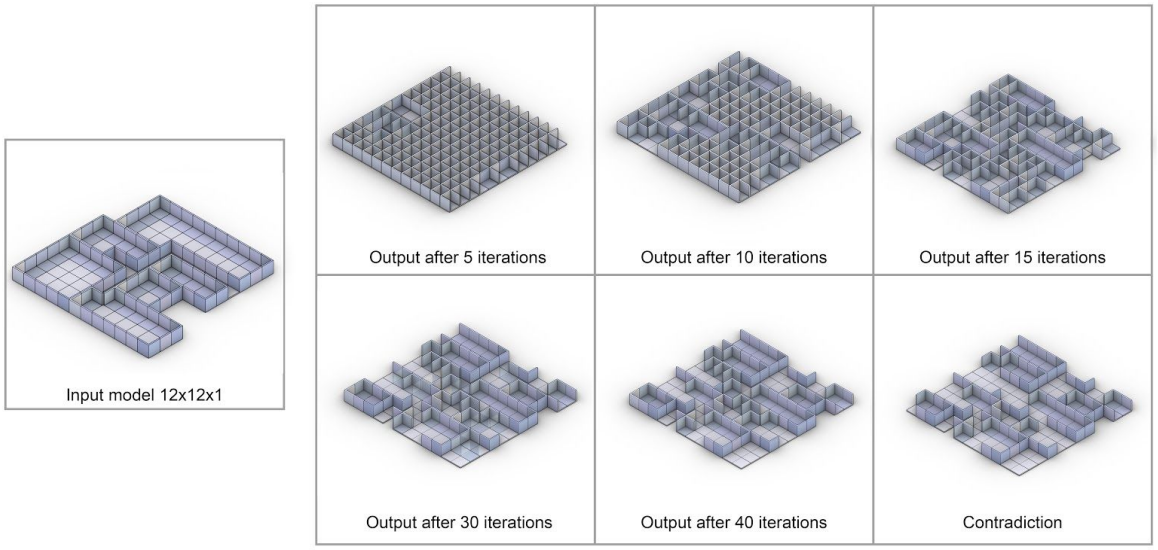


Figure 12: Simple tiled WFC output generation steps and contradiction point

3.3.1.The simple tiled and the overlapping model

There are two implementations of the WFC where the main difference is the way of inferring the patterns from the input image or model. The simple tiled version cuts the input image into NxN patterns, converts them into an index, and replicates them in the output model based on their legal adjacencies. For example, If the pattern with index 1 never appears on the left of the pattern with index 3 in the input image, that should also be true in the output image.

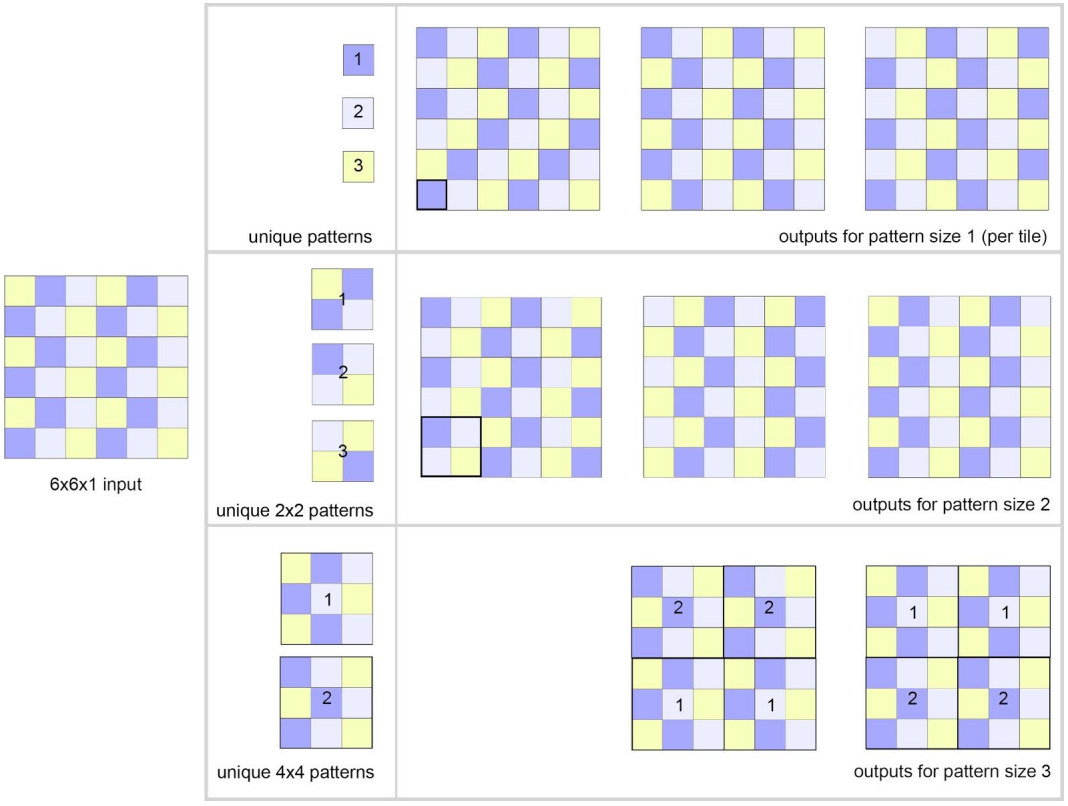


Figure 13: Implementing the simple tiled model for a 6x6x1 input

As illustrated in Figure 13, an input image of 6x6x1 is passed to the simple tiled version of the algorithm. When the pattern size is 1, the adjacency relationships are inferred for every single tile, in this case, for blue, light blue, and yellow tiles. When the pattern size is 2, the input image is cut in 2x2 pieces, and for pattern size 3 it is cut in 3x3 pieces. The algorithm is not able to work for a pattern size bigger than 3 and in that case, it returns an error. That happens because the simple tiled version works only for a pattern

size that can divide the length of the input model in all directions, resulting in an even number. In any other case, the inference of complete pattern sets is not possible.

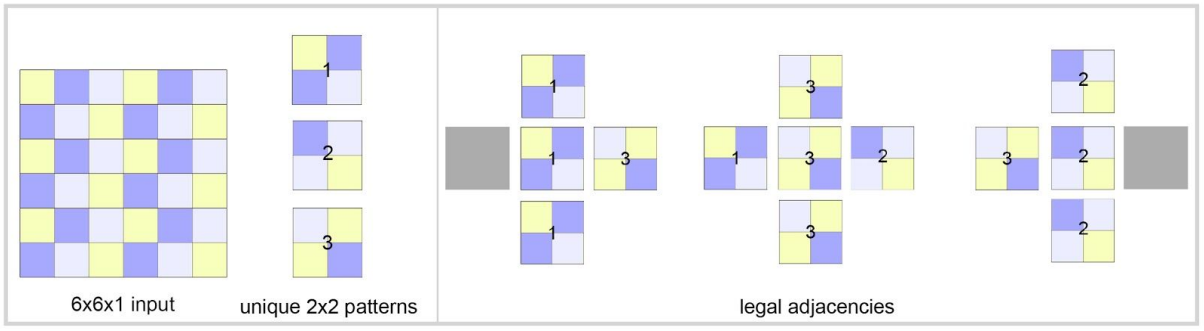


Figure 14: Legal pattern adjacencies with the simple tiled model for 2D input

As seen in Figure 14, when the input is two dimensional, the legal adjacencies are inferred in four directions (left, right, up, down).In case the input is 3 dimensional, the vertical up and down directions are also included.When the input image is described as periodic, the algorithm assumes that it can be wrapped around.That assumption allows tiles in the edges of the input to be placed next to each other on the output, as seen in the following figure. As seen in Figure 15, in the input image, there are no dark blue tiles adjacent vertically or horizontally. When asking for an output image of double the length, the WFC wraps around the original image and replicates it to cover the extra tiles and produce a valid output.

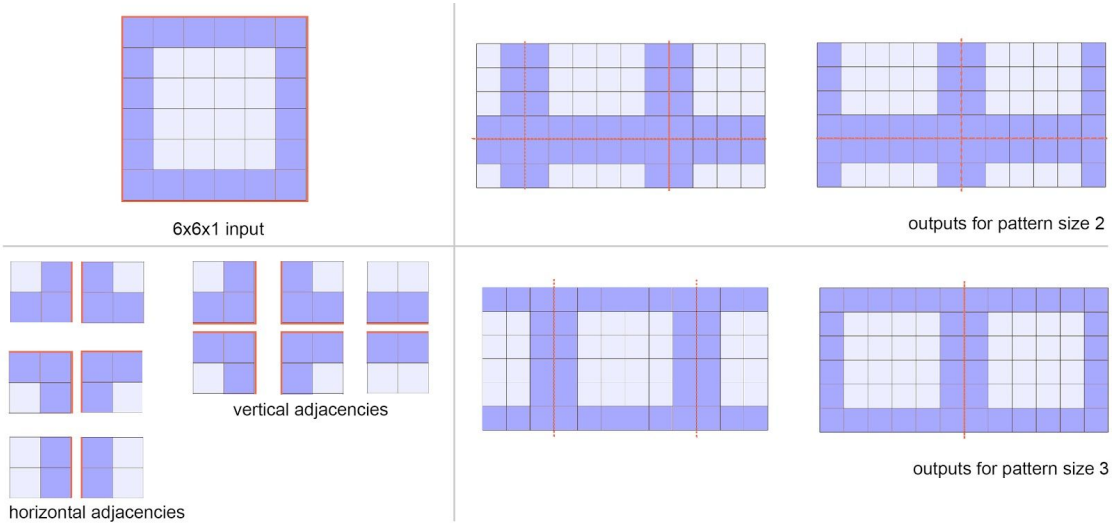


Figure 15: Implementing the simple tiled model with periodic input resulting in additional horizontal and vertical adjacencies of patterns

Another user-defined parameter is whether the probability of occurrence of each pattern in the input image should be taken into account for the creation of the output image. In case the probabilities are used, if a pattern appears more often in the input model, it is more likely to be chosen as the definite state of a node in the output model. In the following figure, the algorithm inferred three patterns with different probabilities of occurrence. When probabilistic, the algorithm preserves these probabilities resulting in the dark blue tile appearing more often as seen in Figure 16:

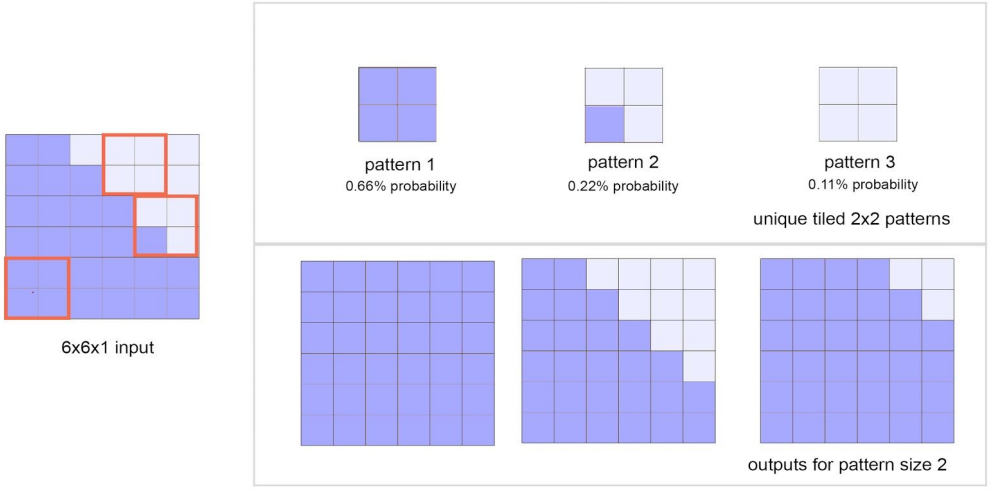


Figure 16: Implementing the simple tiled model with probabilities

In Figure 17, the second implementation of the algorithm, called the overlapping model, is presented. The overlapping model extracts patterns using a sliding window, which is a subgrid with a size equal with the size of the pattern. It can infer a bigger range of patterns, producing a greater variety of images or models. One drawback of this version is that it is significantly slower than the simple tiled model.

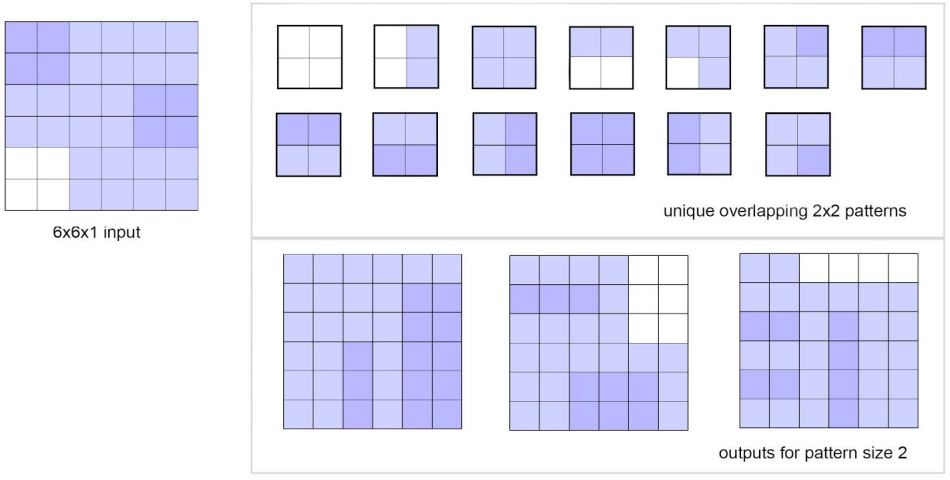


Figure 17: Implementing the overlapping model resulting in the inference of 13 unique patterns

3.3.2. Goal-oriented decision making on the simple tiled model

The developed variation of the algorithm is based on the idea that every enclosed tiled can carry more information than its own identity. Based on that, the algorithm accepts one extra input, a *list of values* of equal number as the voxels. The values do not define the identity of the geometries but rather a quality of them. That quality could be the geometry's area, volume, curvature, or any other measurable quality that the designer wishes to take into consideration. Logically, tiles with the same identity should also have the same value associated with them. However, the assignment of different values is also allowed. The variation of the algorithm is represented in Figure 18.

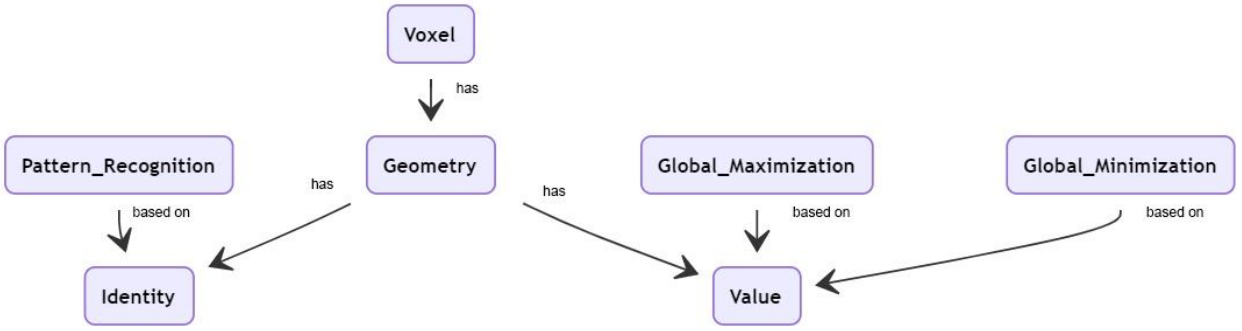


Figure 18: Goal-oriented WFC

In the proposed variation of the WFC, every voxel has a geometry with an *identity* and a *value*. The objective is the maximization or minimization of the total addition of the values carried by the voxels. The probabilistic version of the algorithm uses the probability distribution of the input patterns to collapse a node into a definite state. When it is not probabilistic, the choice is random. This randomness is what the current variation of the WFC attempts to limit. As described in Figure 19, the algorithm, when needed to make a choice of states, chooses based on the user objective function and not randomly, which is the case in a simple model. An example of the process is also provided in Figure 20.

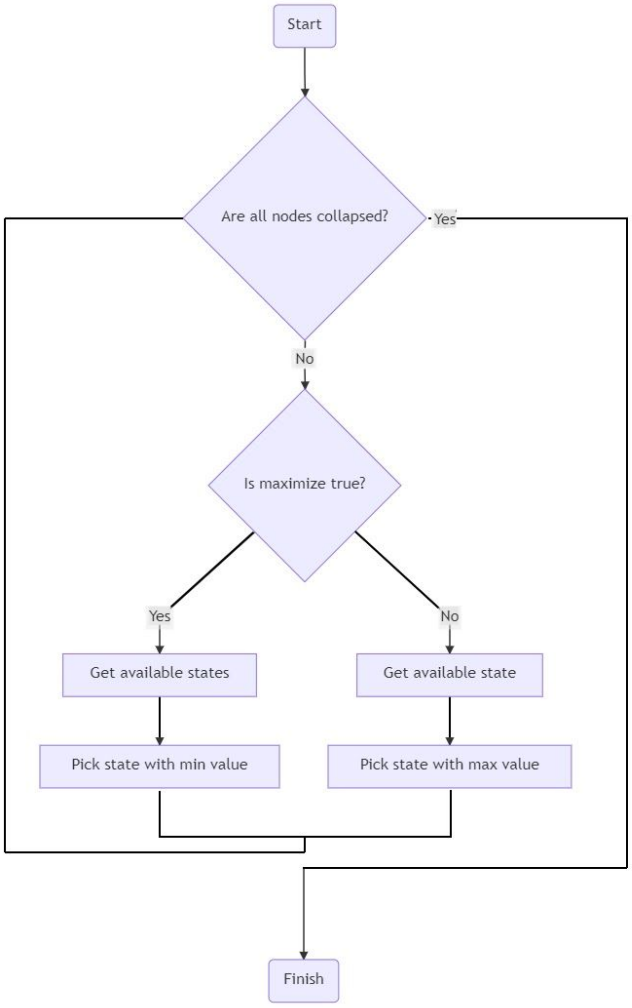


Figure 19: Flowchart of the algorithm decision making

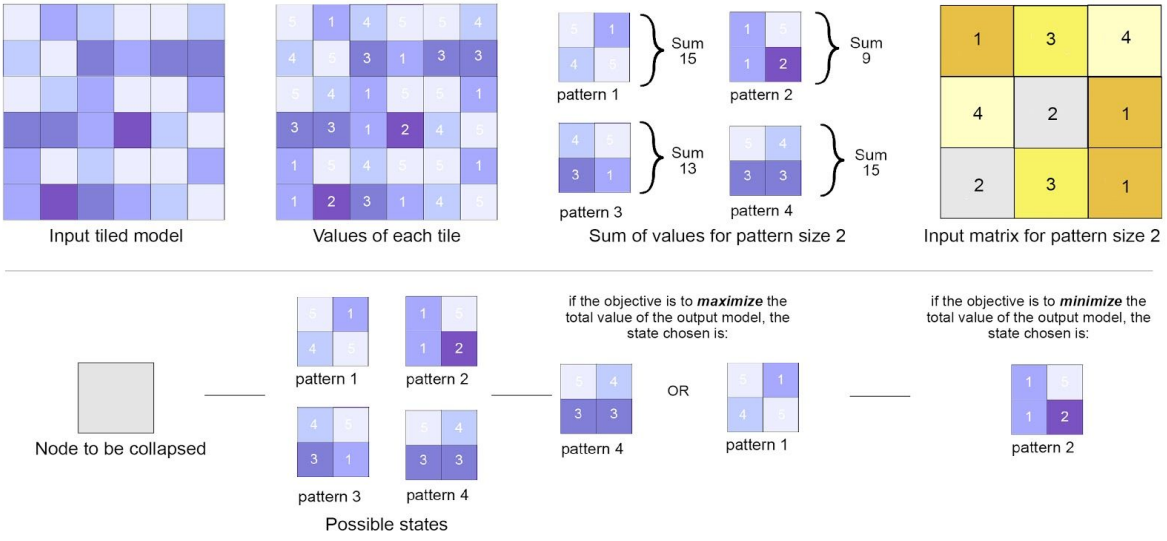


Figure 20: Example input model and altered decision making for node collapse

3.4.Output decoding process

The decoding process described in Pseudocode 4, is the last step of the proposed computational framework. In this step, the output from the WFC is converted back to mesh geometries and then those geometries are placed in the correct positions. The decoding step is the inverse process of the encoding process. It de-serializes the unique numbers into their corresponding mesh tiles. After the list of digits is converted into a list of mesh geometries, the centers of the output voxels are used for the meshes to be placed in the right positions.

The final result is a list of mesh geometries and a list of points that indicate their positions. The list of points is essentially the 2d or 3d output matrix with a size defined in the WFC step of the process. The number of mesh geometries should be equal with the number of points, but it is possible that some positions of the output matrix are not filled with points and left empty.

The decoding step is implemented as two separate GH components, but it could also be contained into one. The full list of developed components is presented in Appendix A.

Algorithm : Decoding

Data: OutputBoxes, OutputIdentities,Encoded List,InputBoxes,InputMeshes

```
Find the unique items in the OutputIdentities List;
Add the unique items in a UniqueItems List;
for (Every item in the UniqueItems List) do
    Add to an Indices List the index in which the item appears in the Encoded List
end
for (Every integer ind in the Indices List) do
    Add to a MeshCollection List the InputMeshes mesh in index=ind;
    Add to a BoxesCollection List the InputBoxes box in index=ind;
end
for (int i = 0; i < Outputvalues.Count; i++) do
    for (int j = 0; j < Uniquevalues.Count; j++) do
        if (OutputIdentities[i] == UniqueIdentities[j]) then
            Add to an OutputMeshes List the MeshCollection mesh in index=j;
            Add to an OutputCenters List the center of the BoxesCollection box in index=j;
        end
    end
end
```

Result: (OutputMeshes, OutputCenters)

Pseudocode 4: Decoding Process

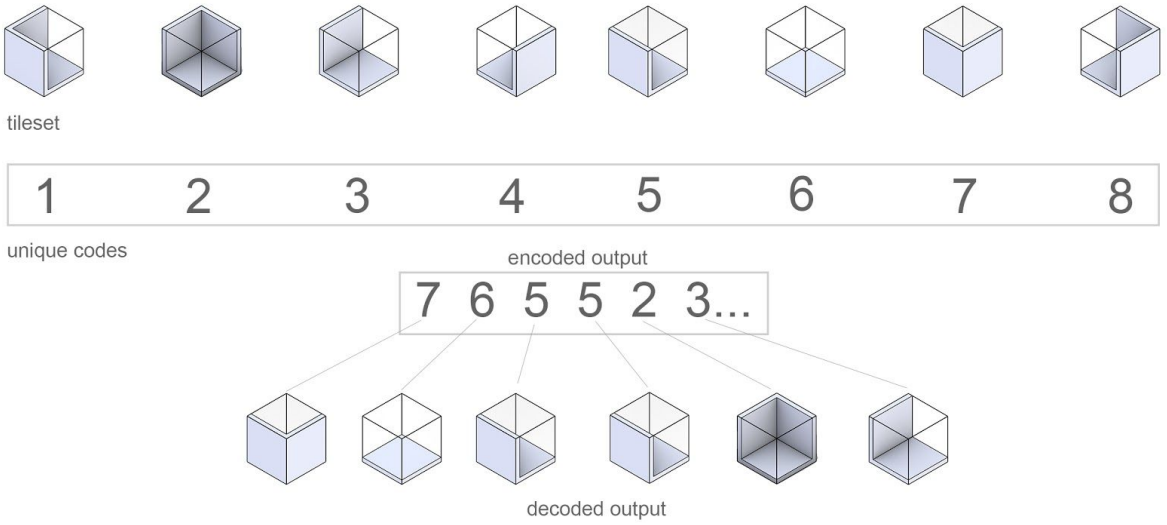


Figure 21: Example tileset, the corresponding code numbers, and the encoding process

Figure 21 illustrates the encoding process for a tileset of eight unique tiles and a small output. The numbers of the output are referenced back to the initial list of unique codes in order for the mesh geometries to be retrieved. For the same number, the same mesh geometry is retrieved and the process is repeated until the full mesh list is created.

4. Results

The Results Chapter presents the experimentations performed using the developed computational framework. The first section briefly discusses the design process of the tilesets. For every following section, a different tileset is modeled and different hypotheses are tested using that tileset. In the previous chapters, 1D and 2D input models were used to explain different steps of the design process and illustrate the algorithmic performance. In this chapter, 3D tilesets will be used with a focus on building massing design since this was the initial objective of the present dissertation. Section 4.6 documents the time performance of the implemented algorithmic variations.

4.1 Tileset Guidelines

The tileset design is essential for the production of interesting building massing results. The main limitation of the tileset design is that every tile should be contained by a voxel. However, there is no restriction on the shape of the tiles themselves. Every tile should also be a joined mesh but the developed workflow can be easily adapted to work with NURBS surfaces as well. An alternative solution, working with multiple geometries per voxel, could offer bigger flexibility. Figure 22 illustrates the design steps followed for the production of tilesets that will be followed in the next sections. The first step is the design of different building elements that are assembled into one building unit. The elements such as slabs, walls, balconies, and shades take into consideration the boundary volume of the voxel. After every building unit is assembled, it is joined into a mesh and a bigger aggregation is created. The encoding is applied in the total aggregation and then manual application of material follows.

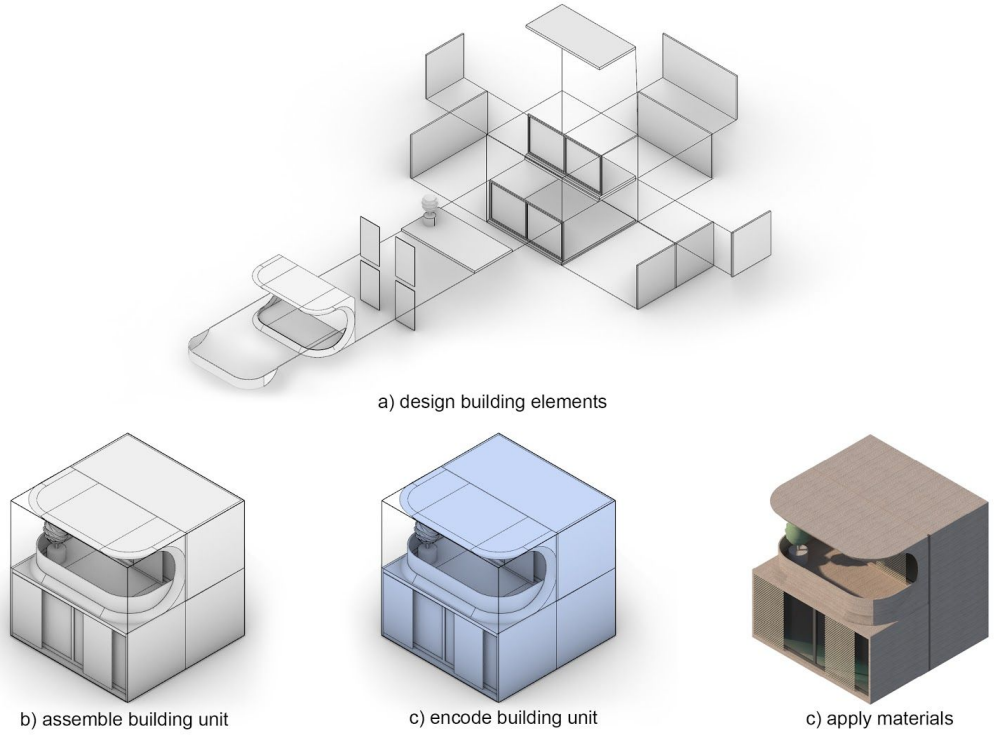


Figure 22: Main design steps of building units: design of elements, unit assembly, unit encoding, and material application

The encoding step identifies the unique tiles from the way they intersect the faces of their voxel. As a result, rotations of the same tile are recognized as distinct tiles since they don't intersect the voxel's faces in the same order. The fact that rotation permutations of one tile are recognized as different is significant for the concept of directionality that will be further discussed in section 4.2.

While experimenting, one main issue that emerged was the recognition of tilesets that intersect all the faces in the same way and consequently generate the same code no matter their rotation. A practical solution to that issue is a small scale of the square tile in one of its directions to detach it from one of the voxel faces. The encoding process is sensitive enough to capture that small difference and generate a separate code for that tile.Figure 23 describes the way the architectural tiles are "perceived" by the encoding algorithm and how their binary codes are generated per face. It also illustrates how a small scale inwards in one of the tiles can affect the generated binary codes.

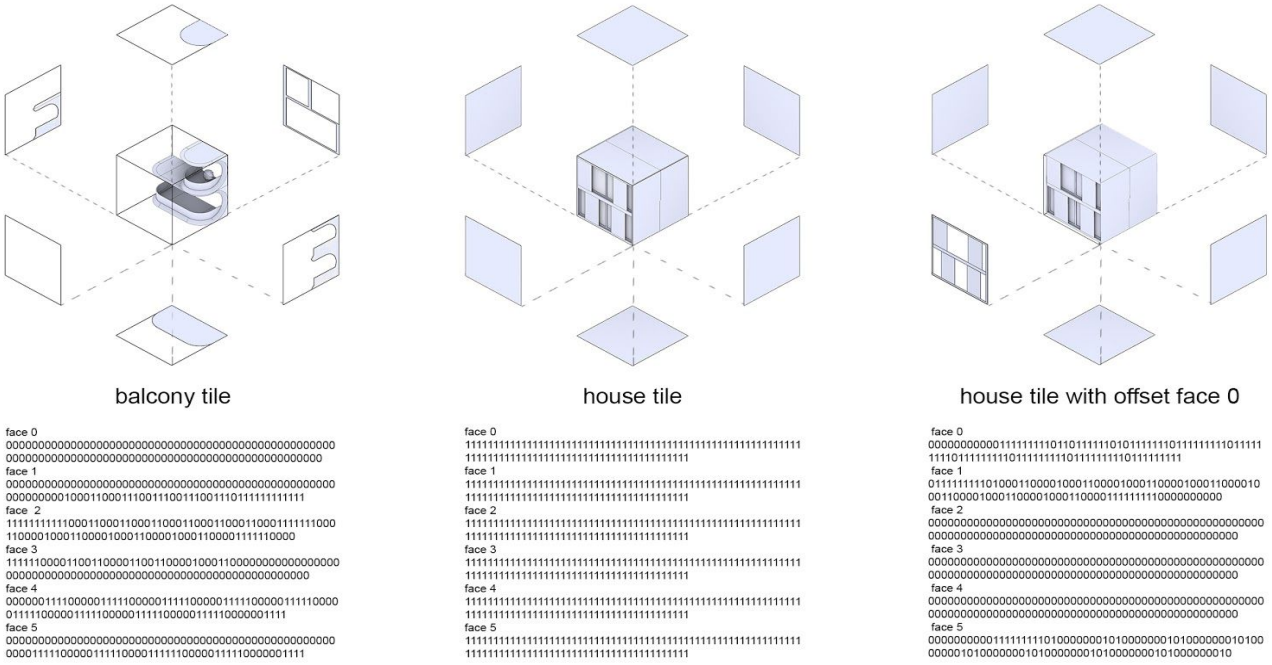


Figure 23:Encoding process of balcony tile, house tile and house tile with offset side

4.2. Maintaining the directionality of the input model

The tileset designed to investigate the ability of the workflow to preserve the input model directionality is presented in Figure 24. A basic housing tile was created with two possible options for the tile that can be placed on its front. The first option is a voxel with a balcony tile and the other is an empty voxel. Two more options for housing units were also designed. Those units have the balcony integrated into their volume and always an empty voxel in their front. That limited definition of direction relationships was chosen to clearly observe and document the behavior of the algorithm.

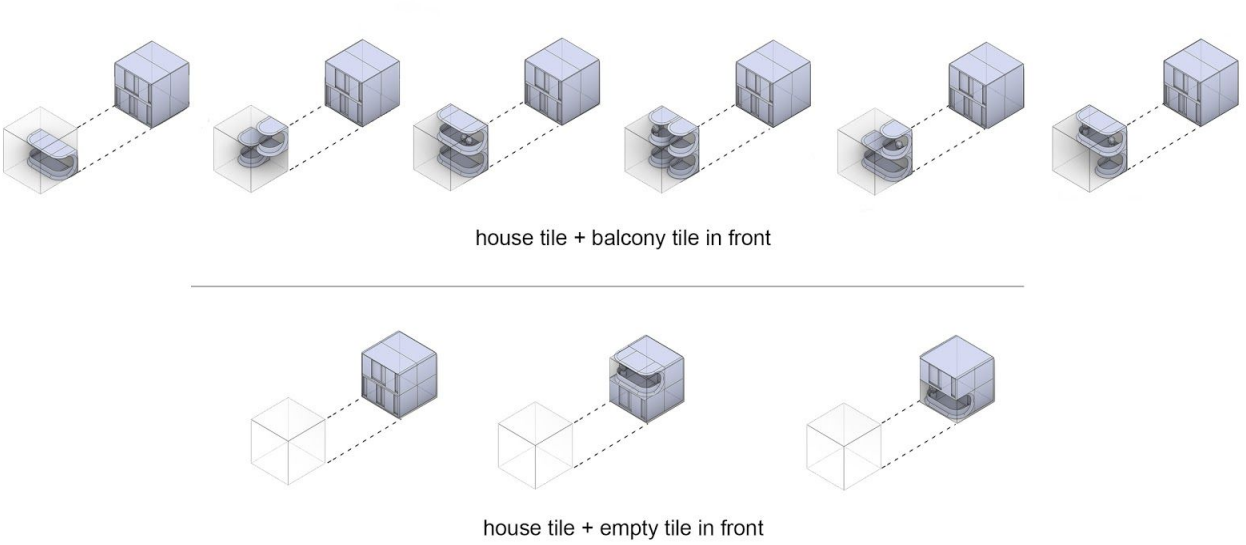


Figure 24: Tileset with direction constraints

To test the tileset, an input of 4 voxels in x,y, and z directions was created. The input has two facades with balconies, the front and back, and two “blind” facades. Such a design would be suitable for the design of a building between other buildings. The input model and sample outputs are presented in Figure 25.

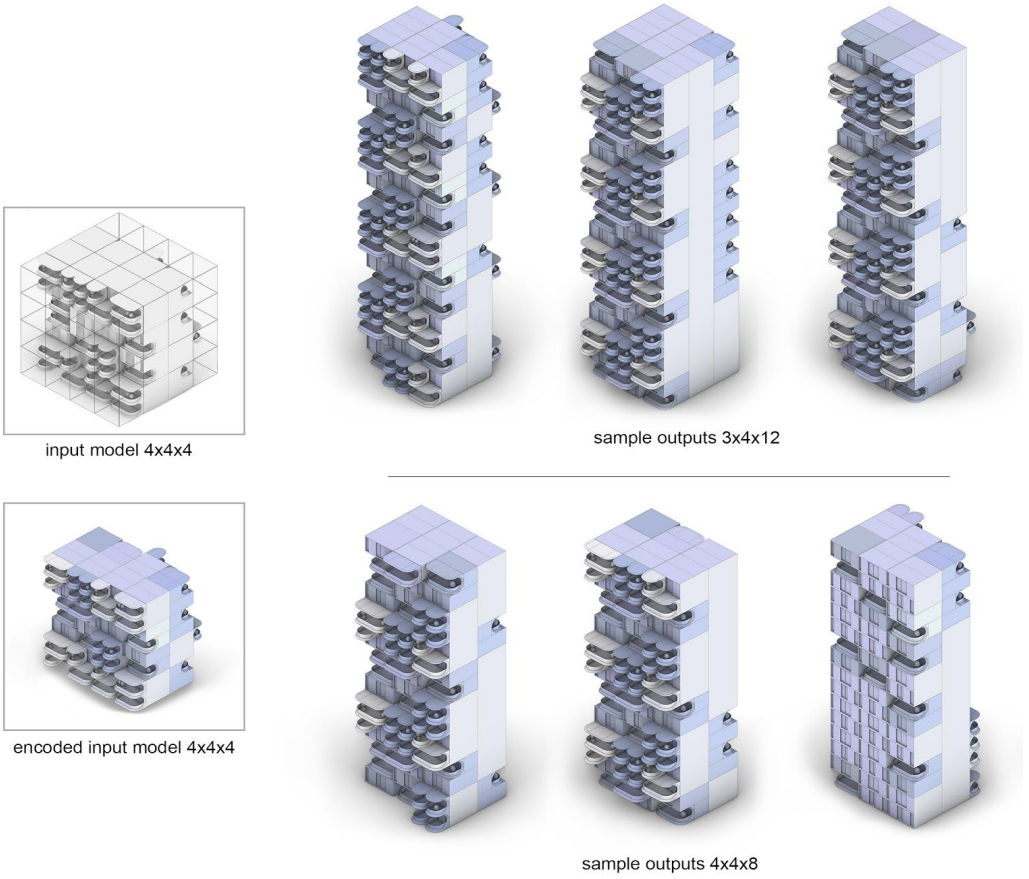


Figure 25: Simple tiled model input and output models with constraint directionality

As expected, since no tiles with balconies or windows appear in the sides of the input, that is also the case in the output. To ensure the preservation of this characteristic in the output model, periodic adjacencies should be avoided. The definition of the input as periodic allows periodic adjacencies. Tiles that exist in the outer sides of the input model can be placed next to each other resulting in unwanted placements of tiles, for example, a balcony tile in front of another balcony tile.

From the above experiment, it is safe to conclude that maintaining the directionality of the input model is possible using the proposed methodology. It is also possible to get output models of a variety of sizes and proportions that present facade variation. In Figure 26, the input model and two outputs of different sizes are illustrated with indicative materiality.

Figure 26: Simple tiled model input and outputs of (4x4x8) and (3x4x12) sizes

4.3. Maintaining a gradient value

In this section, an input model with a gradient facade resolution was tested. The model, with a bottom-up direction, is created with units that are subdivisions of the ones existing under them. The first two layers of the input are created as cubical units, the two above are created as a cubical unit vertically subdivided in half and the next two as the previous unit subdivided again horizontally. Additionally, the voxels at the right top corner of the model are missing giving a dissolving impression.

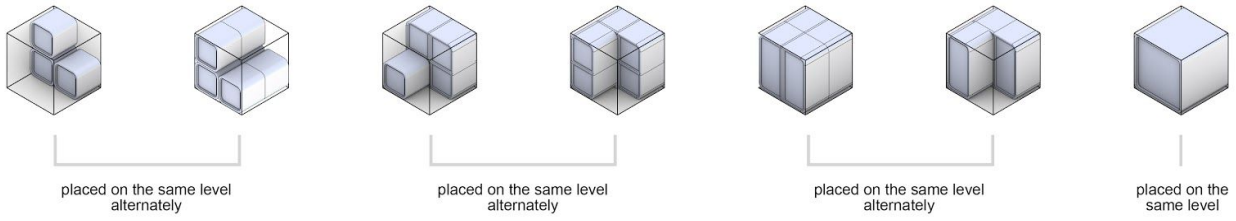


Figure 27: Tileset for input model with a gradient change

For the design of the input model, the tiles presented in Figure 27 were placed in zones, with every zone having two different tiles except from the first one. In Figure 28, the gradient impression of the facade is illustrated as well as the tiles placed in each zone.

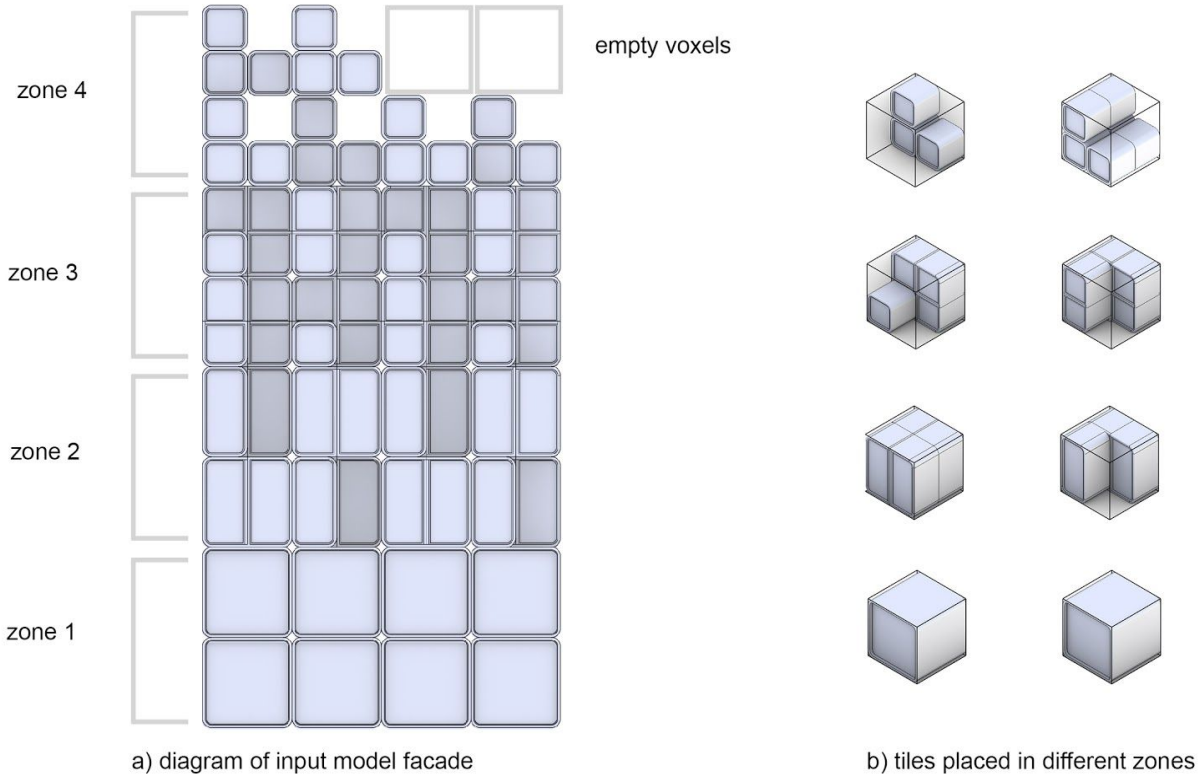


Figure 28: Gradient change on the input model facade

In Figure 29, a series of sample output models is presented. In only probabilistic outputs, the direction of the gradient was preserved, and the output models always started with less subdivided tiles in the first levels. More subdivided tiles appeared in the upper ones. When periodic adjacencies were allowed, the gradient was again preserved in terms of adjacencies (it was not shuffled) but the direction of the gradient was not. The periodicity of the allowed adjacencies is the one responsible for this effect. Finally, in Figure 30, the input model and sample output model with indicative materiality are presented.

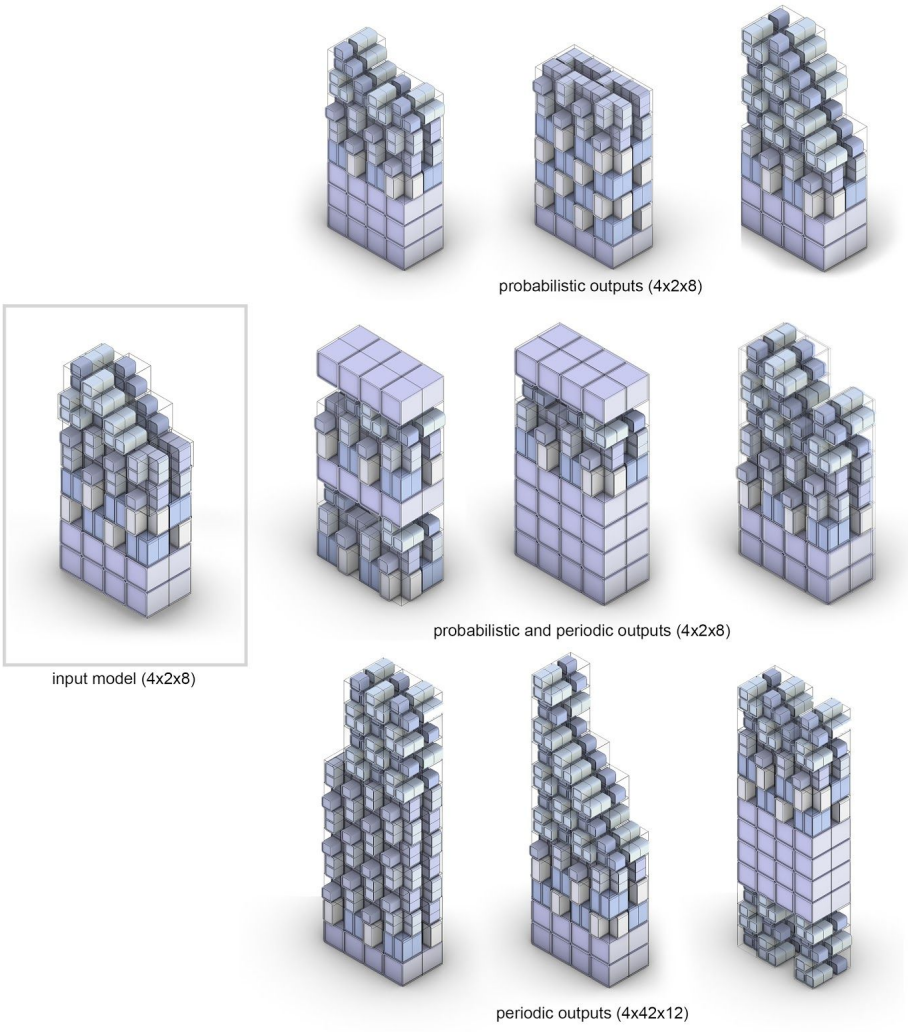


Figure 29:Simple tiled Wfc input and outputs with gradient resolution

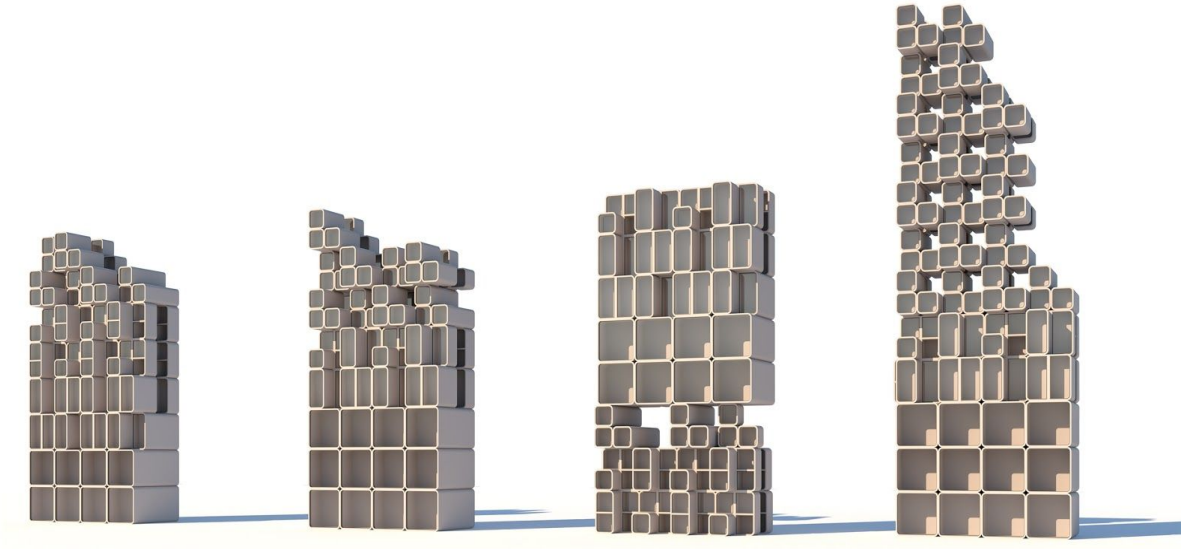


Figure 30: Inputs and outputs with indicative materiality

4.4. Minimizing or Maximizing the density

The experiments of section 4.1 and section 4.2 were based on a simple design decision taken in the tileset design step. Since the WFC is agnostic, it can be assumed that a better tileset can produce a better input model and, consequently, better output models. The quality of the output model is measured based on how close it is to the user's guidelines and how consistent it is based on the defined constraints. In this section, the variation of the WFC presented in section 3.3.2 was used to produce output models with an additional objective. In the form of a question, the experiment could be the following: *"Given an input model, can we produce a variety of output models that will attempt to maximize or minimize a qualitative value of the input model?".*

Figure 31: Architectural geometry tileset and the corresponding volumes

For this experiment, a tileset of simple rectangular volumes was created. Every tile was also assigned a number that equals its volume. The information obtained by the algorithm can be expressed with the following statement: *"All the L shaped tiles are given the number 2, and they have a volume of 0.75 m3".*

This information, known for all the tiles, was sufficient to produce a consistent model and attempt to minimize or maximize the total volume of it. After the output model was created, the simple volumetric tileset was replaced with the equivalent tileset of architectural geometries as seen in Figure 31 and 32.

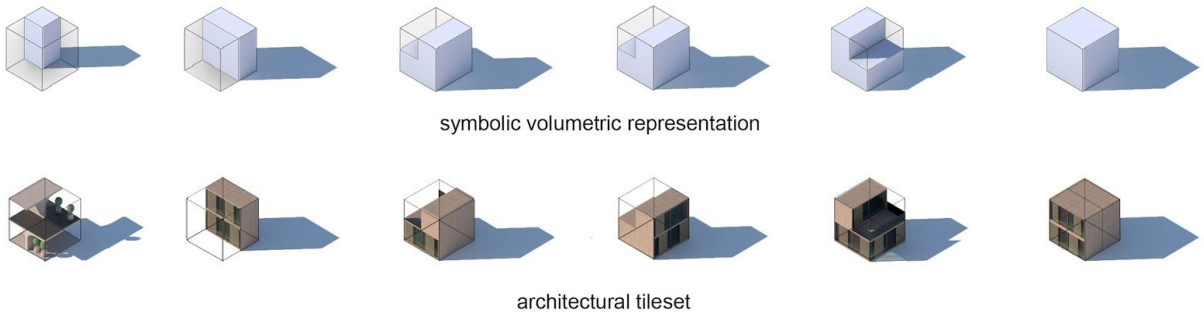


Figure 32: Volumetric tileset and the equivalent architectural geometry tileset

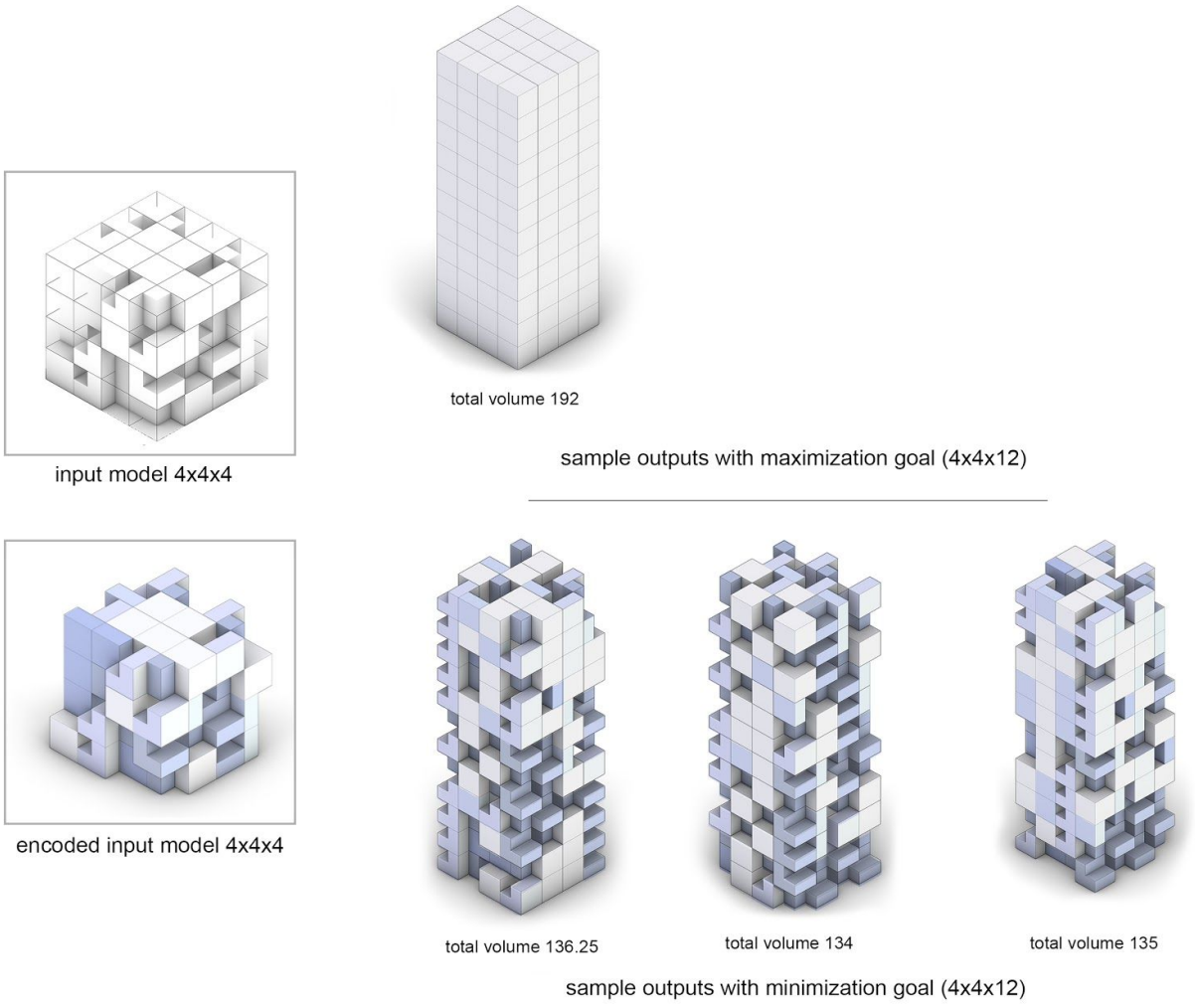


Figure 33: Simple tiled WFC input and outputs with minimization/maximization goal

In Figure 33, the input and outputs of the algorithmic process are presented. When the goal was to minimize the total volume, a variety of outputs was produced. In contrast, when the goal was the maximization, only the maximum volume tile was used. Even though the resulting aggregation is the one with the maximum volume, it might not be desirable due to a lack of variation. As a remedy, a similar input model was created with the maximum volume tile not placed adjacent to its identical ones in all the possible directions.

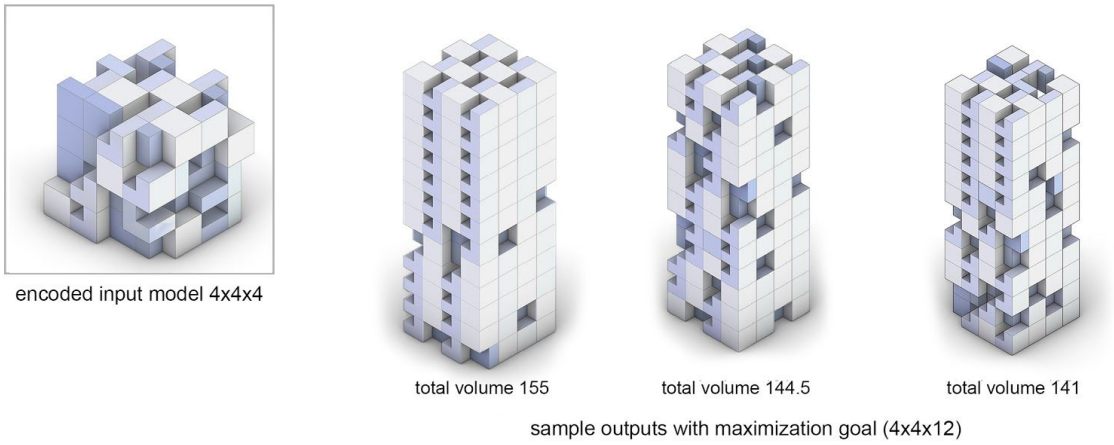


Figure 34: Output models with maximization goal after changing tile repetition in the input model

The limitation of adjacent possibilities for the *maximum volume tile* had a drastic effect on the produced outputs. The algorithm chooses the *maximum volume tile* when it is possible, but it could not be placed in all six possible adjacency directions. This forced the algorithm to choose other tiles instead. In Figure 35, sample outputs with materiality are presented.

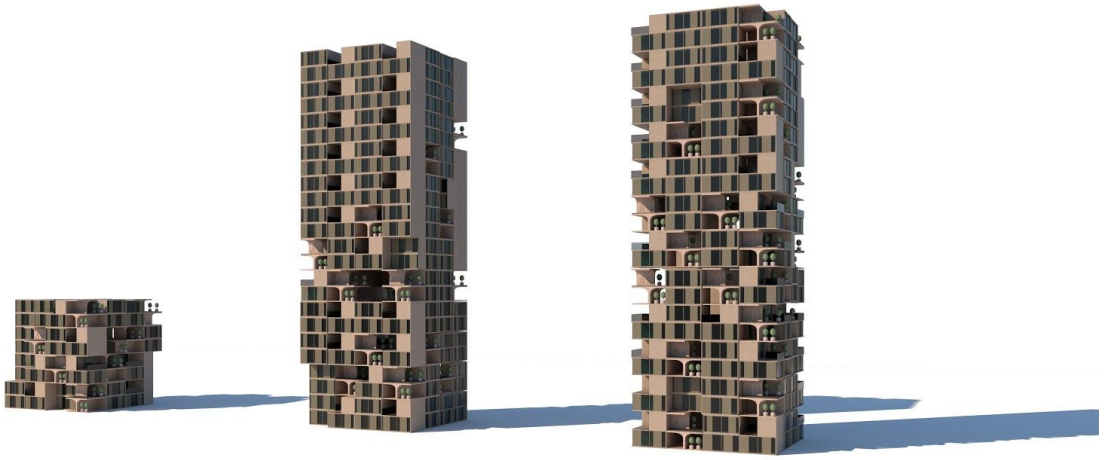


Figure 35: Goal-oriented WFC input (left) and outputs for maximization and minimization

4.5. Applying building program constraints

The basic implementation of the WFC Algorithm usually includes the option to use the probabilities distribution of the patterns in the input model and, based on them, construct the output model. Whether the probabilities will be used or not is a decision, the user should make and has an immediate effect on the produced outputs. The use of probabilities was discussed in section 3.3.1.

For this experiment, the probability distribution of the input model patterns was ignored, and instead, a designer desired set of percentages was inputted. The production of the output models remained probabilistic, including some randomness, but altering the probabilities was an attempt to control the results and evaluate them. In real-life design projects of modular buildings, the architect might have to define the percentages or the number of the different modules. The difference of the modules could be only their allocated use (housing module, office module, etc.), or they could also have a distinct design. The following tileset, illustrated in Figure 36, includes seven units visually different from the others. Each unit was associated with the percentage it should appear in the output model.

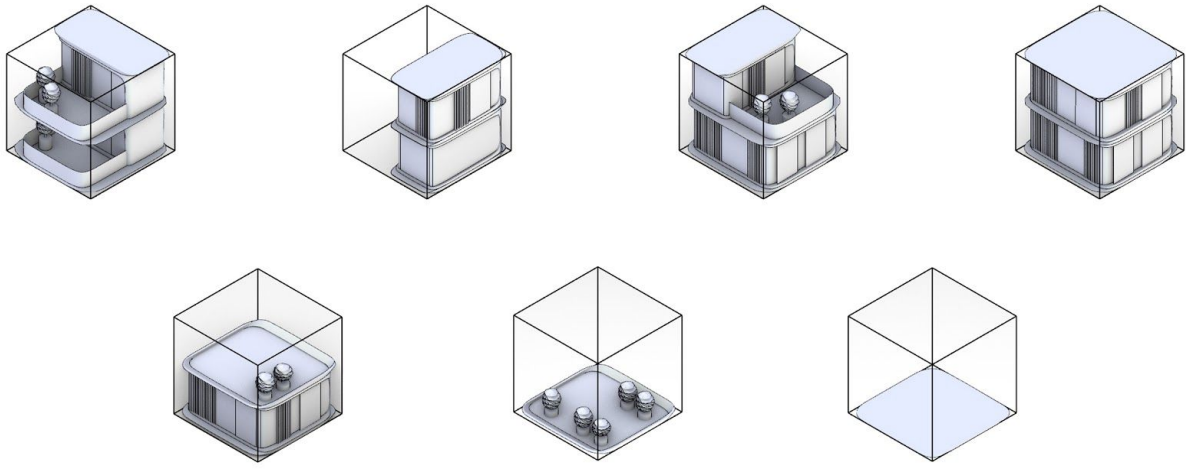


Figure 36: Tileset of different unit types for application of building program constraints

For the following experiment, two main alterations were made to the algorithmic implementation.

1. The addition of the user-defined probabilities for every tile as a new input.
2. The hard coding of the pattern size in number one.

As already discussed, the WFC can work with a wide range of sizes of patterns. In this experiment, the pattern size was limited to 1 to produce more readable results and for user experience purposes. In Figure 37, a simple input model with three different tiles was tested. The majority of tiles in the input model are the tiles of category 1. The second tile appearing the most is the one-floor housing unit named tile number 2, and the last one is the L- shaped housing unit. By altering the probabilities, it was attempted to minimize the tiles of type 1 and maximize the tiles of type 3 while type 2 tiles retain the same probability.

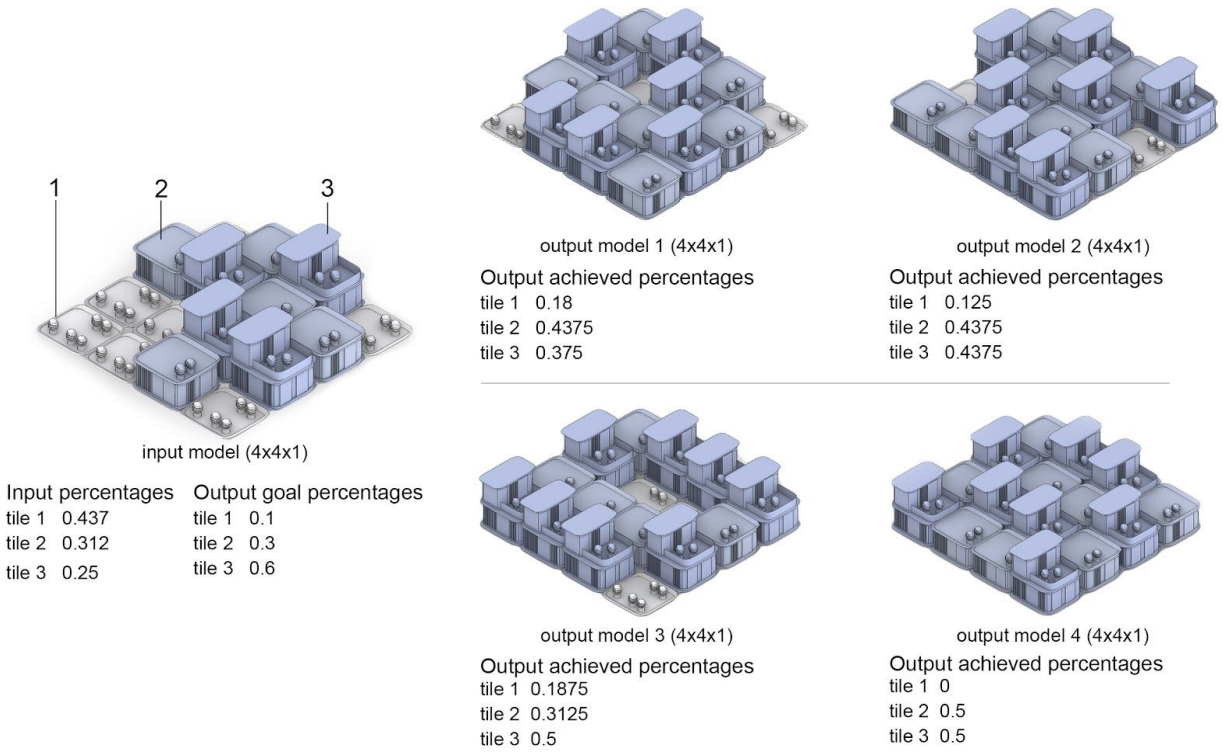
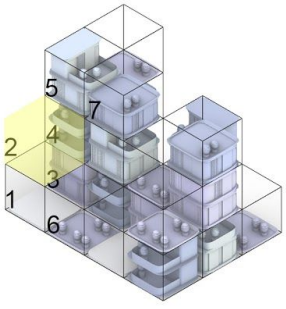


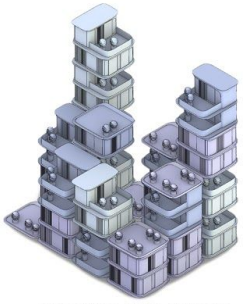
Figure 37: Input model and outputs with defined percentages of occurrence for the different patterns

The findings of the above experiment indicate that the algorithm can work with probabilities different from the ones extracted from the input and can achieve percentages close to the ones defined by the user. The allowed adjacencies and the random starting point of the algorithm can also affect the achieved percentages that are not always the optimum as in output model 4 of Figure 37, where the tile of type 1 does not appear at all in the output. In Figure 38, a bigger input model was tested. The model has seven different tiles, with one of them being the *empty voxel tile.* The tile numbered 2 is the way of the algorithm to capture the absence of voxels to recreate it in the output model.



input model (4x3x4)

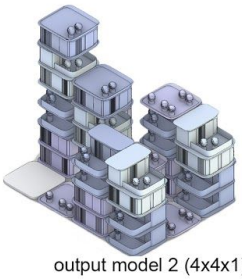
Input percentages		Output goal percentages	
tile 1	0.041	tile 1	0.05
tile 2	0.52	tile 2	0.05
tile 3	0.083	tile 3	0.3
tile 4	0.083	tile 4	0.3
tile 5	0.104	tile 5	0.1
tile 6	0.124	tile 6	0.1
tile 7	0.041	tile 7	0.1



output model 1 (4x4x1)

Output achieved percentages

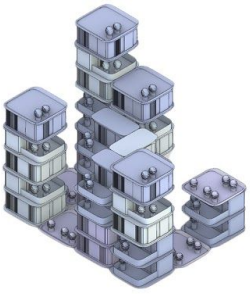
tile 1	0	tile 5	0.1875
tile 2	0.416	tile 6	0.125
tile 3	0.375	tile 7	0.125
tile 4	0.125		



output model 2 (4x4x1)

Output achieved percentages

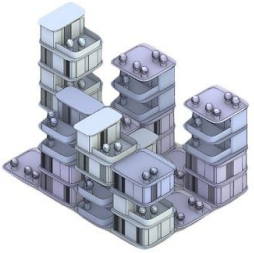
tile 1	0.0208	tile 5	0.0833
tile 2	0.5	tile 6	0.1041
tile 3	0.1041	tile 7	0.0416
tile 4	0.125		



output model 3 (4x4x1)

Output achieved percentages

tile 1	0	tile 5	0.166
tile 2	0.5	tile 6	0.0833
tile 3	0.0625	tile 7	0.0833
tile 4	0.1041		



output model 4 (4x4x1)

Output achieved percentages

tile 1	0	tile 5	0.1042
tile 2	0.5416	tile 6	0.1458
tile 3	0.0625	tile 7	0.0208
tile 4	0.1042		

Figure 38: Input model and outputs with defined percentages of occurrence for the different patterns

In the above test, an attempt to minimize the empty voxels and maximize the presence of tiles 3 and 4 was made. The results were less successful compared to the ones presented in Figure 37.

The algorithm did not manage to minimize the empty voxels with the best percentage achieved to be the 0.416 of the output model 1. In terms of increasing the percentages of tiles 3 and 4, the process was more successful but didn't produce steadily better results.

The experiments done with user-defined probabilities prove that the algorithm is able to produce output models with different probabilities than the ones of the input model. The random initialization of the algorithm defines whether the desired percentages will be reached or not. It is also possible that the legal adjacencies of the patterns might not allow for a certain percentage to be achieved. For example, the allowed adjacencies of a pattern might forbid its extensive replication on the output model.

In Figure 39 the input model designed for the second experiment, and resulting outputs are presented with indicative applied materials.

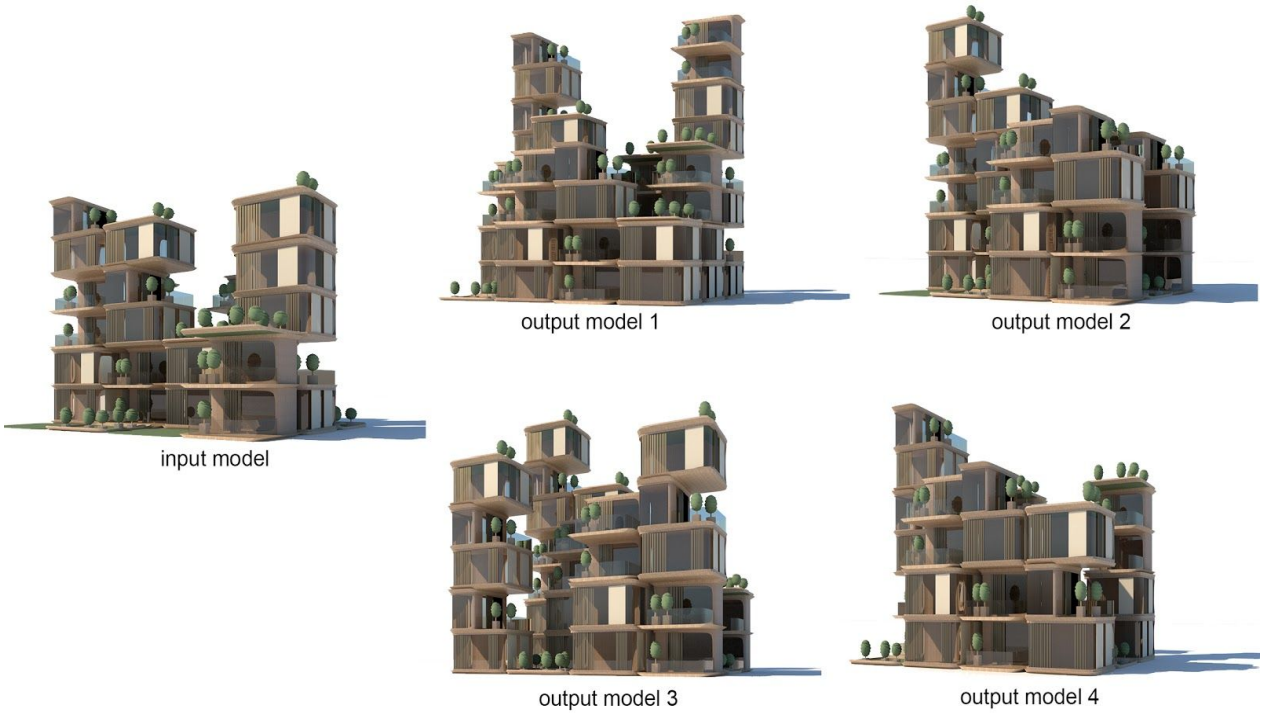


Figure 39: Input model and outputs with applied materiality

4.6 Time performance

In this section, the algorithmic performance of the different implementations was documented.For consistency, a model of size (3x3x4) was designed and used for the production of the following figures.In Figure 40, the performance of the simple tiled and the overlapping model is compared for the production of a (3x6x24) output.For ten restarts of the algorithm both models responded in less than 15 milliseconds with the overlapping model being slower in all of the ten attempts.The time difference between the responses of the two models was minor and not perceived by the user.

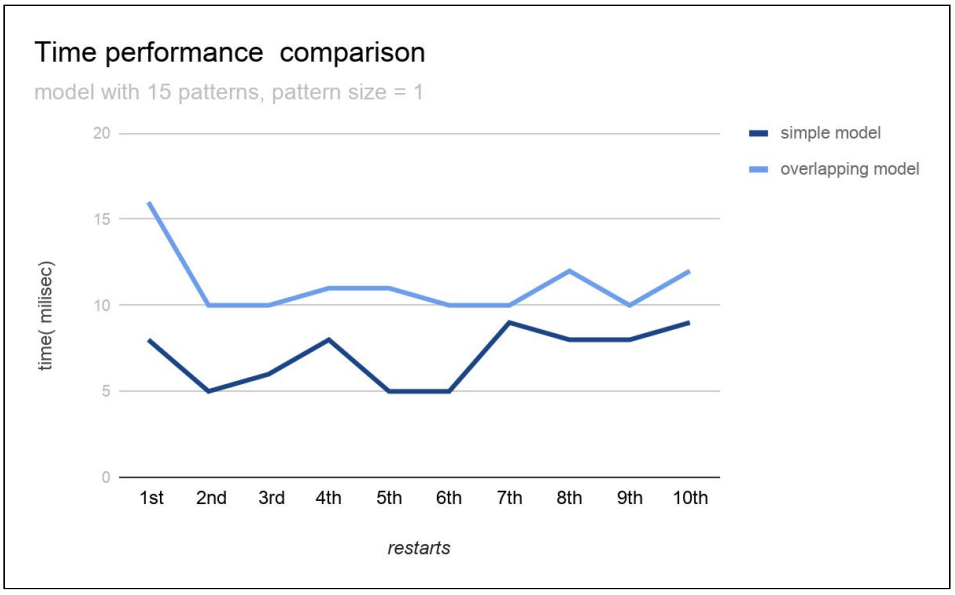


Figure 40:Time performance comparison between the simple and overlapping version of the WFC for a 3x3x4 input and 3x6x24 output model

The same model was used to document the effect of the number of patterns to the algorithmic time performance .The overlapping model was tested twice for the same input, once with a defined pattern size equal to one and once with a pattern size of two.That change drastically affected the inferred patterns and increased their number from 15 to 27 ms.The response of the algorithm became significantly slower but in most cases still not perceived by the user.The results are presented in the Figure 41.

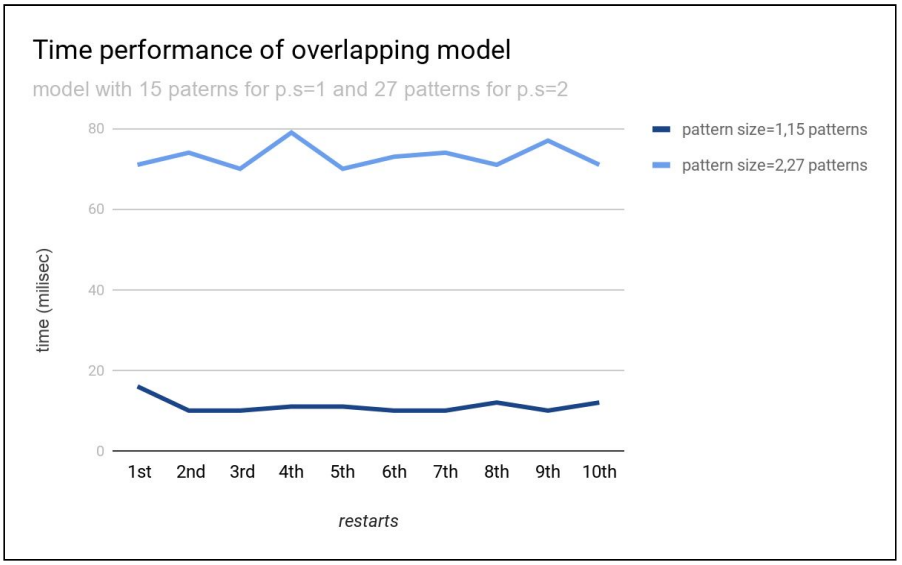


Figure 41: Time performance comparison of the overlapping model for a 3x3x4 input, 3x6x24 output and different number of patterns

Finally, an overall comparison was made between all the implemented variations of the WFC.As illustrated in Figure 42, for pattern size=1, the simple model is responding faster and the version of the algorithm with user-defined percentages is the one performing slower.The version that aims to the maximization or minimization of a total value performs equally fast for both objectives.The difference between the overlapping model and the simple one is not significant for a pattern size = 1 but, as seen in the previous figure the increase of the pattern size affects the performance.

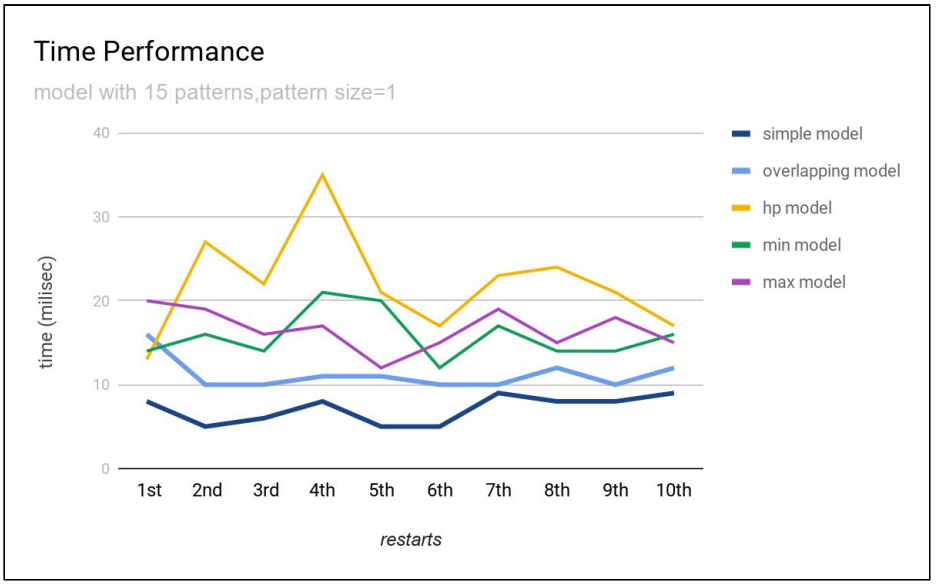


Figure 42: Time performance comparison of the all the implemented variations,hp model is the model with defined percentages for housing program,min model and max model is the same variation with different objectives

5. Conclusions

The majority of current architectural projects are characterized by increasing complexity and depend on several parameters. This fact makes it necessary to use digital tools to assist in the design, decision making, and optimization. Automating repetitive design tasks and the production of design variations is also an integral objective that can contribute to time saving and better communication among the involved parties.

The main objective of this dissertation was to propose a design methodology for the design of building massings using architectural tilesets and the WFC. The proposed computational framework is integrated into a design software (Grasshopper for Rhino3d) as an attempt to address issues related to the user experience.

Based on the research and the results presented in the previous sections, the proposed methodology proved appropriate for the design of massings using architectural tilesets. The system can identify the different tiles and produce a series of aggregation variations. The constraint variations of the WFC Algorithm presented in sections 4.4 and 4.5 adapt better to the design objectives of a project and produce less unexpected results. They allow the navigation of the design space focusing on specific design parameters and produce more targeted results.As presented in section 4.6 the different variations implemented are also efficient in computational time with the number of different patterns being the most significant factor for time performance.

A second objective was to provide insight into the algorithm and guidelines for more efficient use of both the algorithm and the proposed workflow. This was done by technically describing the steps of the framework and by exploring and documenting the results when changing the user-defined parameters. Another contribution is the guidelines on the design of the tileset.

6.Future Work

Future work concerns further experimentation with the proposed workflow and new variations of the algorithmic process.The core algorithm could be divided into two main parts. The input and output matrix construction and the decision making happening in between. The following are ideas that could be implemented in both parts:

1. The regular grid that the proposed workflow is using, both for the input and output matrix, is generally applicable in architectural design. However, working with a version of WFC that is based on an irregular grid or a grid of voxels with different sizes could lead to appealing results.
2. The decision making of the algorithm is mostly stochastic. When intervening in the decision-making, interesting results can emerge. Similar work to sections 4.4 and 4.5 and the integration of additional constraints can lead to better adaptation to architectural design objectives.
3. A step further from the above point is combining a WFC implementation with machine learning techniques operating at the decisional level. This exploration can illustrate how the inception of an AI system can lead to a cognitive understanding and deployment of emergent qualities in a created model.

Bibliography

Artemel, A. (2017, November 06). How Video Games Use Architecture - Architizer Journal. Retrieved September 11, 2020, from https://architizer.com/blog/inspiration/industry/how-video-games-use-architecture/

Barriga, N. A. (2019). A Short Introduction to Procedural Content Generation Algorithms for Video Games. *International Journal on Artificial Intelligence Tools, 28*(02), 1930001. https://doi.org/10.1142/S0218213019300001

Blow, J., (2016) The Witness.[Computer video game].Thekla.Inc

Blockworks (n.d) Retrieved August 1, 2020, from www.blockworks.uk

COMP 200: Elements of Computer Science Spring 2013. (n.d.). Retrieved September 10, 2020, from https://www.clear.rice.edu/comp200/13spring/notes/18/shaney.shtml?cv=1

Efros, A.a., and T.k. Leung. (1999) “Texture Synthesis by Non-Parametric Sampling.” Proceedings of the Seventh IEEE International Conference on Computer Vision. https://doi.org/10.1109/iccv.1999.790383.

Hypar. (n.d.). Retrieved August 19, 2020, from https://hypar.io/

Gagniuc, P. A. (2017). Markov chains: From theory to implementation and experimentation. Chichester, West Sussex: Wiley Blackwell.

Gumin, M. (2015). WaveFunctionCollapse. Retrieved June 2, 2020, from https://github.com/mxgmn/WaveFunctionCollapse

Ijiri, T., Mêch, R., Igarashi, T., & Miller, G. (2008). An Example-based Procedural System for Element Arrangement. *Computer Graphics Forum, 27*(2), 429-436. DOI:10.1111/j.1467-8659.2008.01140.x

Inc, T. (n.d.). Home. Retrieved August 19, 2020, from https://blog.testfit.io/testfit-home

Karth, I., & Smith, A. M. (2017). WaveFunctionCollapse is constraint solving in the wild. *Proceedings of the International Conference on the Foundations of Digital Games - FDG '17*. DOI:10.1145/3102071.3110566

Khokhlov, M. (2017). Kazimir. Retrieved June 2, 2020, from https://github.com/MatveyK/Kazimir

Kleineberg, M. (2018). Github.Accessed June 09, 2020, from https://github.com/marian42/wavefunctioncollapse.

Lecoutre, C. (2009). *Constraint networks: Techniques and algorithms*. London.ISTE.

March, L. (2014). Mathematics and Architecture Since 1960. *Architecture and Mathematics from Antiquity to the Future,* 553-578. DOI:10.1007/978-3-319-00143-2_38

Merrell, P. C. (2009). Model Synthesis (doctoral dissertation). The University of North Carolina, Chapel Hill.

Minecraft Official Site. (2020, May 22). Retrieved June 09, 2020, from https://www.minecraft.net/en-us

Miller, R. & Miller, R. (1993). Myst [Computer software].Brøderbund.

Özkar, M. (2015). Rule-Based Design, in Honour of Lionel March. *Nexus Network Journal, 17*(3), 693-696. DOI:10.1007/s00004-015-0272-6

Parker, J., Jones, R., Morante, O.(2016). Proc Skater. Retrieved July 28, 2020, from https://arcadia-clojure.itch.io/proc-skater-2016

PRISM. (n.d.). Retrieved September 08, 2020, from https://www.prism-app.io/

Robert McNeel & Associates. (1998). Rhinoceros 3D. [Computer software].Robert McNeel & Associates.

Rutten, D. (2007). Grasshopper 3D. [Computer software].Robert McNeel & Associates.

R/3dsmax - Parametric vs Procedural - what's the difference? (2020). Retrieved July 10, 2020, from https://www.reddit.com/r/3dsmax/comments/g054t4/parametric_vs_procedural_whats_the_difference/

Savov, A., Winkler, R., & Tessmann, O. (2019). Encoding Architectural Designs as Iso-surface Tilesets for Participatory Sculpting of Massing Models. *Impact: Design With All Senses,* 199-213. DOI:10.1007/978-3-030-29829-6_16

Scarf, F., Ruppal, H., & Norman, M. (2020, August 14). Property Development Software: Property Feasibility Software. Retrieved August 19, 2020, from https://archistar.ai/

Smith, Gillian (2015). An Analog History of Procedural Content Generation (PDF). Foundations of Digital Games 2015. Pacific Grove, California. Retrieved October 7, 2019.

Stålberg, O., (2020). Townscaper.[Computer video game]. Oskar Stålberg.

Stålberg, O., and Meredith, R.(2018). Bad North.[Computer video game].Raw Fury.

Steadman, P., (2016) Research in architecture and urban studies at Cambridge in the 1960s and 1970s: what really happened, The Journal of Architecture, 21:2, 291-306, DOI: 10.1080/13602365.2016.1165911

Tsang, E., & Fruehwirth, T. (1996). *Foundations of constraint satisfaction*. Norderstedt: Books on Demand.

Yar, S. (2018, September 20). For a Community of Gamers, The Sims Is a Gateway to Architecture. Retrieved July 04, 2020, from https://www.metropolismag.com/architecture/sims-architecture-showusyourbuilds-gamers/

Wei, L., & Levoy, M. (2000). Fast texture synthesis using tree-structured vector quantization. Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '00. DOI:10.1145/344779.345009

Wirth, M. A. (2020, July 7). Texture Analysis. Lecture presented at the University of Guelph Computing and Information Science.

Appendices

Appendix A - Grasshopper Components implemented by Author

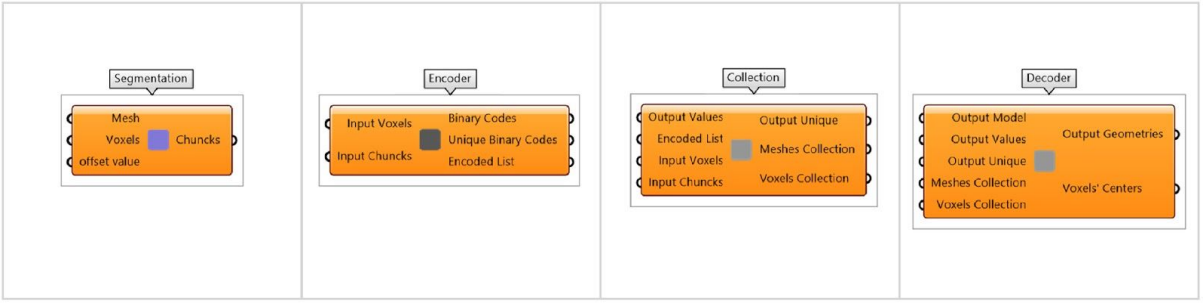


Figure 43: Segmentation, Encoder, Collection, and Decoder GH components

Figure 44: Variations of the WFC implementation as GH components

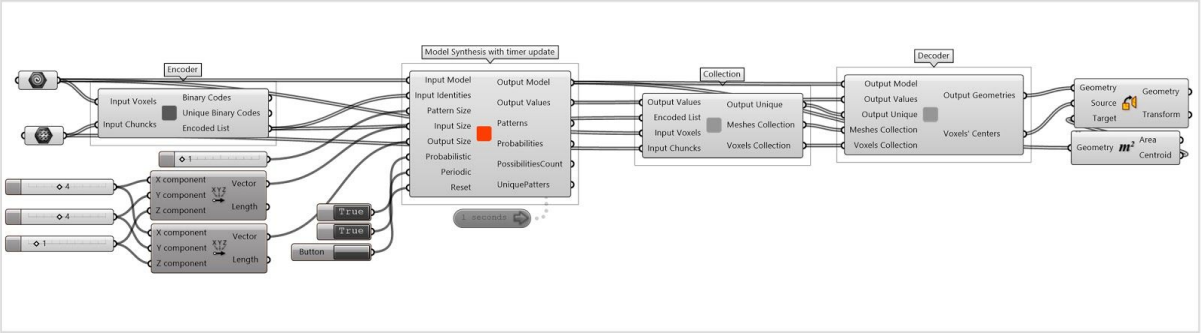


Figure 45: The proposed design workflow as connected components