Houscaper

Developing a tool for easy access
of architectural polygonization
using the Boolean Marching Cubes Algorithm

In partial fulfillment of the requirements
for the degree of Master of Science
in Architecture, Urbanism and Building Sciences
at the Delft University of Technology

Developing a tool for easy access of architectural polygonization using the Boolean Marching Cubes Algorithm

MASTER THESIS – REPORT BUILDING TECHNOLOGY MASTER TRACK
Faculty of Architecture and the Built Environment
Track : Building Technology

First Mentor: Dr. Ir. Pirouz Nourian
Second Mentor: Dr.Fred Veer
External Mentor: Ir. Shervin Azadi
Delegate of the Board of Examiners : Ir. Geert Coumans

Student : Solkyu Park | 5304857

The motivation for conducting the study

Before coming to TU delft, I studied industrial design and worked at a furniture company. While working, my interest was in people's housing space and residential architecture and bringing a better life for people. Living in the metropolis of Seoul, I witnessed house prices doubling and tripling within a few years, which drew more attention to architecture and housing issues. From the experiences, I wondered why architecture could not be as simple and easy to build as furniture and comes with a price tag. Complex required dimensions, safety, insulation, etc., are incomparable to a table. However, I believe if understood the technology and basic concepts used in architecture, similar to the case of furniture, the architecture industry can evolve into an industry that is more affordable and mass-produced through reorganization of the process.

At TU Delft, I developed an interest in computational design and modular construction. Combining the study motivation and interest in the field, in this thesis, I seek to build the modular architectural industry into a more customizable and participatory product for more people with access to the industry. proposing an approach to solve the complexity of construction and demonstrate the potential of modular components. Therefore, the final product I am to develop is a participatory design tool, combined with modularity and computation, which provides combinatorial creativity and transparent CPQ (configure, price, quotation).

Acknowledgement

Big thank you to my mentors Pirouz Nourian, Fred veer, and Shervin Azadi for their support and guidance. Thank you for accepting me as a student for the thesis topic and for understanding the weaknesses and shortcomings of computational knowledge. There were some moments I thought I could not make it till the end, but through the support and wise words, I gained back the courage to try harder and break the boundaries.

Special thanks to the love of my life, Daeun, for supporting me throughout the whole time, taking over the household duties, and financial and mental support.

Many thanks to friends for being a friend to me despite how introverted I am and for the technical support I received from you.

Also, thank the family for always supporting me and trusting me. Without family, this would not have been completed.

Contents

3 Design Methodology

4 Results

5. Conclusion

Acronyms and local terms

MMC Modern Method of Construction

MC Modular Construction

BMC Boolean Marching Cube

BMS Boolean Marching Square

WFC Wave Function Collapse

DFMA Design for Manufacturing and Assembly

CPQ Configure, Price, & Quote (Referring to industrial product configurators)

PCG Procedural Content Generation (Video Games Industry)

Abstract

The demand for lower construction costs comes at the price of duplicating architectural designs. As a result, the misperception that modular architecture is repetitive prevails. However, the developments in computational design and MMC (modern methods of construction) can suggest a standardized manufacturing approach to provide mass customization. The research explores the computation of Modular building envelope systems using the Boolean marching cube algorithm under the premise of a ‘participatory generative design framework’ of the 'GO-Design' developed by Pirouz Nourian and Shervin Azadi [1]. In this project, (1) a set of architectural tilesets is developed that incorporates mass-customization by providing the potential for generating various configurations, and (2) a prototype of an interactive digital tool is established to enable future inhabitants to customize the configuration of their modular houses based on the predefined tileset. The algorithm is developed and applied in the Rhino & Grasshopper environment. The BMC algorithm reads the voxels' labels pertaining to each cube's vertices (a collection of 8 neighboring voxels) to load and choose the corresponding tile from a predefined tileset. The project's focus is the design of such a tileset, its engineering consistency, and architectural coherence. As an input, a voxelated massing will be the input, possibly later accompanied by a colored zoning scheme. and the output will be a modular architectural assembly. This workflow is envisaged to become a procedural component of a Configure, Price, & Quote (CPQ) system for the industrialization of mass customizable housing.

CHAPTER 1

1. Introduction

1.1 Background

The global MC(Modular Construction) market size was USD 72.11 billion in 2020 and is projected to reach USD 114.78 billion by 2028 [2] growth of about 40%. In US 2015 the Modular Building Institute noted that MC constituted 2.9% of all construction and expected it to grow to 6.4% from 2021 to 2028 [3]. Similarly, the UK predicts that MMC(Modern Method of Construction) homes will rise from 6~10% to 20% in 10 years [4]. It is estimated that the increase puts tremendous pressure on the construction industry to shift into modular construction quickly. Choi, professor of University of Nevada predicts that developers will be pushed to consider MC or even be forced to implement it with increased modular construction with reduced price from the economy of scale [5].

While the shift is expected, there are problems with the current modular construction market. Oftentimes, modules in the market are solely optimized for mass production but not for custom configuration. Therefore, the demand for lower construction costs comes at the price of duplicating designs. As a result, the misperception that modular architecture is repetitive prevails. It is essential to mention that adopting a standardized manufacturing approach does not necessarily imply the provision of standardized architecture. It may simply mean benefiting from the manufacturing industry's practices to achieve high productivity rates [6].

Fig.1: example of the duplicated design of modular construction (images sourced from enews)

As technologies advance, the limits and boundaries of modular construction continue to expand. In today's digital world, where powerful computing power is easily accessible to designers in the built environment, mass design customization challenges provide the opportunity to develop digital solutions that can handle complex modularization. Professor Robert Doe of the University of Sydney remarks on the 'Modularity' concerning design informatics that the assembly of parts and their interactions and the configurability of components are concepts common to new computational design tools and improved methods of making architectural solutions.[7]

To achieve this interaction of the components, the thesis works with the BMC algorithm, invented by Lorensen and Cline in 1987 [8]. The original use is to extract a polygonal mesh of an isosurface from a three-dimensional discrete scalar field, mainly concerned with medical visualizations such as CT and MRI scan data images. The application of the algorithm expands from the medical field and is also widely used in computer graphics. In this research, I seek to explore a methodology of design that utilizes a participatory design of small-sized architecture through the BMC algorithm.

1.2 Relevance

Social relevance – In 2019, a survey carried out by the Lincoln Institute of Land Policy (LILP) revealed that 90 percent of the 200 cities around the globe were considered unaffordable to live in [9]. Prices of houses in cities are increasing rapidly, and so is the cost of construction price of new houses due to an increase in material costs and labour wages [10]. Yet, the adoption of the MMC in the industry remains low [6], and The industry's productivity has not risen since the Second World War [11]. The MC is emerging as a solution to the housing crisis, enhancing productivity by shortening the timeline and saving costs. [4] This thesis is socially relevant as the research is searching for a computational method of mass customization of the modern modular construction to bring more people available for more transparency in the architecture industry.

Scientific relevance – Due to its complexity and high price, the process of architectural customization has only remained bespoke, available only to a few people. To break this convention, the thesis aims to develop a mass customization method of architecture through participatory design involving people in the design process and achieve a more transparent CPQ(configure price quotation) of the industry. Since the project is a study of translating the digital information into valid modular and connectable components, the potential benefit of the research can be a link between construction to computer generative or participatory design methods. Connecting the two can suggest a solution to an increase in productivity and the labour crisis that the industry is facing.

1.3 Problem Statement

The traditional construction method is characterized by bespoke solutions delivered by a bespoke process in architectural design [6]. Which is highly customizable but results in low productivity and architecture one of the most expensive purchase in life. Whereas the MC (Modular Construction) in the market is often characterized as standardized solutions provided by a standardized process in architectural design, optimized for mass production but not for custom configuration.

Therefore, the lower construction costs come at the price of duplicating designs, and as a result, the misperception that MC is repetitive and limits the design options or restricts their control prevails. However, contrary to the misperception, the MC can accommodate various forms and styles through mass customization, offering the combinatorial creativity of modules similar to the LEGO blocks.

The design process of mass customization, in the end, can enhance the architect's ability to coordinate the design of manufactured components and enable more creativity in design.

The challenge lies in the assembly of parts and their interactions and the configurability of elements. Unlike the participatory design output of digitalized information or toys, the real-world architectural modules require far more complexity in material, functions, connectivity, and interaction with each part. If computational methods can address the challenges of such complexity, the modular architecture industry will evolve into a customizable mass industry with enhanced productivity.

1.4 Research questions

Research Question –

How to devise a participatory computational design tool for constructing valid modular building frames in a Modern Method of Construction as an assembly of architectural tiles using the Boolean Marching Cubes (BMC) algorithm?

Sub Questions –

- How to develop an interactive tool for normal users to customize their modular architecture from an input voxel array with enough visual representation?

- How can the structural validity check of the frame generate?

- How to offer a set of architectural tiles that incorporates mass customization enabling expert users to develop the tileset?

1.5 Research Objectives

In housing production workflows optimized for mass production but not for customized configuration, the lower construction costs come at the price of standardized products. As a result, the underlying misperception that MC offers repetitive design limits and restricting control. Contrary to the misperception, the MC can suggest a coexistence of a standardized construction process and a bespoke design by achieving mass customization.

In this project, the research seeks a methodology of mass customization through the participatory design of a small-sized residential architecture. The suggested method will contain. **First, a set of architectural tiles** (hereinafter referred to as the tileset) is developed that provides for combinatorial mass customization of building envelopes by ensuring an exhaustive extent of consistency that would allow for procedural cladding of virtually any possible colored configuration of voxelated building designs. **Secondly, the project develops a prototype of an interactive digital tool** that enables future inhabitants to customize the cladding and the envelope structure of their modular houses based on the predefined tileset from the first phase. **the third is the structural analysis of the architecture for the purpose of validity check,** environmental evaluation in terms of embodied energy, and pricing efficiency of a certain configuration (to be possibly compared with alternative configurations).

The algorithm is developed and tested in the Rhino & Grasshopper environment using the BMC algorithm developed by Shervin Azadi. The tool will read the voxels' labels of each cube's vertices (a collection of 8 neighboring voxels) to load and choose the corresponding tile from a predefined tileset. The project's focus is the design of such a tileset, its engineering consistency, and architectural coherence. Figure 2 is the use case diagram of the thesis. As an input for the algorithm, importing a colored voxelated mass (later referred to and specified as a configuration) produces an output of a modular architectural assembly. This workflow envisages becoming a procedural component of a Configure, Price, & Quote (CPQ) system for the industrialization of mass customizable housing.

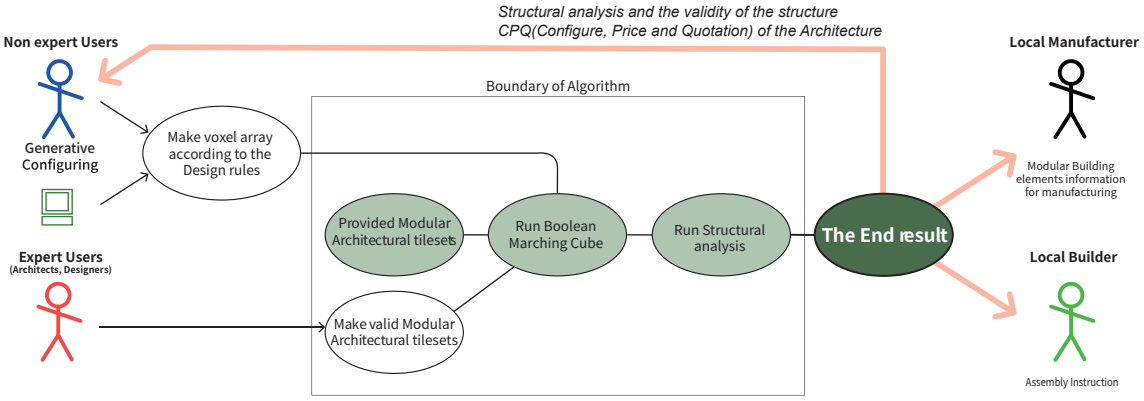


Fig.2: Diagram of Research Objective as Use case diagram (Author)

1.6 Scope

Scope- The Go_Design framework, a modular generative design framework for mass customization and optimization in architectural design [1], articulates the framework to three meta procedures, 1) space-planning, 2) configuring, and 3) shaping. Planning' is the procedure involving multi-value, multi-actor, and multi-criteria complexities and aims to reach a collective design goal. 'Configuring' is a procedure focused on generating a configuration of spaces from the specified criteria and relations in the planning step. The 'Shaping' stage, which focuses on the topological design, involves multi-dimensional and multi-value design complexities and determines the aesthetic styling of design.

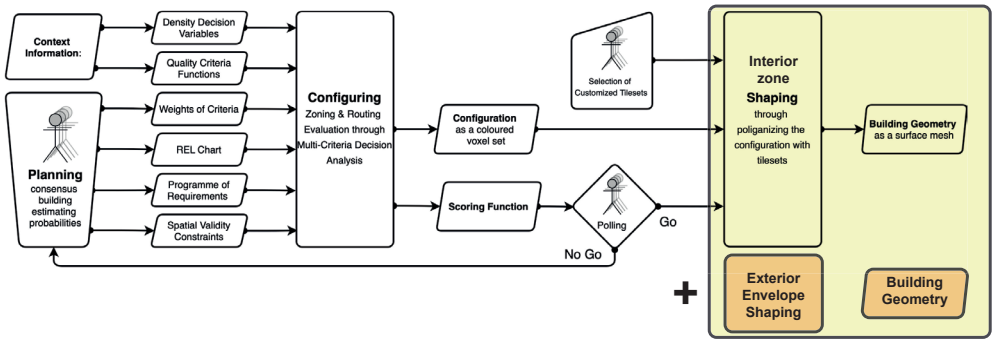


Fig.3: The scope of the thesis in the GO-Design framework [1]

The thesis focuses on developing a method for the 'Shaping' stage of the GO_design framework in developing a system for polygonizing a modular architecture, described as colored modular configurations from the configuring stage. The proposed framework of the thesis maintains the GO_design framework but works on another layer of colored modular configuration, which defines the architecture's exterior envelope, apart from the zoning configuration. This approach is also related to the Open building concept [12] of physically separating and layering architecture components by longer and shorter life spans for easier adjustable over time.

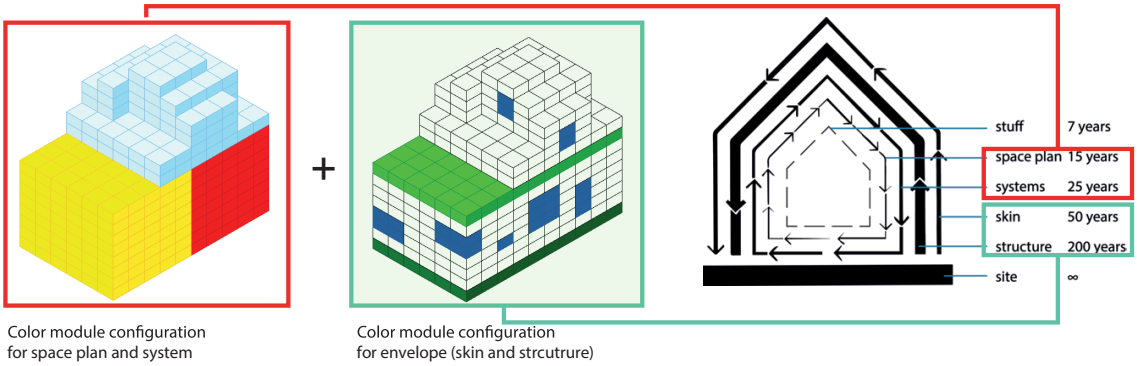


Fig.4: The scope of the thesis in the Open building concept (Author)

The scope of the thesis is not configuring colored modular configurations(voxel array). The research aims not to present a ready-made system for the market but to determine the direction of further investigation and acknowledge that there are multiple solutions to the problem.

1.7 Research methodology

The research methodology of the thesis is 'research by design.'

Background Research- The literature review divides into three parts. The first part discusses the Modern method of construction, background research on previous modular construction practices, categories, benefits, and case studies of modular construction to search and conclude on the fittest element type of MC for the thesis. The second part discusses the modular participatory design of toys, gaming, and architecture, providing participatory creativeness. The research includes the toy industry, LEGO and Mobaco, the game industry, and Townscaper, and in the architectural context, the case study from the Savov from ETH Zurich. The third part is the research on algorithms, discussing the two procedural generation algorithms, the Boolean Marching Cubes algorithm and the Wave Function Collapse algorithm. The two algorithms will be compared and selected for the project. **Design Methodology - I**n the design development stage, First, set the preset of the algorithm and module components: the terminology, the Module dimension, and the 26 unique cube tilesets. Second, the surface and architectural tilesets and examples of each tileset. Third, design the polygonization BMC algorithm and discuss the input voxel rulebook. **Result-** In the result chapter, validate the result from the design phase for accuracy, and the possibility of structural validity made using a grasshopper plugin, Karamba3D. **Conclusion/reflection** - contains the concluded results, future works, limitations, and project discussion.

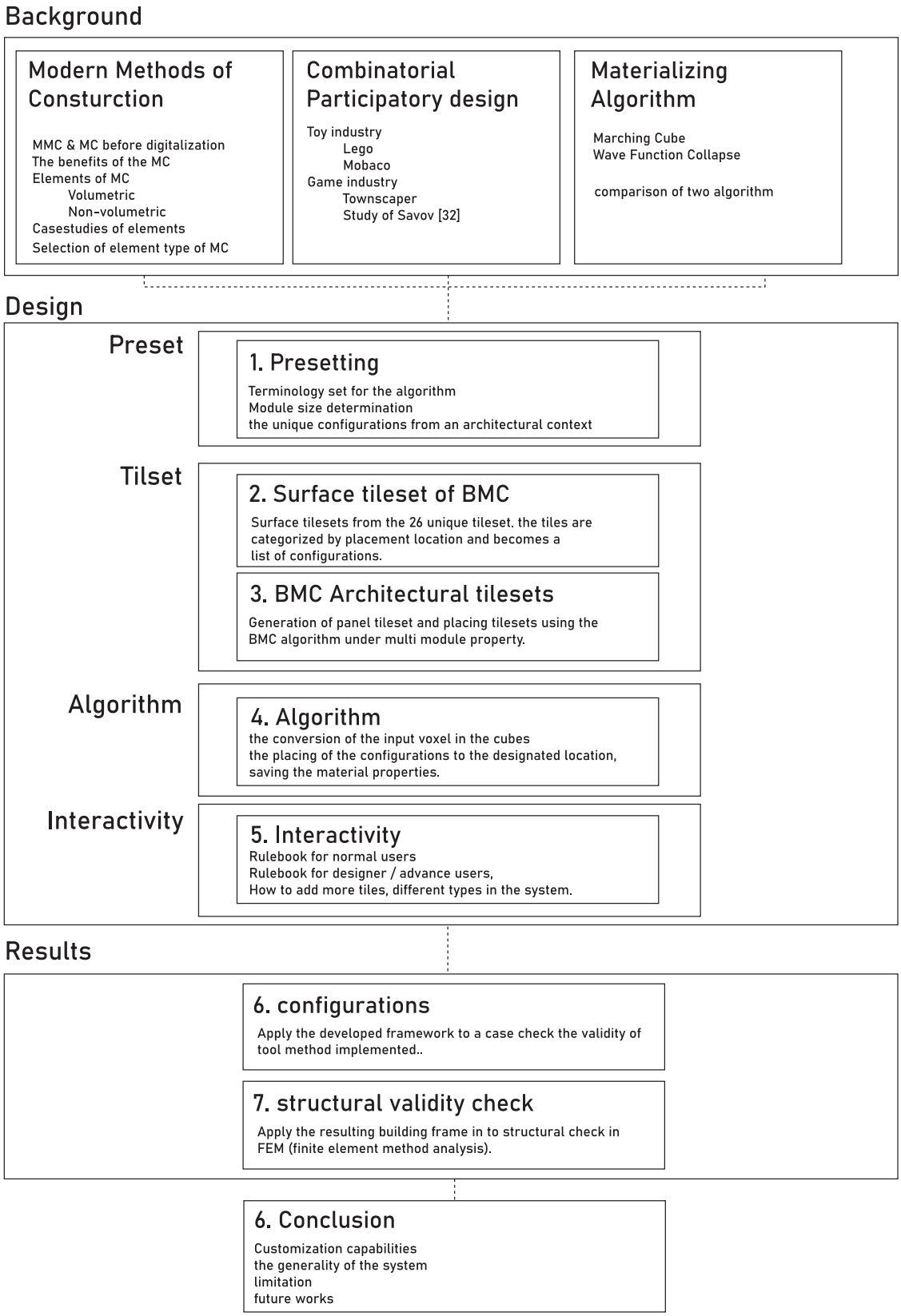


Fig.5: The overview diagram of the research (Author)

CHAPTER 2

2. BACKGROUND RESEARCH

2.1 Modern Methods of Construction

2.1.1 Modern Methods of Construction

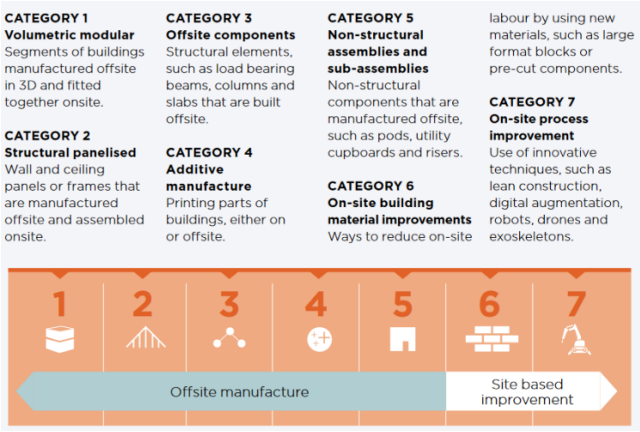


Fig.6: Seven categories of Modern Method of Construction [4]

The MMC (Modern Methods of Construction) is a broad term to describe a wide range of construction processes aiming to produce more sustainable housing for better quality, cost, and less time. In many cases, the term is parental for modular and offsite construction. In the Savills [4] report, the UK Government published a definition of the framework and the seven categories for defining MMC: and in the seven categories the five: Volumetric modular, Structural panelized, Offsite components, additive manufacture, and Non-structural assemblies and sub-assemblies are the primary methods of offsite modular construction. According to the technical report of Oliveria[13], The UK government has identified MMC as a critical vision for meeting the UK housing needs and promotes its application to encourage the adoption of modularity in the construction sector.

2.1.2 Open Building concept

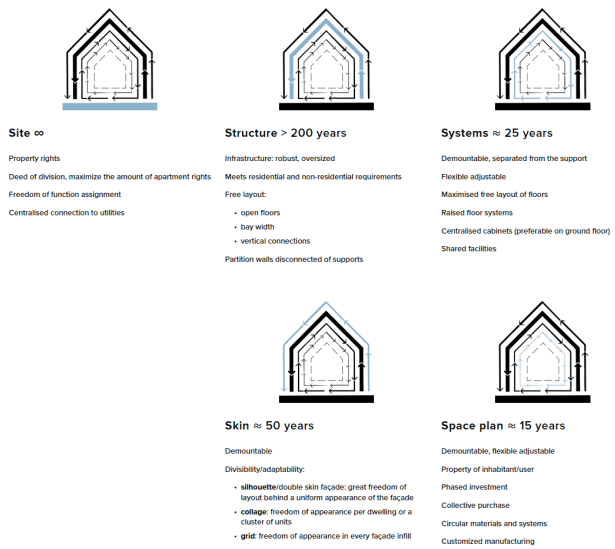


Fig.7: Shearing layers of change from Openbuilding [12]

The Open building concept is an idea developed by N.John Habraken in the 60's. the main objective being to physically separate the elements with different life cycles, and increase flexibility and adapability of the building. The principle is often described using the Shearing layers of change designed by Stewart Brands's giving example of layers of different life span of the architectural elements [12].

2.1.3 MC before digitalization

Throughout the decades, the attempt to industrialize and modularize construction has existed in the belief that the new construction approach can bring rapid urban housing development, mass customization, and more affordable housing. Such attempts of MC existed even before the digitalization of parts and assisted computation. Some of the attempts succeeded in the market and enhanced construction productivity. However, the attempts vanished due to the financing issues outside the company's handle, such as the great depression and the war that caused government policy changes. Although the case projects no longer exist in the market, the attempts build the foundation of the current MC industry.

Fig.8: Images of Sear's Modern Home Catalog [4]

Sears Modern Home Catalog - In 1908, Sears, Roebuck, and Co. introduced a Sears Modern Home Catalog in the US. The Sears Modern Home catalog presented more than four hundred houses in various styles. The instruction and parts came in prefabricated pieces, ready to be assembled. Sears also provided a service allowing buyers to submit their designs to present and share plans offering creative control of the parts without the quality reduction. Sears found massive success, and 75,000 houses were built in this manner for over five decades, reducing housing costs to less than two-thirds of conventionally built homes [14]. In 1941, as the war broke out, residential construction was only permitted for workers in the defense department, and Sears was troubled in sourcing the lumber they needed. In 1942 the US government called for the need for lumber supply and gained control of all lumber mill sales and deliveries to ensure the supply of lumber for military purposes [14].

Fig.9: Images of Oak ridge project [4]

Oak ridge project - In 1943, Skidmore, Owings & Merrill (SOM) managed to construct three thousand prefabricated houses for members of the Manhattan Project in the secret location now known as Oak Ridge using MMC. The houses were prefabricated in cement as planar wall types and plugged in varying configurations. The windows and casework were often manufactured off-site and then installed on-site, and assemblies such as machine rooms and HVAC systems were commonly built off-site and craned into place. Manufacturing One house In every 30 minutes and, on average, built around 17 homes per day. [14], [16]. After the war ended, the government withdrew its funds and its expansion. According to a report from AIA(American Institute of Architecture), The increasing recognition of the modular construction industry benefits contributes to increasing projects using the MC method. Often the MC involves the off-site prefabrication of manufacturing architectural elements contributing to improved quality, productivity, safety, schedule, cost/value, and sustainability [3].

2.1.4 The benefits of the MC

Quality - Under a monitored manufacturing setting, precise fabrication tools and automated processes allow high-quality control and consistency. Contrary to traditional construction, off-site construction enables the fabrication of wall components to achieve increased precision. As a result, the building envelopes are much tighter, with fewer air and water leaks. [3]

Productivity- Labor productivity can increase working in a plant with more negligible effect by weather and equipped with necessary tools and machinery. Working under a roof in a factory enables workers to perform in the best conditions to achieve a high level of productivity. [3] Also, MMC requires fewer operatives on-site, and it's possible to significantly reduce the level of skill needed. The use of low-skilled personnel is possible if the design is sufficient enough. [6]

Safety- The most praised benefit of MMC is the speed of construction. According to the Modular Building Institute, the construction schedule can be saved dramatically by 30 to 50 percent. This time saving brings significant benefits compared to the traditional wet construction requiring concrete curing. The more work can be completed off-site, the greater the savings from the increased time saved on-site.[3]

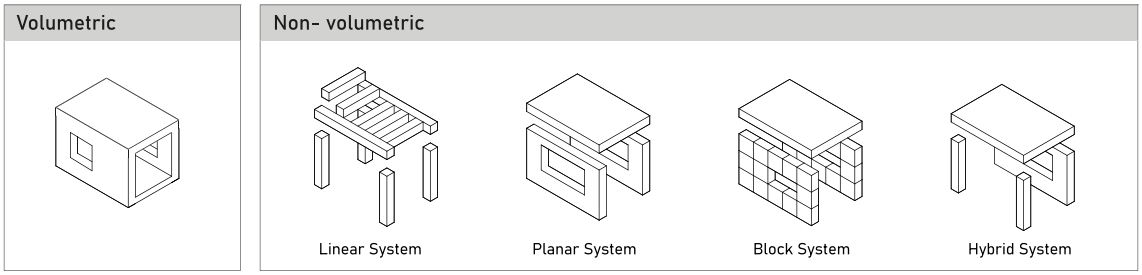
Schedule- The most praised benefit of MMC is the speed of construction. According to the Modular Building Institute, the construction schedule can be saved dramatically by 30 to 50 percent. This time saving brings significant benefits compared to the traditional wet construction requiring the curing of concrete. The more work can be completed off-site, the greater the savings from the increased time saved on-site.[3]

Cost and Value- The AIA report says that although modular construction can be more cost-efficient than on-site construction, it will not automatically reduce the project cost. However, The prices are more predictable than traditional construction methods. In the document of Construction 2025 by the HM Government, MMC products resulted in a 33% reduction in the initial cost of construction and the whole-life cost of assets. [6] if MMC is implemented correctly, it will prove to be a more cost-efficient way to create value in the long term.

Sustainability- The off-site production of building components allows for optimal material use control, resulting in reduced material input and waste than traditional on-site construction. Surplus material and fall-off can be captured and recycled back into the inventory for use on other projects.

2.1.5 Building elements of MC

In general, the Offsite construction divides into Volumetric and Non-volumetric construction methods. The thesis of T.M de Haas [17] defines the four different elements of modular prefabrication (linear, planar, and hybrid). The primary criteria for categorizing the elements are the size, shape, and assembly method. The thesis adds one more element type to Haas's definition: the block type element.



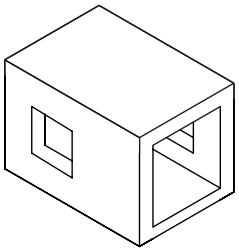


Fig.10: Volumetric elements (Author)

Volumetric elements

Volumetric systems are the most factory-based production elements. It involves off-site prefabricating individual three-dimensional units representing a part of an entire room, which then is connected on-site to form a single building. These modules can be finished in the factory to include all fixtures and fittings, requiring minimal on-site installation work. Once arriving at the site, the Volumetric element assembly method enables the fastest assembly and requires minimal time and work.

Another advantage is that the elements are fully finished, and operational as most of the work is done in the controlled environment factory. Often, having modules fitted with stairs increases productivity and excursion, as there is no need for ladders, scaffolding, and lifts to move from one level to another [3].

However, The size of spatial elements is often limited by the transportation size on local roads and compared to non-volumetric systems, and the volumetric elements cover more transportation volume as they are three-dimensional. Another disadvantage is that a manufacturing facility requires a more significant investment. For this reason, The elements are less suited for small and medium construction and more optimal for larger housing complex projects and multipurpose highrises buildings.

Non-volumetric elements

The non-volumetric system connects the off-site prefabrication of building elements on site. This method uses a standardized process to deliver bespoke solutions, making build-to-order manufacturing in plants possible and enabling combinatorial creativity by using standardized components. Components can be transported more compactly than volumetric units, potentially reducing transportation costs. But the system requires additional assembly and sealing work on-site [6]. The non-volumetric system is categorized by the elements used, linear, planar, block, and Hybrid types.

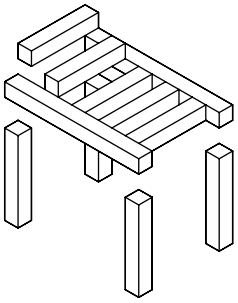


Fig.11: Linear elements (Author)

Linear elements are the closest elements to the conventional construction elements, which are already widely adopted in the construction industry, as most structural beams are prefabricated as linear elements. However, It isn't easy to find them fully serviced as MC and finished as single units in the market, as most cases are in the form of concrete, steel, and wooden beams delivered on-site. The Linear elements are often easy to connect, but additional processes, parts, and finishings are required. In general, this linear system requires the most work on-site as the elements stay as pure material on-site in terms of customization.

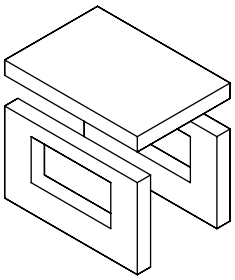


Fig.12: Planar elements (Author)

Planar elements are two-dimensional panel elements. These panel elements perform better on design flexibility and optimized logistics than the volumetric system. Therefore they are more adapted in small or medium-sized architecture, such as single-family housing projects, than more giant-scale constructions. Planar elements with a complex level of MMC can equip building services such as HVAC, plumbing, and electrical systems in the wall element before the assembly. [6] When done correctly, planar elements provide easy connection of elements and do not require more space for construction than volumetric construction as they can be transported as flat-pack.

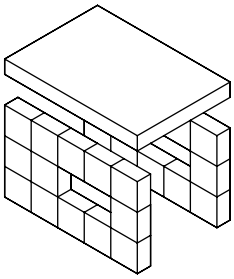


Fig.13: Block elements (Author)

Block elements are similar to stacking toys such as LEGO modules; these blocks are usually lightweight and filled with insulation to provide sufficient thermal, acoustic, and energy efficiencies and offer users freedom of design variation. Yet, the block system does not provide the level of complexity of MMC as other systems offer. In general, the size of each element remains smaller sizes that one person can lift and are easy to connect without skilled labor, making the block system more suitable for small housing projects.

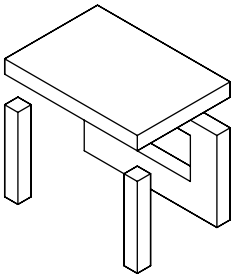


Fig.14: Hybrid elements (Author)

Hybrid elements are mixed uses of different systems, and most modular construction is in hybrid form to offer the benefits of each approach to the market. In most cases, prefabricated columns, walls, and floor slabs are widely used as elements. Also, developing computation and digitalization enables the mixing of components more accessible. According to Bertram et al. (2019), mixing such systems can be cheaper (3%) than just using one element.

2.1.6 Case Study of Building elements of MC

Volumetric System Case Study - Finch building

Fig.15: Images of Finch buildings representing volumetric element type of MC [18]

Finch building [18] is an example of using volumetric elements as a modular unit. In Finch buildings, a sustainable approach is delivered as the modular volumetric units are produced from mainly sustainable wood material (CLT) consisting of a fixed number of ingredients. The module size is set to specific dimensions prefabricated from partnering factories. By standardizing the prefabricated units, waste and energy consumption are reduced, The quality is maintained under a controlled environment, and each volumetric unit complies with the dutch building codes.

Compared to the average construction period in the Netherlands, between 7 to 10 years, the Finch buildings take an average construction time of seven months from delivery of the modules, as the products are prefabricated in a plant and assembled on-site, significantly reducing the construction time. Another aspect the Finch building emphasizes is its re-locate ability. The modules can be transported to other places as a finished product as a unit.

The Finch building offers design, coordinating their product's permit, which provides a rather bespoke approach using standardized units. The reconfigurability of spaces is relatively lower than other companies as the dimensions of units determine the spaces and are supported by a load-bearing wall of a fixed size.

Linear System Case Study - Bone structure:

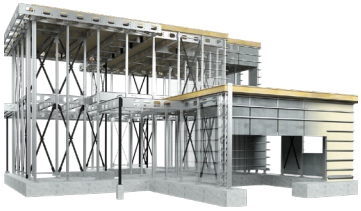


Fig.16: Images Bone structure representing linear elements of MC [19]

The 'Bone Structure[19] is a company practicing the linear system of MMC. 'The Bone structure' offers a patented light steel construction technology that combines the advantages of post and beam structure.

Their linear elements are produced with galvanized steel as the primary material and designed on 3d software, outsourced to manufacturers, and laser-cut from factories. The construction is pre-engineered from the design phase and requires no highly skilled labor. The firm's pre-cut openings in elements of the structure are ready to accommodate electricity, ventilation, and heating. The company also emphasizes the environmental sustainability of the products. The steel elements are reusable and can reduce the integrated process of precisely calculating its parts, waste reduction, removal, rent of containers, and labor.

The 'Bone structure' specializes in medium-scale architectures such as residential and light commercial buildings, which often require no interior load-bearing walls and offer adaptable and reconfigurable space to customers. As the linear elements are less volumetric than the volumetric and planar systems, minimizing transportation size and space reconfiguring is possible.

Planar System Case Study - gologic.us:

Fig.17: Images of Go_logic houses and planar element installation [20]

Go Logic[20] houses are examples of an MMC using planar elements as their construction unit. Using wooden '2×8 stud framing', their planar elements provide structural support for the building and accommodate insulation. The computer model of each panel design allows generating a specific plan for each panel, detailed down to the individual stud.

The panelized exterior wall model is prefabricated under factory conditions, completed with building wrap, windows, and exterior insulation installed and tested. Then walls are delivered to the building site, craned into place on a foundation constructed in advance. The fabricating of building panels for a single house takes approximately three to four weeks. Their design process and manufacturing method decrease the construction time.

The firm offers some level of customization, from a catalog offer the basic types, and as a service provides unique customization design. Customers can select from various kinds of exterior cladding, such as fiber-cement clapboards, Eastern white cedar shingles, charred cedar boards, and corrugated weathering steel. The firm also offers a range of options for other home elements, from window trim and porch decking to interior flooring, appliances, lighting, and hardware.

Block System Case Study - Gablok:

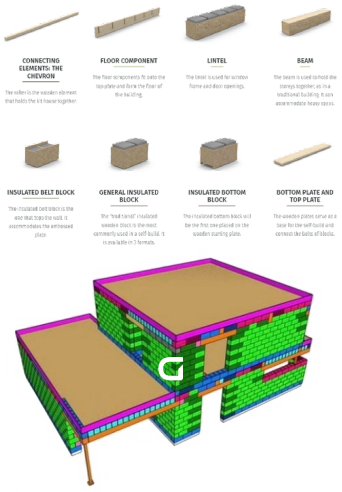


Fig.18: Images of Gablock digital modeling and elements [21]

Gablok, founded in 2019 in Belgium, is an example of a block system in MMC. Gablok proposes a concept that assembling a building can be done using block elements that are lightweight and right-sized that one person can lift and stack like a lego block. Their blocks are made in three lengths: 30, 60, and 90cm. The height and depth of the blocks are fixed to 30cm, and the weight of a block-sized 60cm is 7.5kg. Made of OSB and 264mm thick graphite EPS insulation components, The insulation is free to slide to form the protrusion necessary for interlocking. Their block elements have low complexity but offer limitless design freedom [21].

After installing the blocks, the exterior finishes can be selected from options of plaster, bricks, facade cladding, etc., since the installation system is fast and requires no drying time, reducing construction time. However, the

transportation volume is the same or more than conventional construction since the blocks are prefabricated as volumetric block elements, and reconfiguring after completion is just as difficult as traditional architectures.

Hybrid System Case Study - Sustainer Homes:

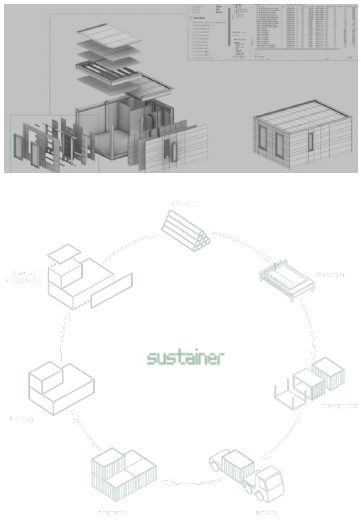


Fig.19: Digital building a system of Sustainer Homes [22]

The Sustainer Home [22] is an example of a Hybrid element system. The Sustainer home provides a digital building system and does not manufacture or supply parts or modules. Still, the firm's product is the information set of highly detailed 3D models via a digital building system. The self-developed software can automatically load a database of geometric elements into a new design based on their design. With this, the company and customers can achieve a cost reduction from the digital twin and get precise data on the use of the material.

Sustainer Homes aim to achieve seamless integration from design to realization in partnerships with builders. In Sustainer Home's digital product, every screw and notch is visible in the model of digital development. This makes the model the foundation of the most critical business processes in the assembly, including purchasing, calculation, planning, logistics, quality control, and aftercare.

Since the product sold is only the highly detailed information, any local manufacturer with a suitable machine can produce the provided information, and transportation can be linked with the closest manufacturers. Reconfigurability can be achieved easily since all parts are transparently given as detailed 3d data and information are given.

2.2 Participatory design of modular elements.

The main question that the thesis address is the participatory design of modular architectural elements. The modularity, the combinatory creativeness, and the interaction and connection between modules are the three traits of the participatory modular design discussed in each case study.

Gamification and serious gaming refer to computer software designed for collective decision-making or participatory design not just to play but also has other severe aspects like education, decision-making, and problem-solving. Following the idea, video games and toys are interactive mediums where players or children can engage with the production of form and systems thinking. In toys and computer games, the participatory design of modules has been studied by researchers worldwide and achieved in smaller product sizes and digital forms.

The case studies of such modular participatory design involve,

the toy industry: LEGO, and MOBACO,

the game industry: townscaper,

the architectural industry: research of Savov [32], GO-design framework [1]

2.2.1 Participatory design of modules in the toy industry

Case Study of LEGO- LEGO, a toy brick company, established in 1932 in Denmark, is commonly used to describe the participatory design. LEGO bricks clearly illustrate features of the participatory design of modules.

The article by Linus “Getting started with advanced LEGO” [25] describes The modularity of LEGO, the essential piece of LEGO is the LDU part, The name originated in the LDraw(LEGO CAD) community, representing the smallest LEGO unit. The size of a single LDU piece is 4.8mm in height and 8mm in width and length, and the Combinations of LDU parts create all lego pieces no matter their variations in size and shape.

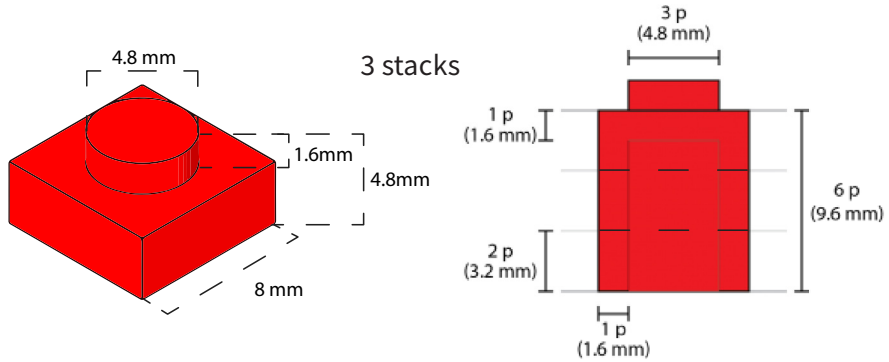


Fig.20: LDU smallest part of LEGO, the smallest module size of LEGO [23][24]

Another essential aspect of LEGO is combinatorial creativity. An associate professor at the University of Copenhagen, Soren Eilers,[26] has written a computer program that systematically generates and counts all configurations with six two-by-four studded LEGO bricks.

The result was 915,103,765 combinations. Also, all pieces of all varieties constitute a universal system. Despite variation in the design and the purposes of individual pieces over the years, each piece remains compatible with existing pieces. Lego bricks from 1958 still interlock with those made in the current time, and Lego sets for young children are consistent with those made for teenagers.

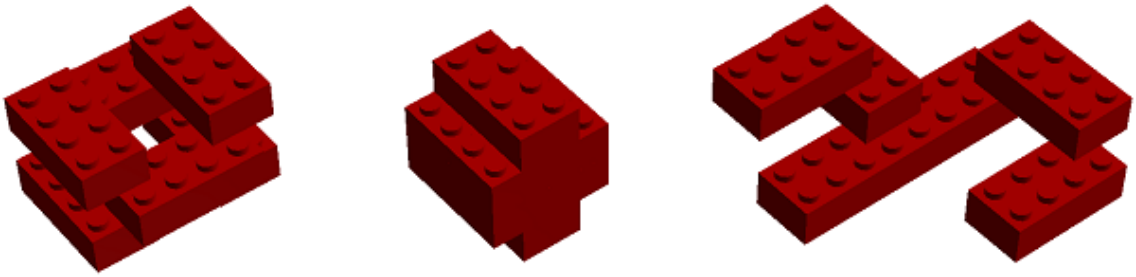


Fig.21: six-two by four studded LEGO bricks generate 915,103,765 combinations [27]

Case Study of Mobaco - Mobaco was a Dutch construction toy made of wood and cardboard from the 1930s, containing all the features of the participatory design of modules. The building system included most of the architectural elements that resembled standard architecture, and some descriptions of its parts could be found on the website of the museum of kinderwereld [28].

The Mobaco consisted of brown and green cardboard base plates of at least 15 by 15 centimeters and 1 centimeter thick, in which holes of about 1 centimeter punched at a distance of about 7 centimeters. There were black hardened cardboard floor plates with holes in the exact dimensions of the floor's base plate. Each floor is 10cm high, giving it a scale of approximately 1:30. Wooden posts with grooves on all sides could be one, two, or three stories high. The Wall, Door, and window elements are made of pressed cardboard and available in various colors fitted into the grooves. There were also roof elements of Mobaco, always in red and intended to form pitched roofs, with hooks punched out at the ridge to interlock.

The Mobaco system came with an instruction book, but the combinations that could be generated were endless, as the figure shows. The facade systems could slide in different variations, and the roof pitched tiles could interlock with other tops to extend the wall plates.

Fig.22: Mobaco toy assemblies from the museum of kinderwereld [28]

2.2.2 Participatory design of modules in the Game Industry

The Townscaper is an indie city builder video game developed by Oskar Stålberg, featuring low poly graphics and a simple, minimalistic user interface. In the official description of the Townscaper [29], Stalberg describes the Townscaper as a toy than a game. The game lets users construct an island town by placing and removing colored blocks on an ocean.

The game modules are a block that can be constructed and deconstructed into different houses with clicks. Each click can place these architecture blocks vertically and horizontally. As the module connections are made using an algorithm, the link of each module is seamlessly modeled in 3D. The game is based on mixing two algorithms, the Marching Cubes and the Wave function collapse algorithm, addressed later in the thesis paper. The combination the game can create is endless as all required for placing modules are clicks. By discretizing a grid space into a series of points, players can create a polygonal mesh using points within that 3D space. Some rules dictate the block's appearance, some as spires and others as balconies. This method of rule-based decoration allows arches, gardens, and stairways to be created without specific user instruction.

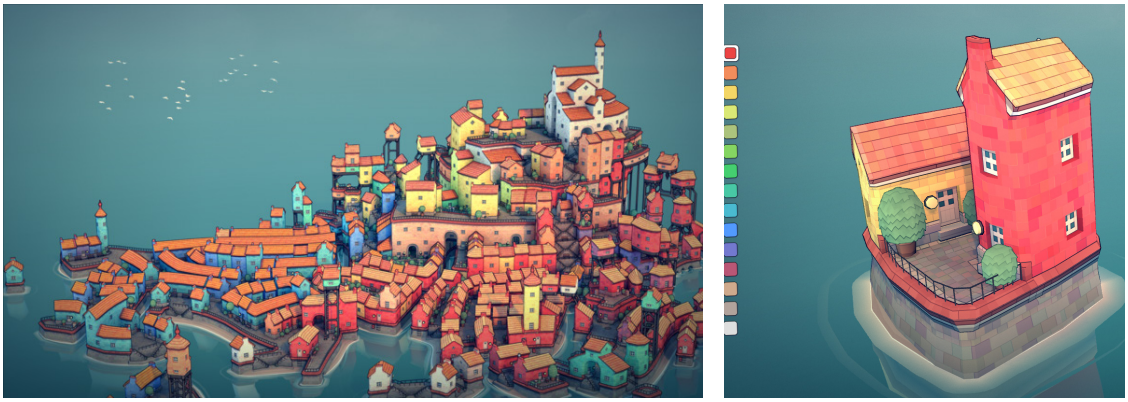


Fig.23: Townscaper visualization from the official website [29]

2.2.3 Participatory design of modules in the Architecture Industry

In the research of 'Encoding Architectural Design as Iso-surface Tileset for Participatory sculpting of massing Models' conducted by Savov [32] in ETH Zurich developed a participatory computational design system. Here Savov presented the concept of 'assisted sculpting,' which is 'the assisting of non-experts in exploring design options using a combination of iso-surfacing and constraint-solving tools to make design easy for non-experts and could bring them into a creative flow. The device is made with the BMC algorithm implemented in Rhino/grasshopper environment and tested with massing models and schematic space allocation.

The Modules of the program are the input of a 3d array of voxels with tile options that designers or architects can themselves model. The program provides a tileset editing mode offering creative challenges for architects making full use of their skills and knowledge. Inside, the tool uses 15 unique tiles that can form an isosurface covering. Using the tileset, if the tileset is valid in connection, the combinations can be created in seamless and countless combinations.

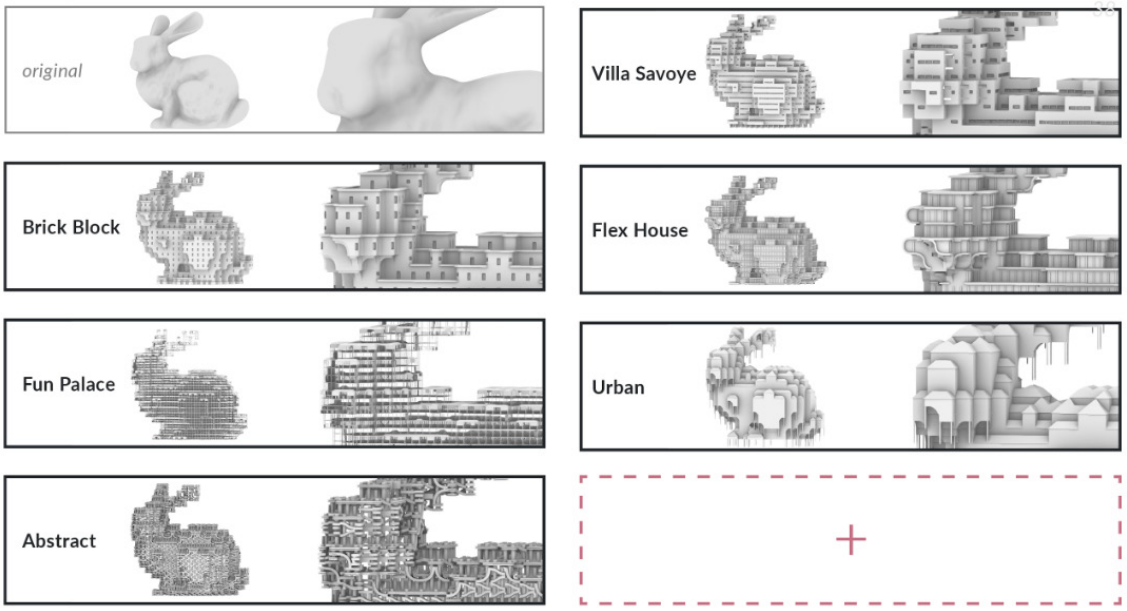


Fig.24: The tilesets created by workshop participants from the research of Savov. [32]

Savov [32] also suggests in the paper that various options are available to stage the interactions that generate the input field (voxel). It can be done via a point-and-click as a participatory design. But it also can be based on different logic such as structural stress, solar analysis, cost estimates, or any other performance-driven analysis. Savov also suggests that the program should be developed as a web version to allow more people to explore the collaborative qualities of the design system and involve as many non-experts and reach a broader crowd as possible.

2.3 Procedural content generation Algorithms

In computing, procedural content generation(PCG) creates data algorithmically instead of manually, typically through a combination of human-generated assets and algorithms coupled with computer-generated randomness and processing power. The two algorithms are prominent in PCG for generating isosurface using modeled tileset, the BMC(Boolean Marching Cube and WFC (Wave Function Collapse). In this chapter, the two algorithms are researched to compare.

2.3.1 Marching Cube algorithm.

Lorensen and Cline invented the Marching Cube algorithm in 1987 [8]. The original use is to extract a polygonal mesh of an isosurface from a three-dimensional discrete scalar field, the main concern of medical visualizations such as CT or MRI scan data into three or two-dimensional images. (the description and diagrams of the two-dimension of BMS (boolean marching square algorithm) are in appendix 1. However, The application of the algorithm expands from the medical field and is also widely used in computer graphics.

The algorithm proceeds through the scalar field. Then, the imaginary cubes form the field taking eight neighbor locations from its vertices. When passing the specific cube, the cube is read as 8-bit integer indices, fusing the preset polygons into the specific cube index.

the possible eight vertice configurations are 256 possible polygon configurations. (two to the power of eight) and the configurations can be represented as an 8-bit integer. (in Boolean marching cube 1 and 0, If the value read: ‘1’ if the value does not read: ‘0’) After checking all eight vertices or scalars, the final 8-bit integer becomes the index of cube indices. Finally, in the appropriate position of the cubes along with the configuring number, fuse the preset polygons (Figure 25).

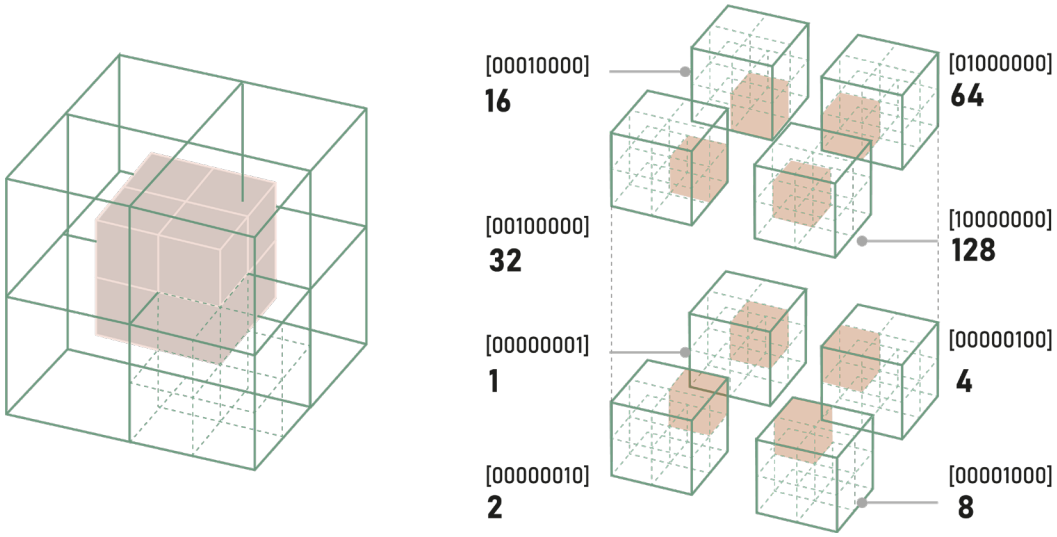


Fig.25: BMC of a single voxel module represented as eight cubes with unique configuration id. (Author)

2.3.2 Wave Function Collapse

The Wave Function Collapse algorithm was developed by Maxim Gumin[31][35] as a texture synthesis method based on simple configuration or sample images. It is a constraint-based procedural algorithm inspired and named after the concept of wave function collapse from quantum physics. [35] The WFC algorithm defines each cell in a grid as a slot. Whereas the Marching cube reads from vertices of imaginary cubes, the WFC algorithm reads the lists of possible neighbors. Six neighbors in a three-dimensional system and four neighbors in a two-dimensional system. Each module inside the slots contains information about the 3D model and the constraints module's neighbors, allowing neighbors to be next to them or not. (as illustrated in Figure 26) [35]

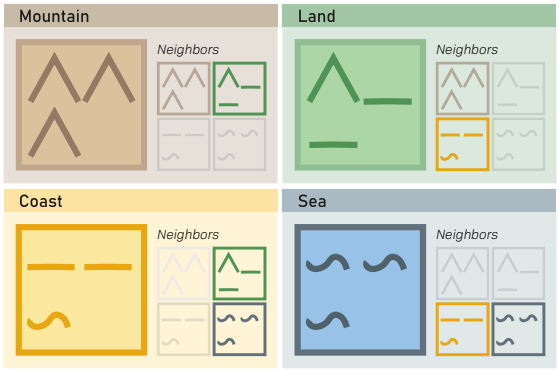


Fig.26: Example diagram of WFC neighbor contraints adapted from [31]

In the unobserved state, each slot can be filled by every module possible. When the algorithm runs and the slots are observed, as one slot of the grid collapses down to a single possible module, the neighboring slot will also be restricted in possible modules because of the adjacent collapsed slot. The algorithm continues to proceed to the following slots, containing them in possible modules until the process is over. In computer graphics, the algorithm mostly generates maps for gaming as the process can create an endless and seamless connection of tiles while keeping the constraints inside each module.

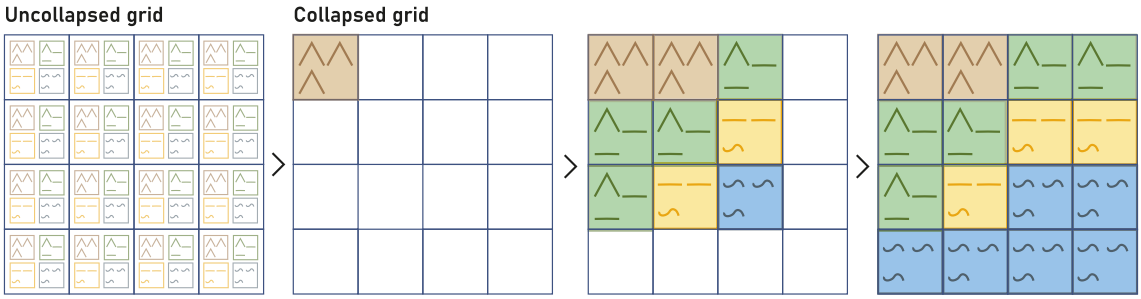


Fig.27: Example diagram of WFC algorithm adapted from [31]

2.4 Conclusion of background research

The background research aimed to overview the current status of the modular construction industry, focusing on modularization using participatory modular design methods using computational algorithms. To research this, devised the following topics: Which type of modular element is the most suitable for the modular participatory? And consider which PCG algorithm for the modular participatory design?

Which type of modular element is the most suitable for the modular participatory design?

In choosing a modular element, multiple criteria need to be considered, such as architecture's size, types, function, reconfigurability, the transportation of the elements, etc. There is no single universal element system for all kinds of architectures. Specifying the architecture's type and other criteria for choosing a modular component is essential. The main architecture types defined in the thesis are houses and small commercial buildings that can be structurally stable from only the exterior framing. The thesis provides a methodology for creating an exterior architectural frame using an input voxel array. The report on Delivery Platforms of Government assets [6] provides the (Figure 28) mapping of MMC initiatives from least to most efficient. The highest efficiency increase is in Manufactured elements, in which it is possible to bring bespoke solutions from the standard process.

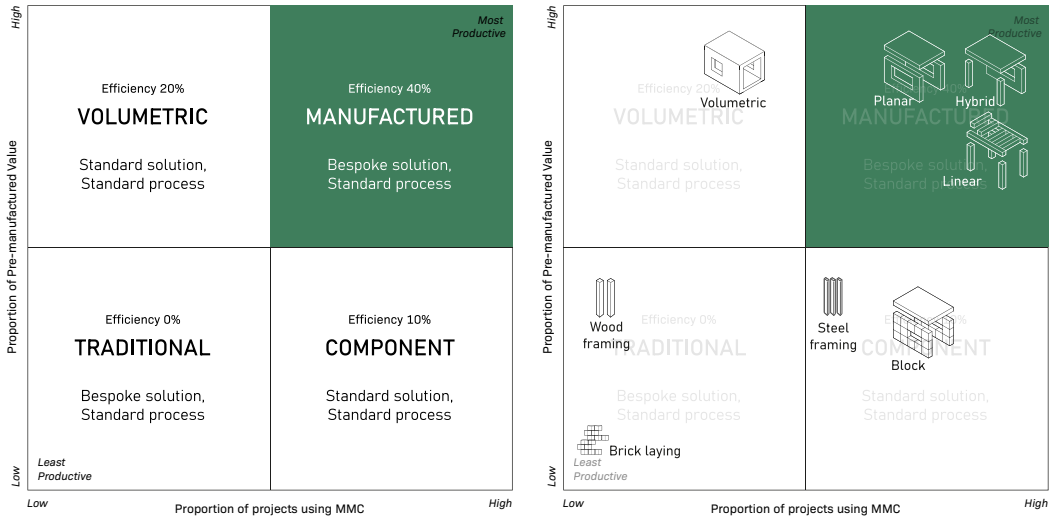


Fig.28: Mapping elements of the MMC initiatives from least to most efficient adapted from [6].

In Figure 26, a conclusion graph is illustrated Based on the research. The Hybrid, planar and Linear elements, offer bespoke solutions, and the manufacturing process is standardized, showing the most potent to enhance efficiency. Since the parts are already widely used in the market, the components require a small transportation volume. Developments of the elements provide the most benefit as the industry does not require significant investment in changing already existing facilities. The thesis uses the Hybrid of Planar and Linear elements of MMC as the research element since the component is closely linked to the conventional construction method and is the most basic form of architectural elements.

Which PCG algorithm for the modular participatory design?

Among the two PCG algorithms listed, the Marching cubes and Wave Function Collapse, a comparison of the two algorithms is considered. In the report, Savov documents and compare the two algorithms. The table is modified based on the context of the thesis to compare two algorithms and implicit conventional modeling CAD tools.

	Implicit modeling	**Wave Function Collapse**	**Marching cube**
Description (Users)	Users can freely define all the final product's geometric, material, and compositional character-istics.	Users edit rules as text or in a visual coding editor. configures considers constraints and expand in a random generation.	Users can "sculpt" the composition of the final product into a voxel grid while the configurator takes care of the correct selection of tiles.
Description (Experts)	There is no distinction between expert and non-expert. Any CAD or 3D modeling software is in this category.	Experts program rules and model parts, as well as the generative algorithm itself.	Experts can model the possible tiles and define rules for their placement.
Considering connections count	None(limitless)	six sides	eight vertices
Degree of user can express own ideas from encoded content	Very low	High	High
Degree of which experts can encode design ideas	Very low	Medium	High
Challenge experienced by user	High	High	low
Do not let the user start from scratch	NO	NO	YES
Actions are traceable and impact on result	NO	NO	YES

Fig 29. The amount of control offered to the experts and the non-experts adapted from Ref [32].

The table derives that both the BMC and WFC provide some degree of non-experts users expressing their ideas from encoded content, and experts can also encode design ideas. However, the BMC offers less challenge experienced by users. Using a pre-modeled tileset, the BMC gives users a start point without starting from scratch. Most importantly, the user actions are traceable and impact the resulting polygonization (not randomized). However, the results and activities are not traceable or controllable for users using the WFC algorithm (randomized). This traceability makes the BMC more applicable for the participatory design.

Chapter 3

3 Design Methodology

3.1 Algorithm preset

This chapter marks the first step of developing an algorithm and sets the preset required to achieve a participatory design framework under an architectural context.

3.1.1 List of Terminology for BMC

As explained in the BMC chapter (2.3.1) and visualized (Figure 30), the Cube (green empty boxes) contains information on the octant of a voxel (sub-voxel) from the original voxel boxes represented as 8 binary digits. In the BMC Algorithm, the components share similarities in cubical geometry, to avoid ambiguity, it is necessary to define the terminology of the components.

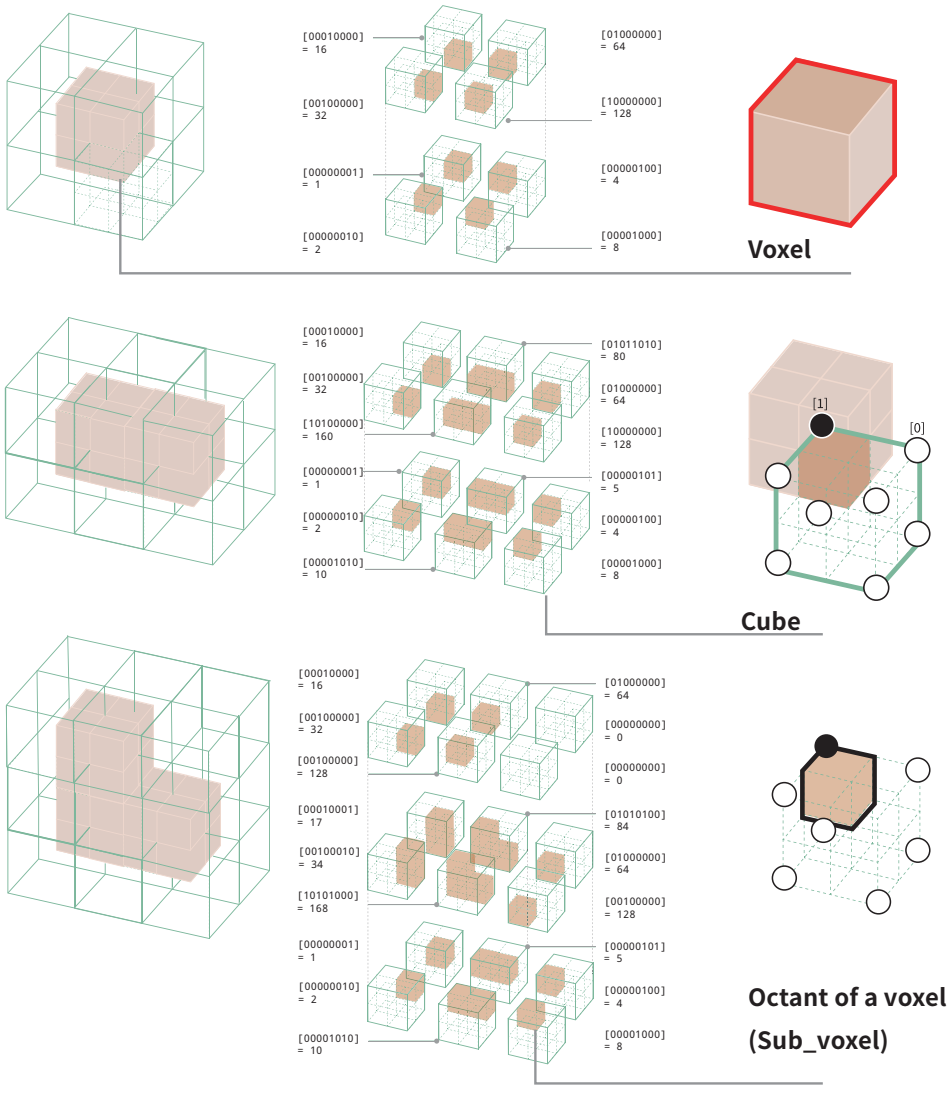


Fig.30: Examples and reference image for terminology for BMC (Author)

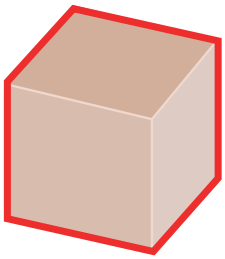


Fig.31: Module diagram (Author)

Voxel

A Voxel is a fundamental component in making a modular architecture and a single unit representation of the initial voxel array from the user input. It is visualized as a hexahedron of a specified dimension and is defined as inside a building. Therefore, the assembly of Voxels represents the connection between the insides of a modular architecture. For the thesis and architectural context, the connection is limited to only surface connection.

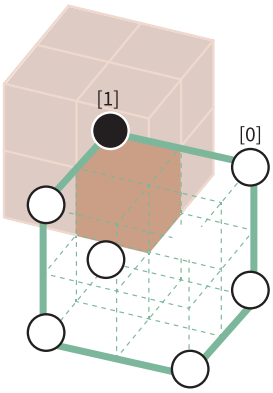


Fig.32: Cube diagram (Author)

Cube

A cube is a unit of the Marching Cube algorithm without any actual substances and is only a representation of 8 codes. In a Boolean Marching Cube algorithm, A cube reads the center points of voxels and checks whether the center point is inside the voxel as a boolean value of 0 and 1. As a cube contains binary information on eight vertices, a cube is translated into eight binary integers and read based on the preset reading sequence. A cube can have eight binary configurations; thus, a single cube can contain information of 2^8 = 256 possible polygons. In the thesis and the architectural tileset, more than two options of 0 and 1 are used in the codification.

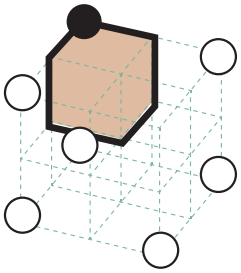


Fig.33: Octant of a voxelSubvoxels diagram (Author)

Octant of a voxel (Sub-voxels)

The Octant of a voxel or Sub-voxel is without any actual substances and only represents one digit from the cube configuration. It is the region representing the overlap of a voxel and a cube and It is a visual expression of the center point of a voxel (also located at the edge point of a cube), containing information about a presence of a voxel inside a cube. An active sub-voxel represents voxel information inside the cube as 1 and if not 0.

3.1.2 Seqence of numbering vertices of cubes

The counting sequence of Sub-voxels(Cube vertices) is essential in the BMC algorithm as the sequence determines the reading order of cube indices and the unique cube ids. (Different programs take the various sequences of coordinate systems in appendix 2). Under Rhino, Grasshopper environment, the reading sequence is based on the reading order of X->Y->Z in F-order,

when given a cube size of (X:1, Y:1, Z:1). The direct translation of the coordinate position in binary to the decimal numbers results in the sequence shown in figure 28 starting from the bottom counts and continuing to the top counts. The sequence works throughout the thesis to generate the unique id of the cube vertices.

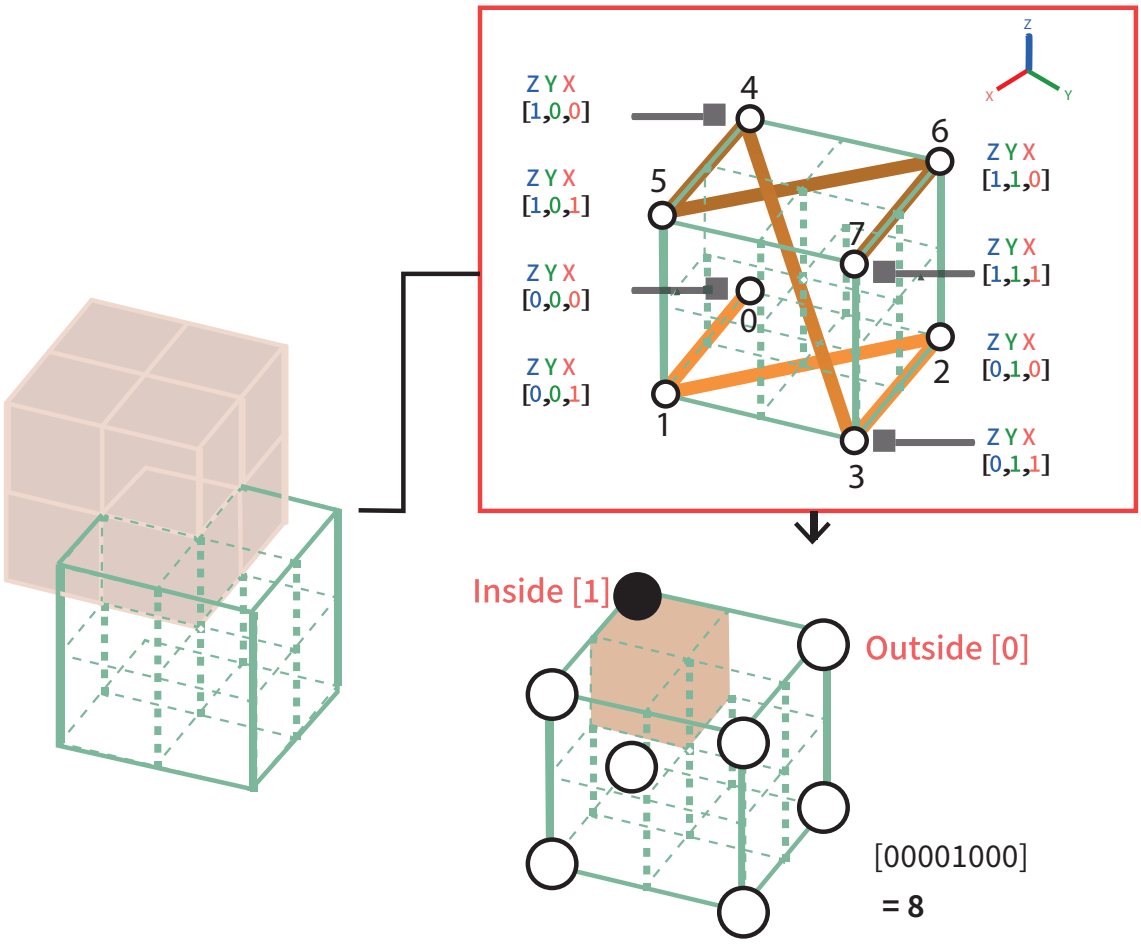


Fig.34: Sequence of numbering vertices for cube configurations (Author)

3.1.3 Establishing 26 unique tilesets

This chapter will develop a methodology for defining the 26 unique tilesets from the 256 configurations within the architectural context. As introduced in the background research of BMC(2.3.1), the algorithm generates a level set (the geometry that separates the 1s from 0s). The thesis defines the level set as the configuration and information of the vertex indices, represented as 8-digit integers. In the Boolean marching cube, the possible configurations are 256 as there are eight vertices to a cube.

Fig.35: the list of 256 configurations of a cube. (Author)

The original unique tileset for the Marching Cube is defined as 15 tilesets [8], considering both the selected and unselected vertices and the horizontal and vertical symmetry of the cube configuration. (visualized in figure 36)

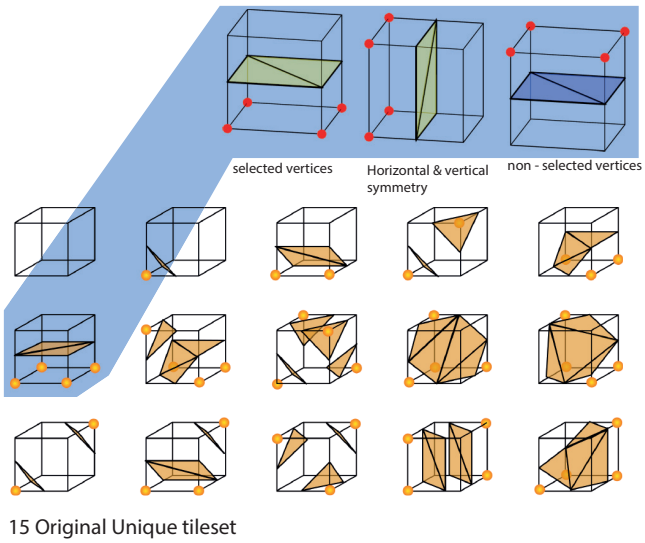


Fig.36: 15 Unique tileset contains selected/unselected and vertical/horizontal symmetries [30]

However, in an architectural context, such vertical symmetry of modules does not create reasonable modular elements. For example, a window frame has different details on the top and bottom for air and water leak protection and structural purposes, and the same applies to the roof and floor modules. Due to these architectural constraints, the thesis will research and practice a different methodology for defining unique tilesets than the original marching cube tileset.

To eliminate from the all 256 configurations of the BMC algorithm, compositions such as ‘Point-to-point’ and ‘edge-to-edge’ connections of the voxel do not create a possible result in the architectural context. Thus, such configurations are eliminated from the list, leaving only the ‘surface-to-surface’ connecting configurations of sub-voxels from the connection of the voxels, resulting in the 126 configurations.

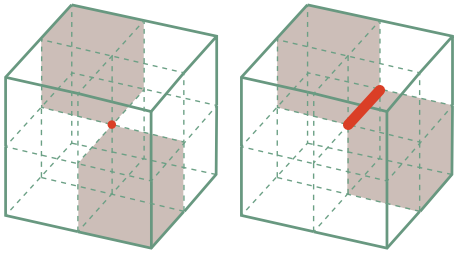


Fig.37: point-to-point and edge-to-edge connection(Author)

Fig.38: 126 surface connecting configurations (Author)

The configurations are grouped by the 90-degree horizontal rotation of the same shapes. The same configured tiles are grouped into 33 tilesets with 31 four directions of horizontal rotations, with the two configurations already horizontally symmetrical.

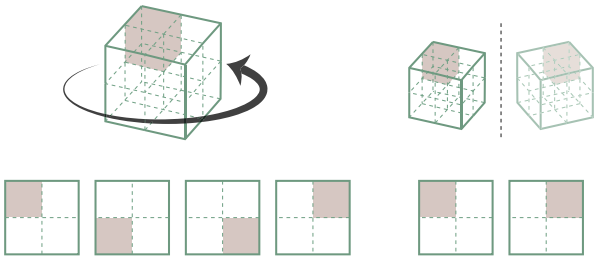


Fig.39: The horizontal rotation and mirroring (Author)

Of the 33 tiles, the 14 tiles mirror the same shapes on a horizontal axis. Such tiles are referred to as “Equivalent up to’’ each other in mathematics. The ‘’Equivalent upto’’ tiles make into seven unique tilesets, each containing information on eight configurations of both rotated and mirrored. This results in the 26 unique tilesets containing information for 126 configurations of an architectural context.

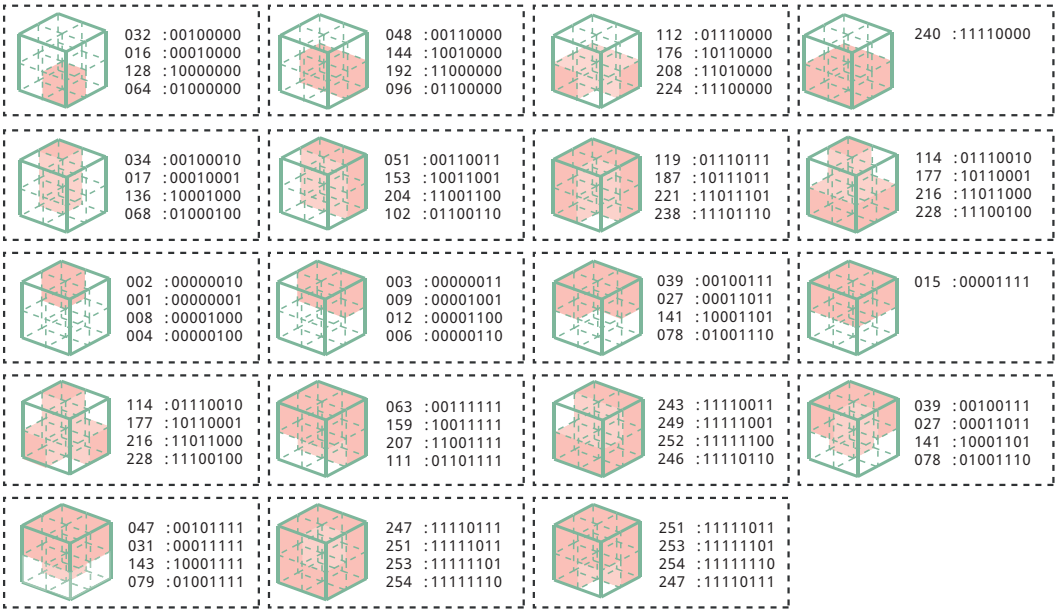


Fig.40: The 26 unique tilesets contain the information of 126 configurations. (Author)

3.1.4 Defining a module size

The dimension of the module size is one of the most important decisions to be made for modular construction. Of all the architectural building elements, the stair is most connected to the human scale as stairs are one of the most common objects and consider human ergonomics dimensions.

As a general equation established by french architect Francois Blondel, the stride length should span two risers and one tread, ranging from 500 to 700mm. Following the rule, the riser and tread sizes of the module are 180mm and 300 mm, which sum to 660mm.

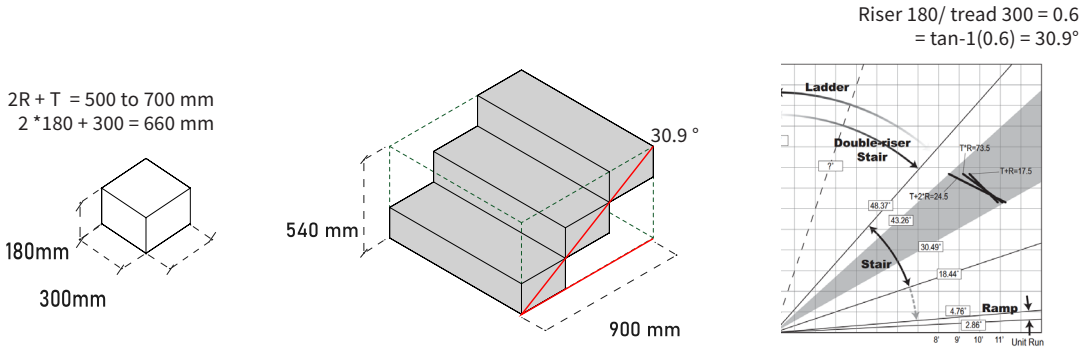


Fig.41: step module, stair module (Author) and standard staircase angle diagram [31]

The stair's angle can be calculated from the size riser and tread; the inverse tangent of 0.6 is a 30.9° angle in degrees, and the angle is within the optimal angle between 30.49 and 43.26 degrees for stairs [31].

For the horizontal dimension of the stair modules, As the modules are rotated and mirrored horizontally, the modules need to have the same length and width and a reasonably sized for human ergonomics. The module dimensions are trice multiplications of the steps sized 900 by 540mm in height.

The modules are stacked to generate architectural space, and the increase of the modules creates a height of each floor space. Six modules' stacks create a floor-to-floor height of 3.24m, making it a reasonable floor height, and stacking more or fewer modules will adjust the floor height by the height of modules(540mm).

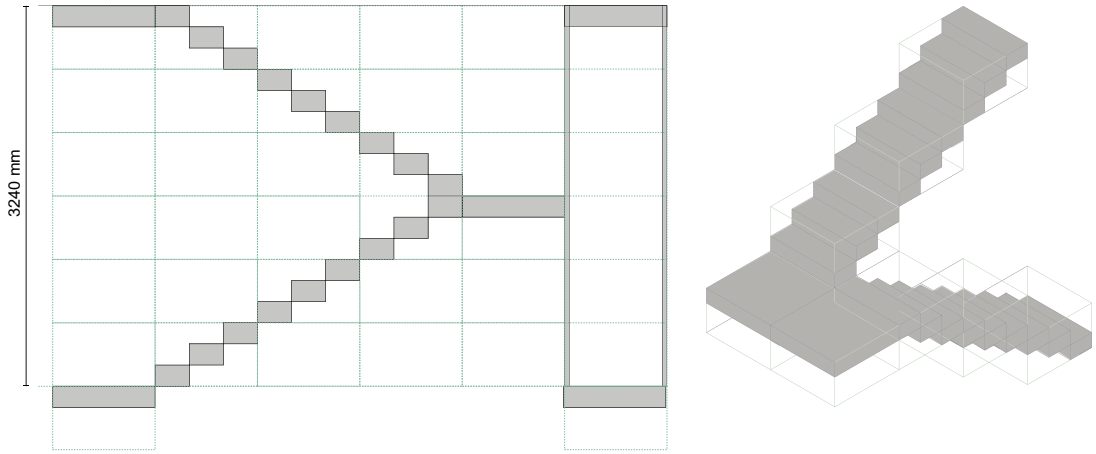


Fig.42: six stacks of modules to make one floor height (Author)

3.2 Surface Tilesets

3.2.1 Categorizing tilesets

As the cube configurations do not consider the vertical symmetry of sub-voxels due to the architectural context, the distinction of vertical placement of sub-voxels becomes the criterion for categorizing the tilesets. The placement of the sub-voxels can be divided by the top and bottom count order and the order can be categorized based on the architectural meanings of each tile as shown in (Figure 43).

- wall tiles - Top Subvoxels counts = bottom Subvoxels counts
- roof tiles - bottom Subvoxels counts > top Subvoxels counts
- floor tiles - bottom Subvoxels counts < top Subvoxels counts
- extra tile - The extra tile consists of the same 2:2 Subvoxels, but the upper and lower Subvoxelses are located in different places.

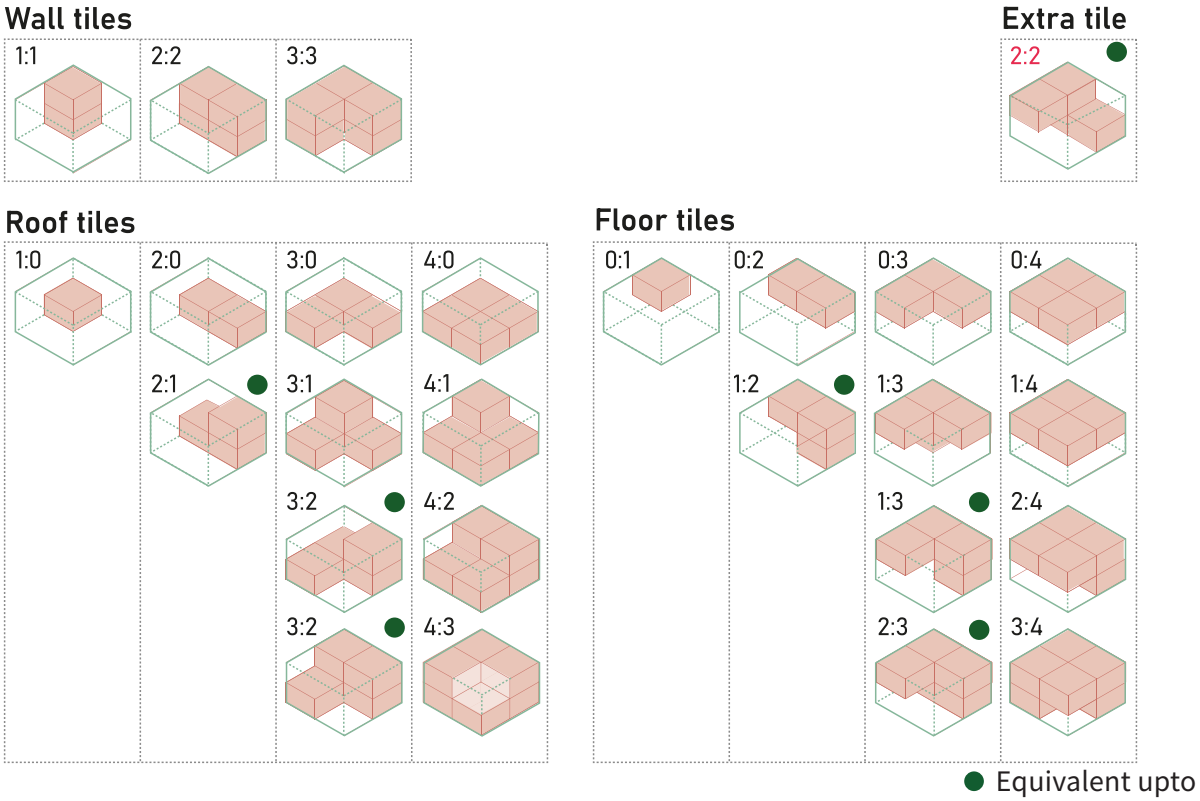


Fig.43: Tile classified of the 26 unique tiles in four functions, wall, root, floor, and extra. (Author)

3.2.2 Casestudy surface tileset 1

Surface tileset 1 - Surface tileset 1 is a simple surface tileset that generates a simple offset of the input voxel array as an output (Figure 45). The unique cube id represents that each tileset geometry is callable by the ID of the cube configurations (Figure 44)

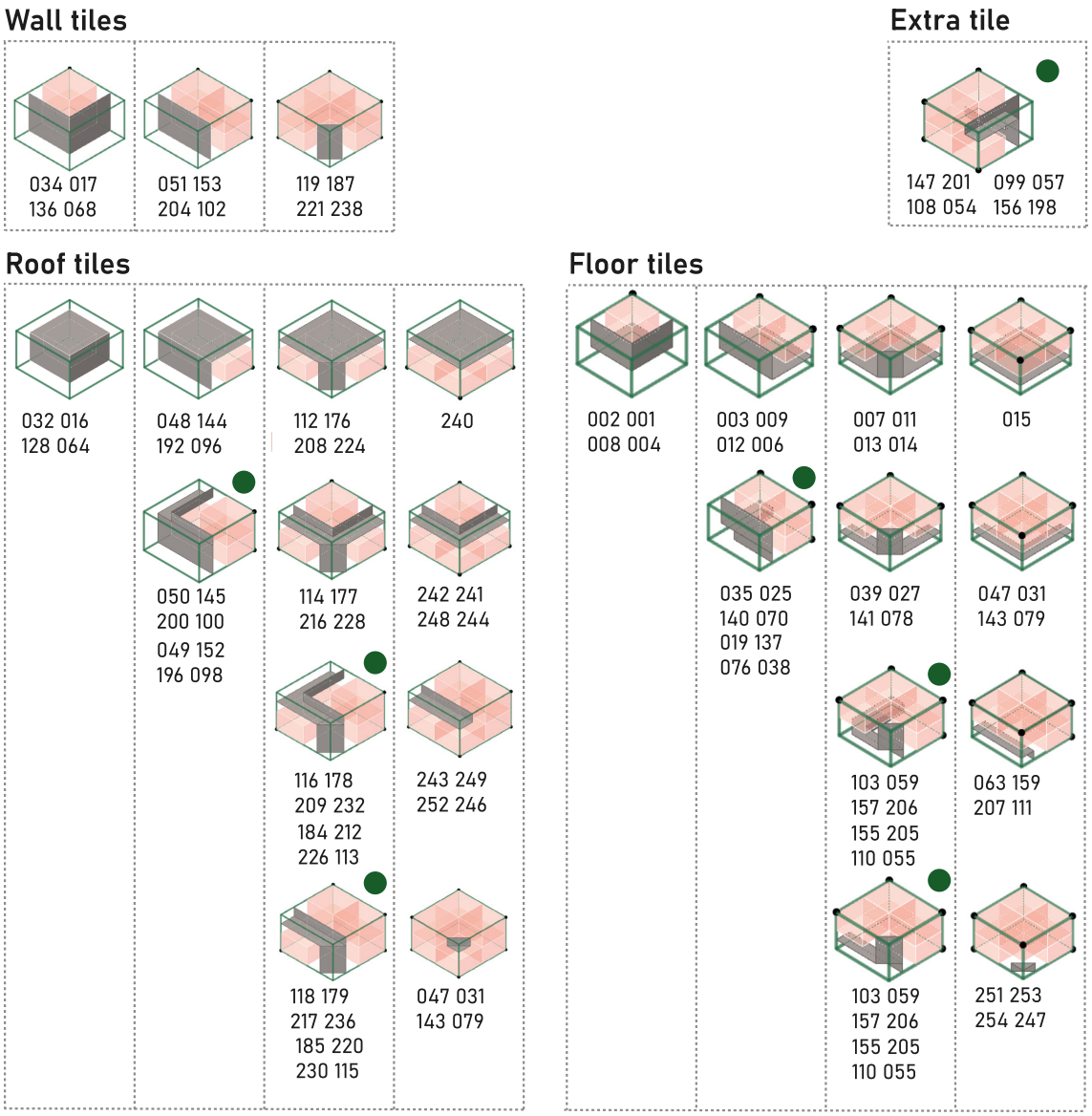


Fig.44: Surface tileset 1 visualized with Subvoxels and cube IDs. (Author)

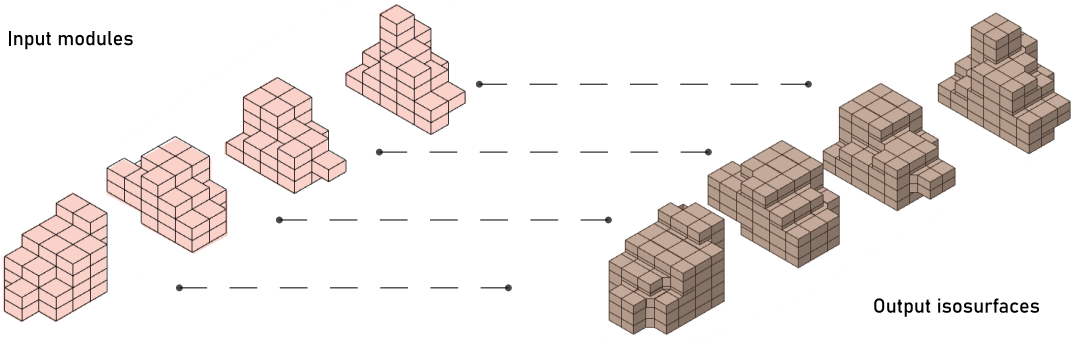


Fig.45: Resulting isosurface from surface tileset 1 with given input array of voxels (Author)

3.2.3 Casestudy surface tileset 2

Surface tileset 2 - The second case study of the surface tilesets intends to model geometry with more resemblance to the architectural tilesets. The result of the second surface tileset shows the roof shapes forming slanted roof connections of surfaces and floor surfaces.

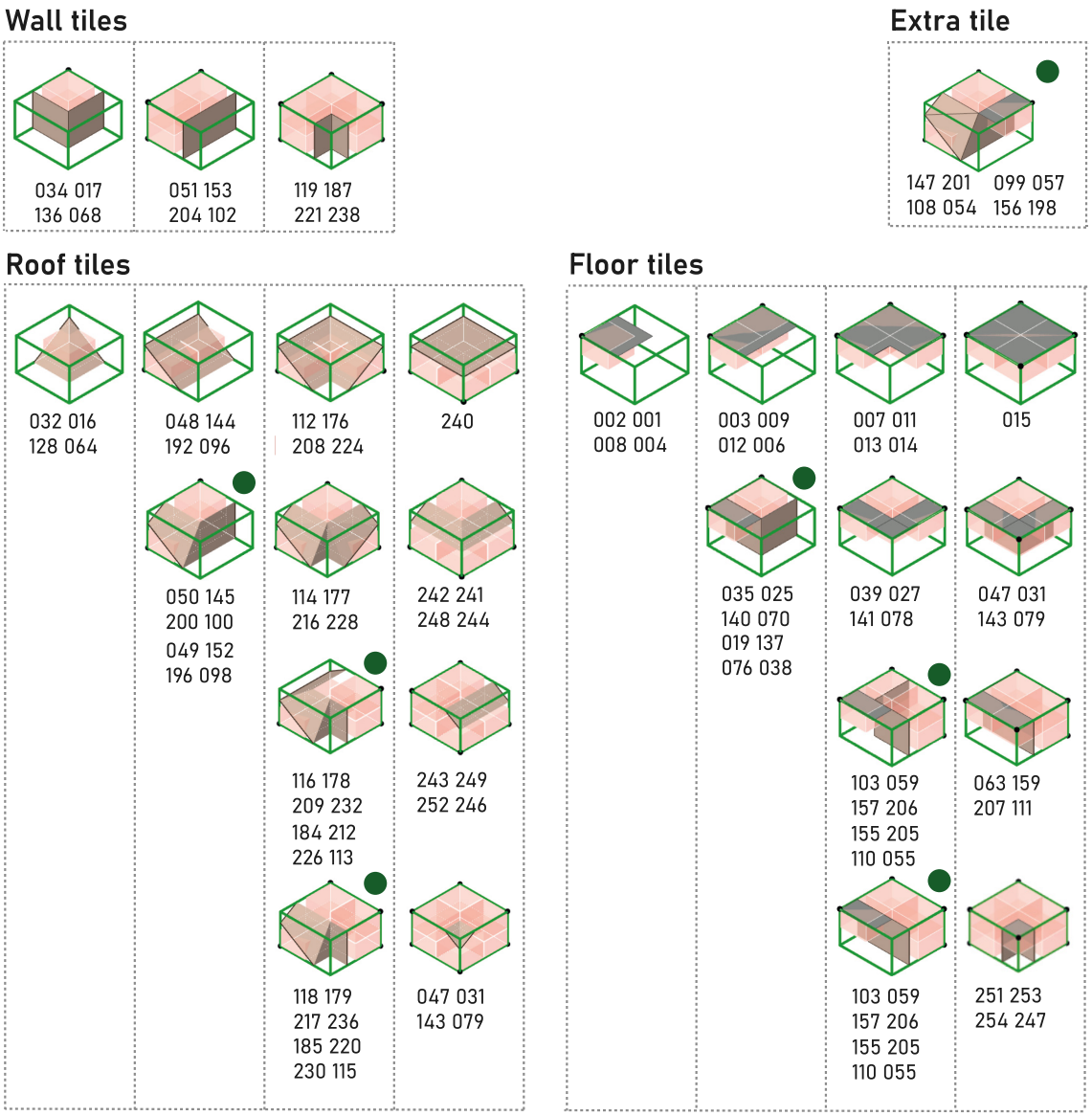


Fig.46: Surface tileset 2 visualized with Subvoxels and cube IDs.

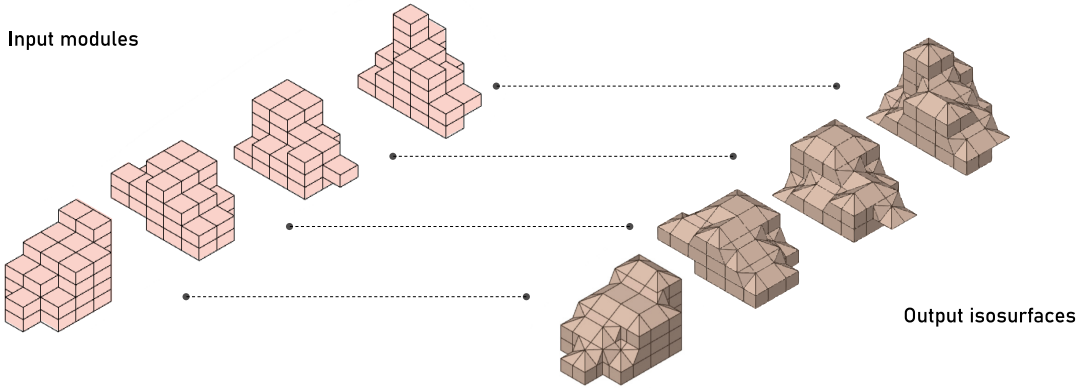


Fig.47: Resulting isosurface from surface tileset 2 with given input array of voxels (Author)

Roof tiles - The angle connecting the cube's top and bottom edge, which is the same as the stair's rise calculated, is essential for creating slanted roof surface tiles. In Figure 42, the angle on the left side generates seamless connections of surface tiles, as shown on the right side. For the tilesets to connect universally, the surfaces are placed inward to connect with the edges of other tiles, such as wall tiles.

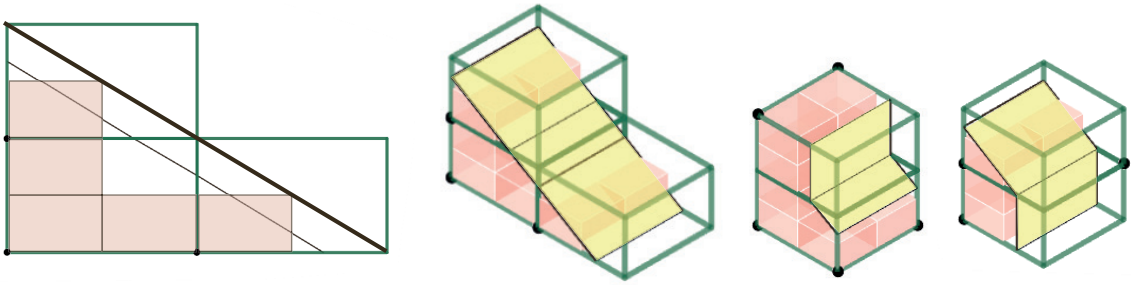


Fig.48: Roof tiles angle and the connection of roof tiles (Author)

Floor tiles - Unlike surface tileset 1, covering every voxel array as one, surface tileset 2 creates a surface outside the input voxel array. The floor tiles for surface tileset two are moved to the top of the cube configurations to maintain the floor-to-floor height. With increases in the number of modules, the floor-to-floor height increases by 540mm (equivalent to three stair steps).

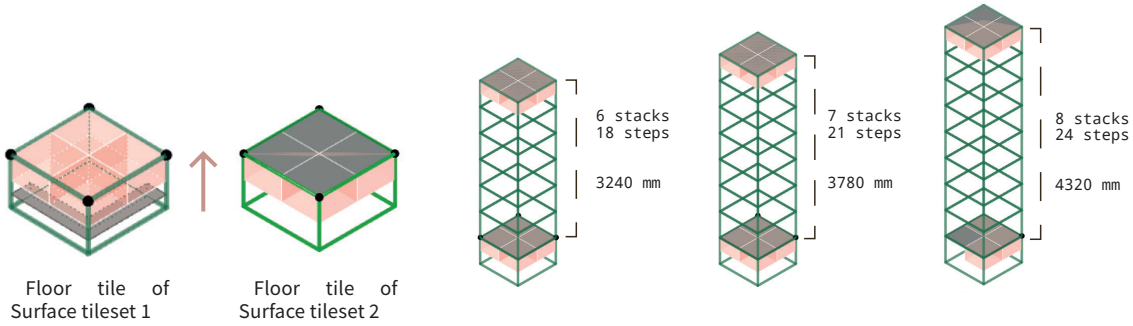


Fig.49: Diagram of floor tile and increase in the number of floor tiles (Author)

3.3 Architectural Tilesets

3.3.1 Architectural tilesets voxel types

The case studies of surface tilesets could generate a single type surface envelope. However, in architecture, multiple architectural elements of different materials and properties form a valid architecture. These architecture variations require various modules (voxels) to be recognized as distinct subjects. For the intuitive identification, their color and numbers are assigned to voxels.

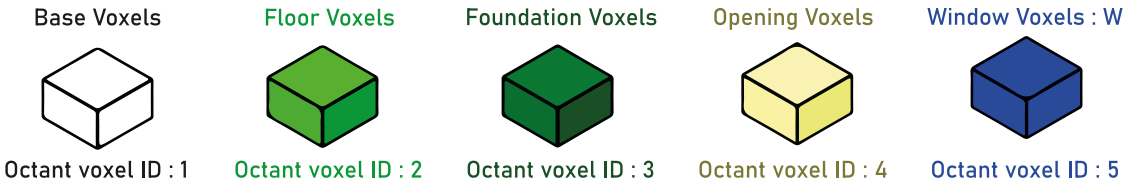


Fig.50: Five different voxel types representing a different architectural functions as colored voxels (Author).

The possible options for sub-voxels are six (0,1,2,3,4 and 5). The change from binary to heximal (6 elements) in 8-digit increases the possible cube configuration exponentially from 256 to 1679616 (6^8).

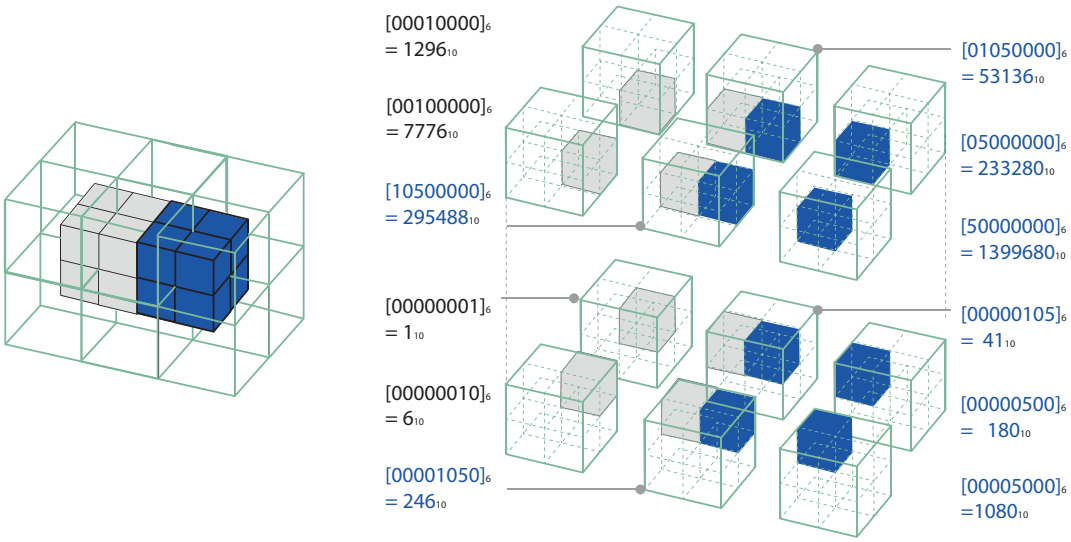


Fig.51: Method of defining cube id for color cube configuration of architectural tileset. (Author).

In heximal, the cube configuration orders the position and the kind of sub-voxels as 8-digit integers. In conversion to decimal, the numbers are the specific id used in the marching cube algorithm, which is also traceable by converting back to heximal integers of eight. It is essential to set design rules(3.5) and provide enough cube configurations to enable the algorithm to generate the wanted envelope using the modular components. Without assigned cube configurations, the algorithm leaves empty blocks without input geometry.

3.3.2 Architectural tilesets Overview

The architectural tileset contains the information for 109 tilesets, covering 493 situations of cube configurations. Due to the traits of architectural context, (described in each tileset chapter), the number of tilesets can be reduced by grouping the same geometries.

Base tiles	(containing only 1)	= 26 tiles
floor frame tiles	(containting 1 and 2)	= 25 tiles
foundation frame tiles	(containing 1 and 3)	= 10 tiles
foundation tiles	(containing only 3)	= 4 tiles
Opening tiles	(containing 1 and 4)	= 12 tiles
floor to opening tiles	(containing 1, 2 and 4)	= 10 tiles
foundation to openging tiles	(containing 1,3 and 4)	= 10 tiles
window tiles	(containing 1 and 5)	= 12 tiles

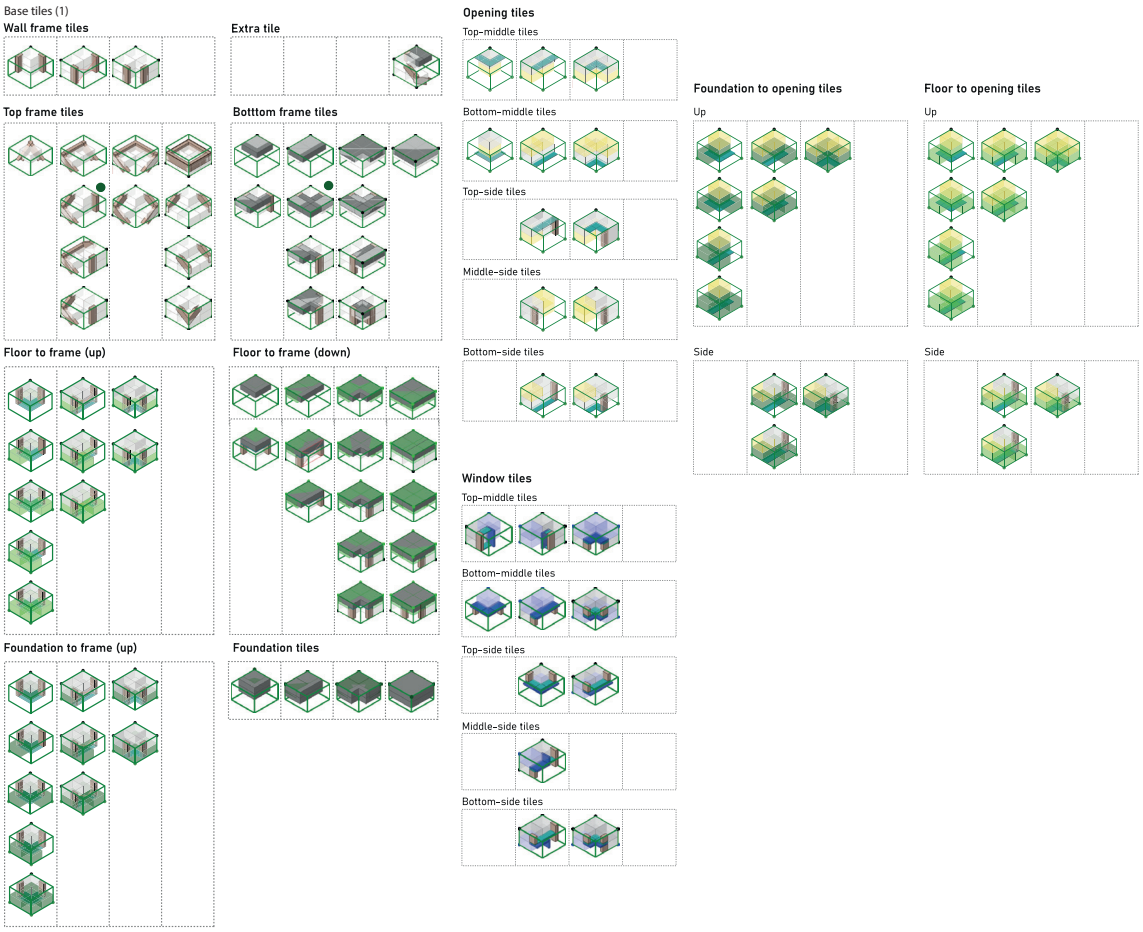


Fig.52: 109 Tileset before grouping tilesets with same shapes . (Author).

Grouping the same shaped geometry reduces the number of tileset modeling. The number of tiles reduces down to 72 tilesets but keeps the same number of configurations of 493 cube configurations.

Base tiles	(containing only 1)	= 26 tiles
floor, foundation frame tiles	(containing 1,2 and 3)	= 3 tiles
floor to frame down	(containing 1,2)	= 15 tiles
foundation tiles	(containing only 3s)	= 4 tiles
Opening tiles	(containing 1, 2 and 4)	= 12 tiles
window tiles	(containing 1 and 5)	= 12 tiles

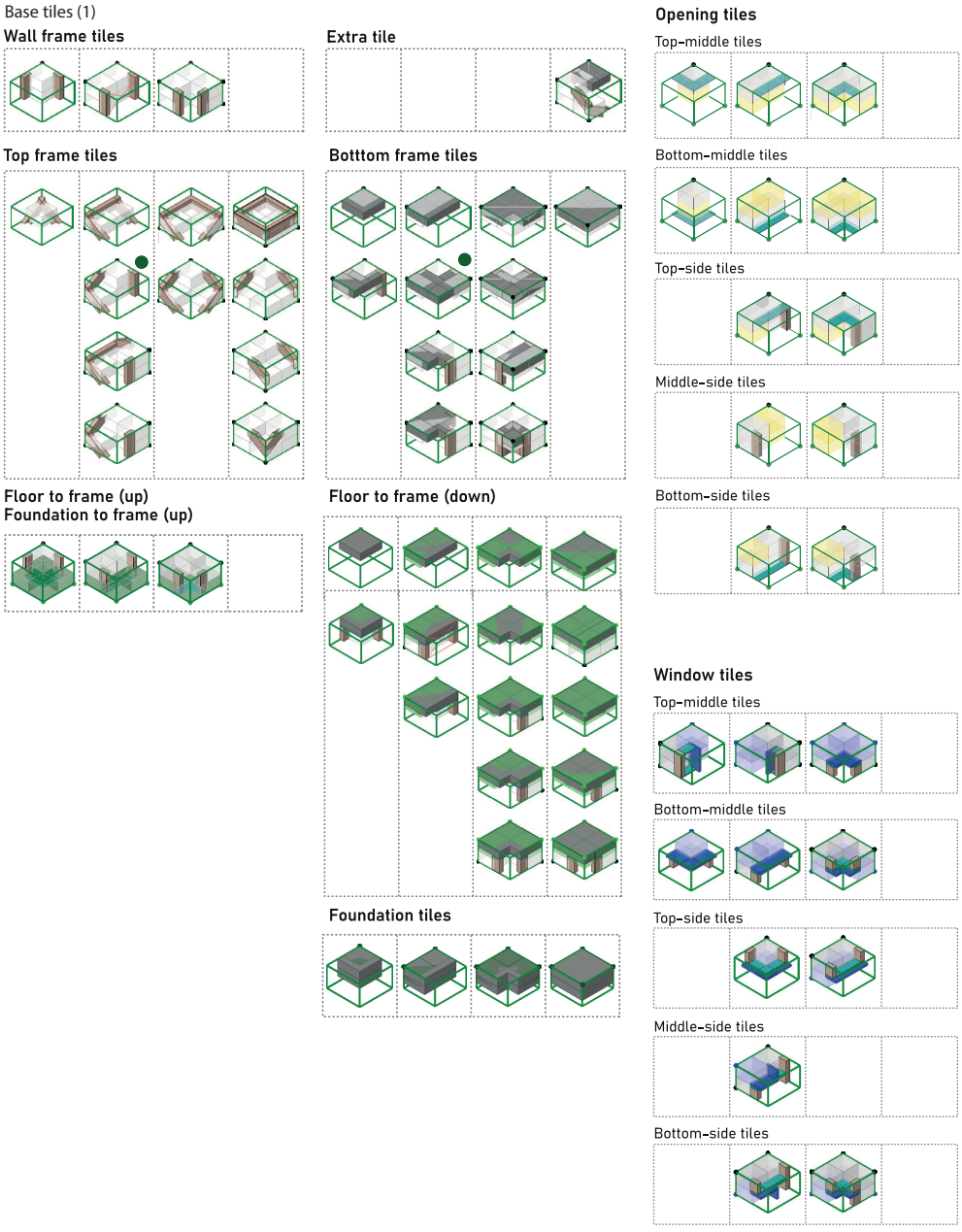


Fig.53: 72 Tileset after grouping tilesets with same shapes . (Author).

1) Base cube tileset

The Architectural base tilesets consist of only the base voxel configurations, meaning the configurations only consist of eight integers of 0 and 1 as the assigned number for base voxels is '1'. as the tileset consists of the same configuration as the surface tilesets, The categorization of the base cube tileset is identical to the surface tilesets. the base tilesets form the frame of the architecture.

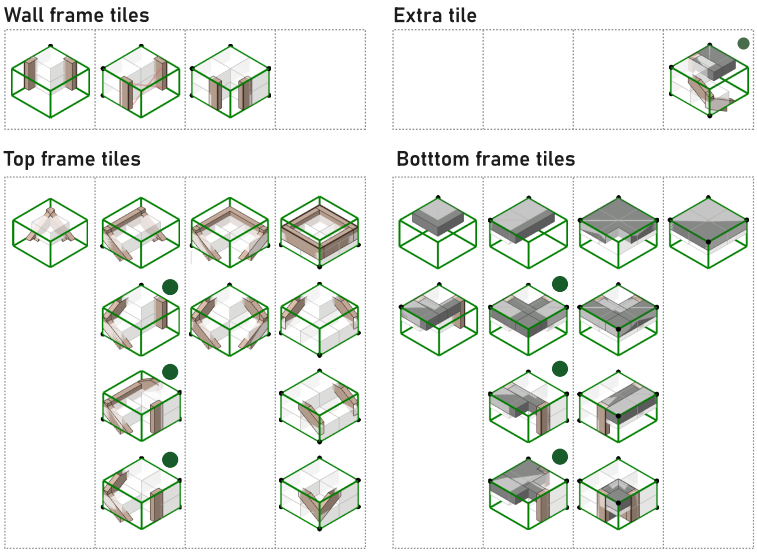


Fig.54: Categorization of Base cube tileset (Author).

Roof tiles geometries - Similar to Surface Tileset 2, the shape of the base tile's roof modular geometry consists of the same angle as the slope of the stairs. Because it is a connection of three-dimensional geometry, a continuous interlock of a given cube configuration is possible through modeling the connection, as shown in Figure 55.

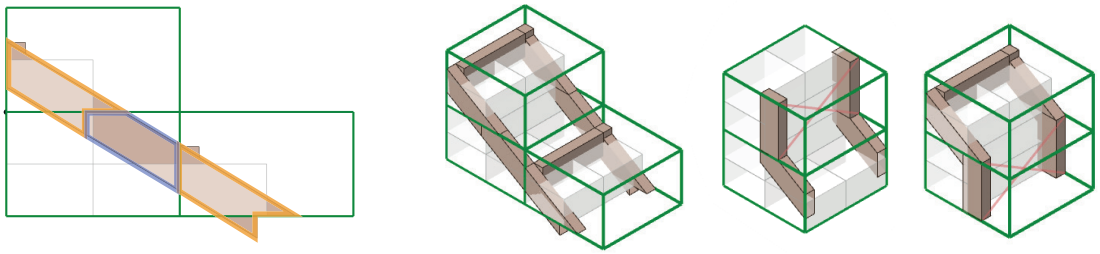


Fig.55: Roof tile geometry of the Base cube tileset, and the connections visualized (Author)

Floor tiles geometries - Same as surface tile 2, the floor geometry of the base floor tile is located on top of the cube. The thickness of the slab is 180mm, equivalent to one step height. As the height of the floor increases, the height increases by the module size, matching the maximum height of the stairs with the slab height—the floor-to-ceiling height increases by the module height, 540mm.

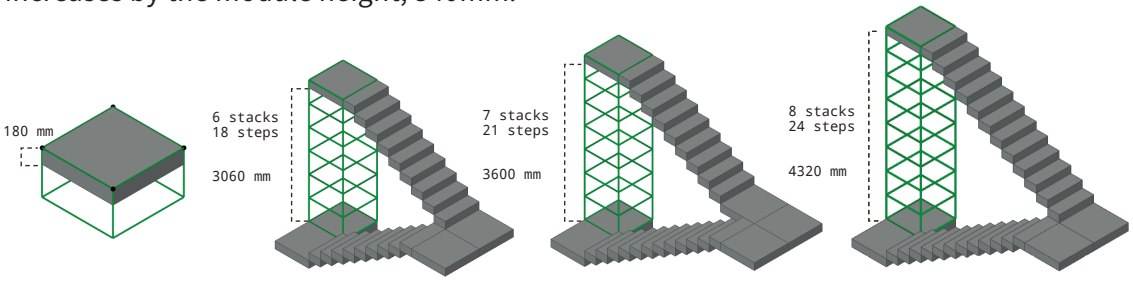


Fig.56: floor tiles and increase in height based on staircase design (Author)

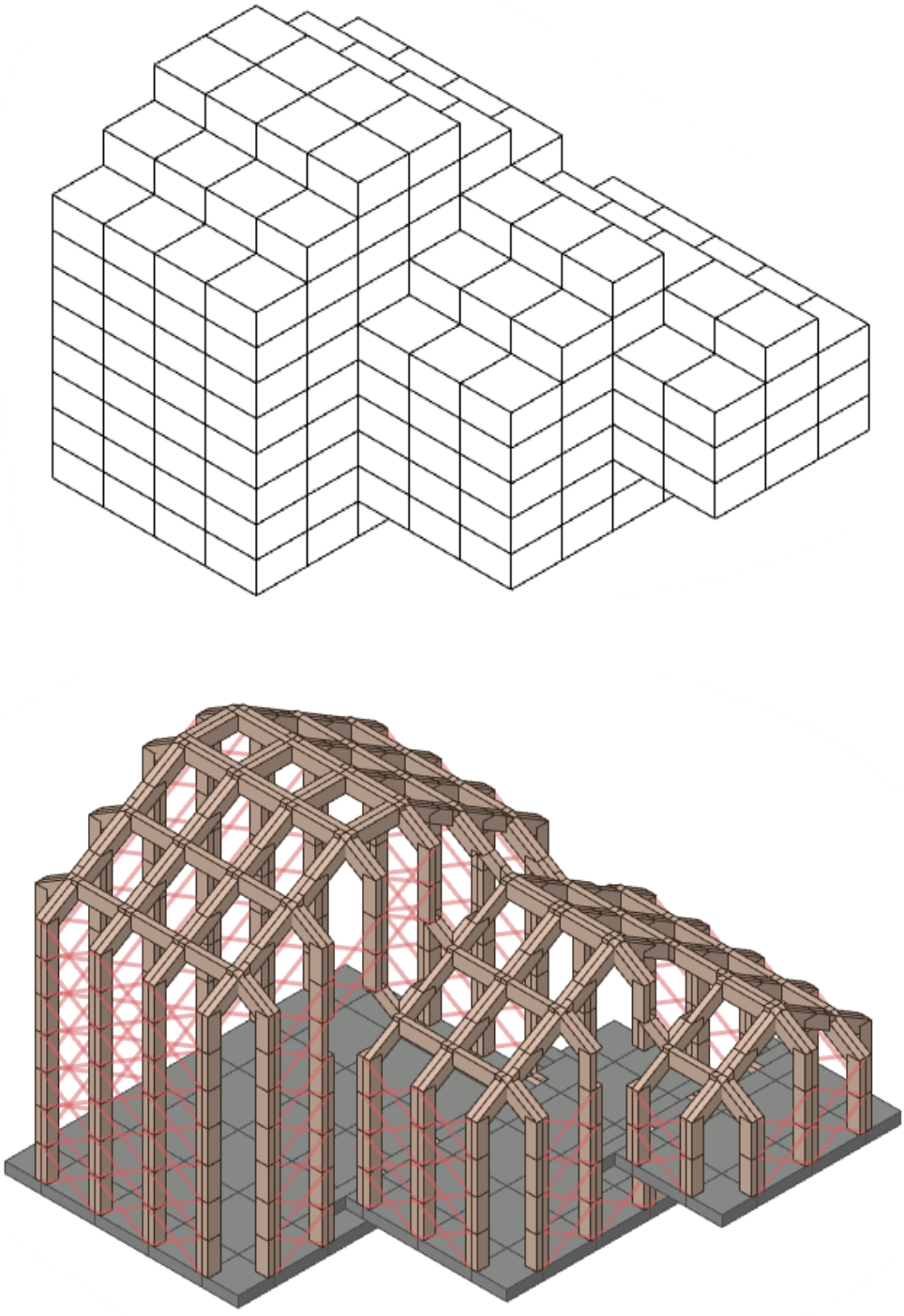


Fig.57: Generated frame using the Base cube tileset (Author).

2)Floor cube tileset

Light green cube tiles refer to the corresponding cube configurations between the floor voxel and base voxel. The sub-voxels of light green are represented by ‘2’ in the heximal configuration of 8-digits. The tileset is divided into two parts, the connection tiles that connect the floor to the frame upward and the tiles that connect the floor to the frame downward.

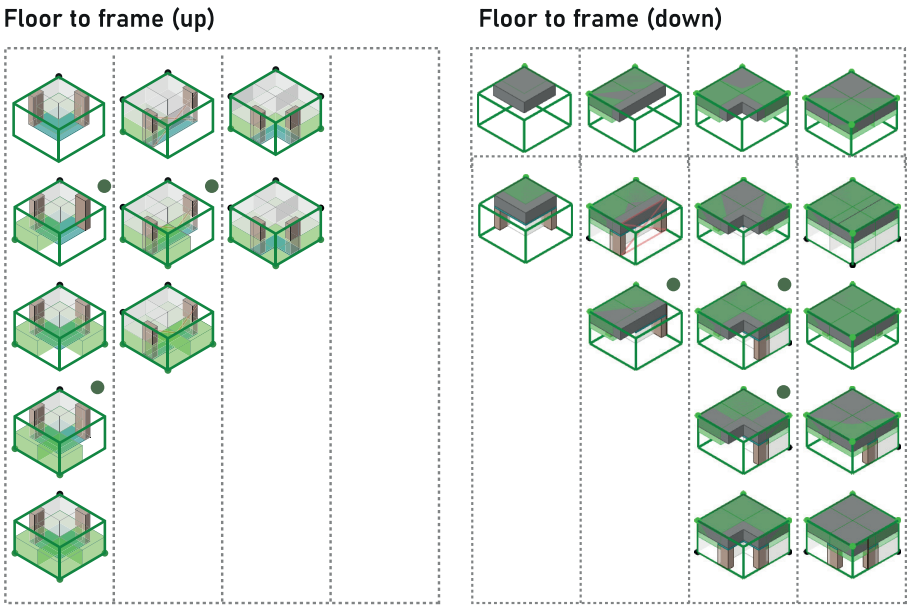


Fig.58: Categorization of floor cube tileset (Author).

3)Foundation cube tileset

The dark green cube tiles refer to the cube configurations above and under the foundation voxels, represented by ‘3’ in the heximal numbering of 8-digits. These tiles are divided into two parts, connecting the foundation with slabs and foundations.

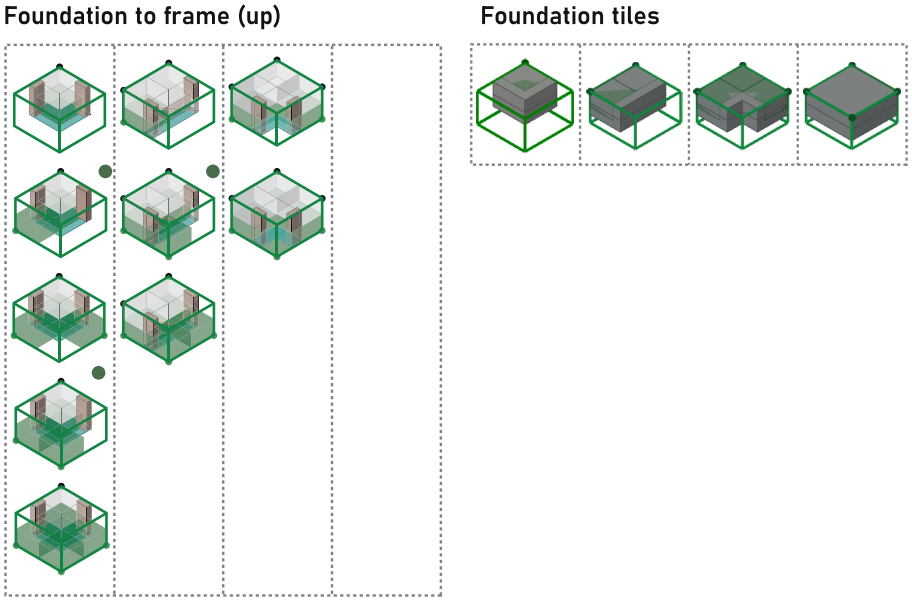


Fig.59: Categorization of foundation cube tileset (Author).

Since the floor geometries are lifted upward inside cube configurations, the configurations of the ‘Floor to the frame(up)’ tiles and ‘Foundation to the frame (up)’ tiles are only determined by the base voxels’ location at the top. When the base voxel is in the same position at the top, the geometry inside has the same shape. This redundancy of tiles with the same geometry is addressed by creating tiles of different codes of the same shape through the configuration group.

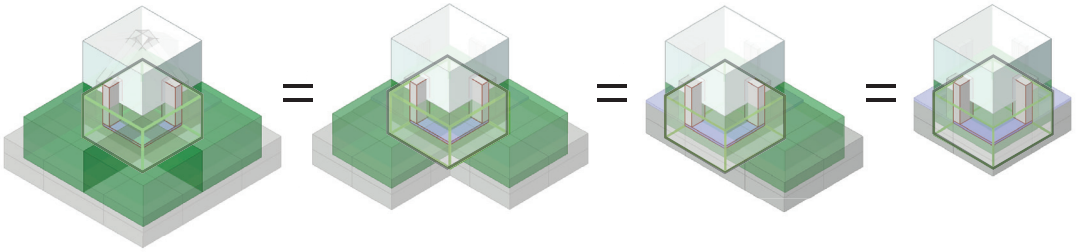


Fig.60: Grouping the same geometry of floor and foundation tileset (Author).

The ‘floor to frame(up)’ and ‘foundation to frame(up) tiles are grouped into three tiles. The grouped tiles maintain the same configurations as the original number of 39.

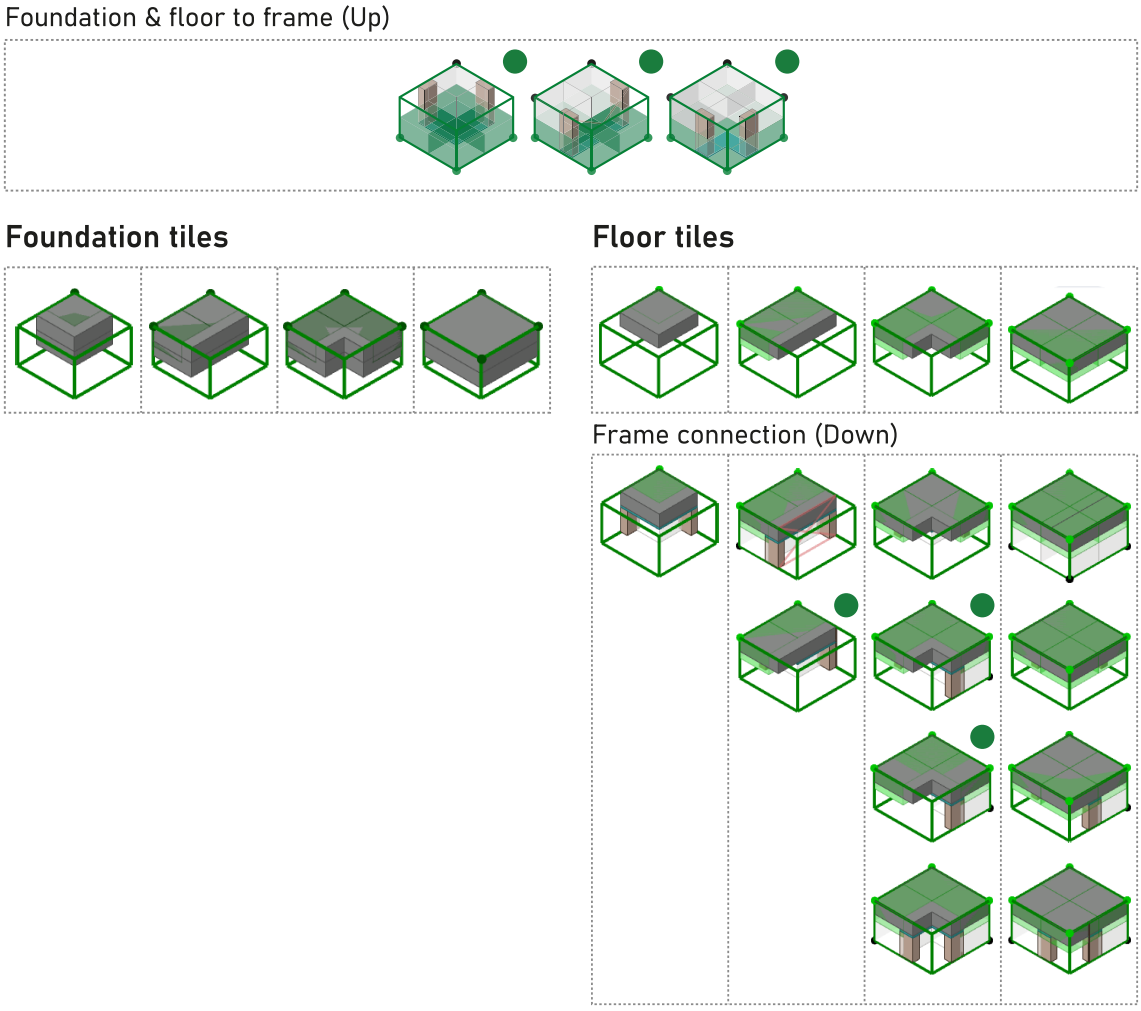


Fig.61: Categorization of foundation cube tileset (Author).

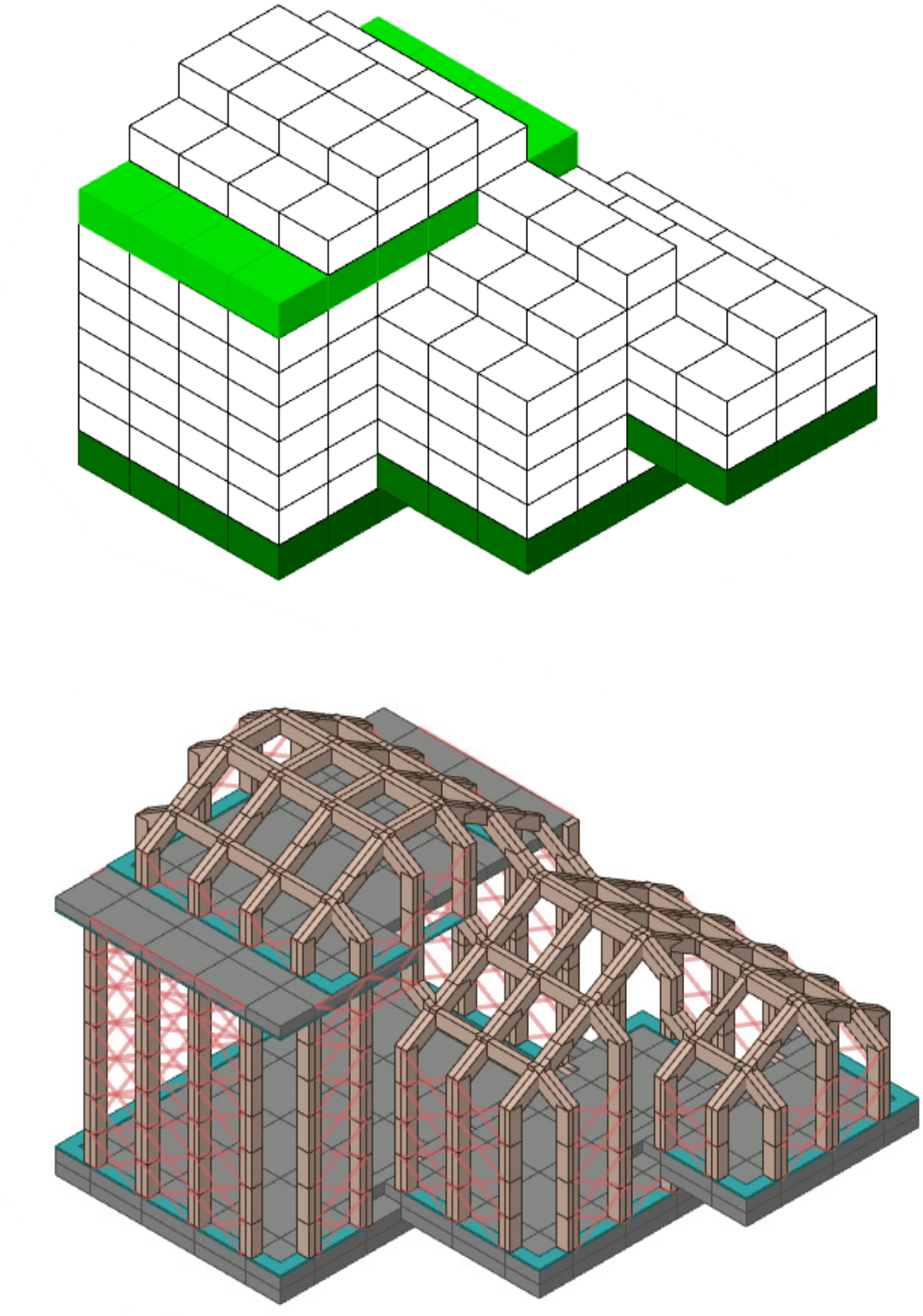


Fig.62: Resulting polygonization from Base and floor, foundation tilesets (Author)

4) Opening cube tileset

The yellow cube tiles refer to the cube configurations of the opening voxel and three different voxels, the base, floor, and foundation tiles. The number of opening cube tiles is 30, but by grouping the same shapes, the tileset required reduces to 10 tiles containing all information. The integer representing opening sub-voxels is ‘4’ in the heximal numbering of 8-digits.

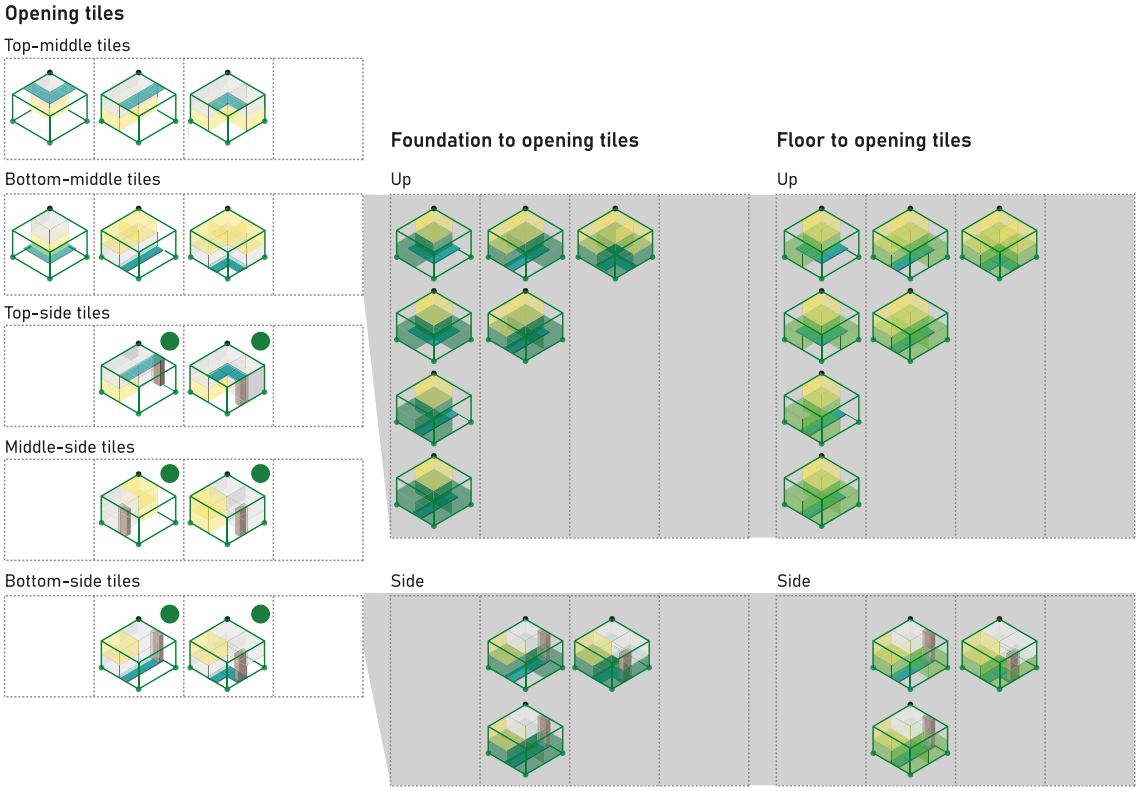


Fig.63: Categorization of opening cube tileset (Author)

The opening voxel must connect to the floor or foundation tile to create an opening for a door starting from the floor or foundation tile. For this to be possible, this situation must be specified as a cube tile. This designation of specific conditions allows the bottom surface of the opening voxel to connect to the foundation and floor voxel.

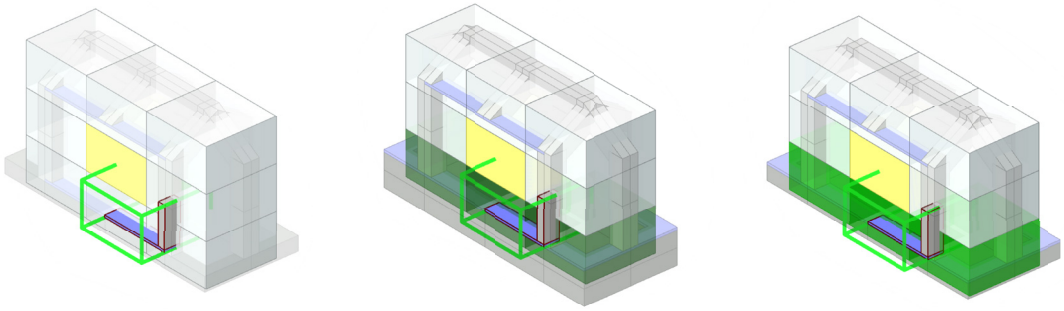


Fig.64: Polygonization of opening tileset from Base and floor, foundation tilesets (Author)

5) Window cube tileset

Window tiles

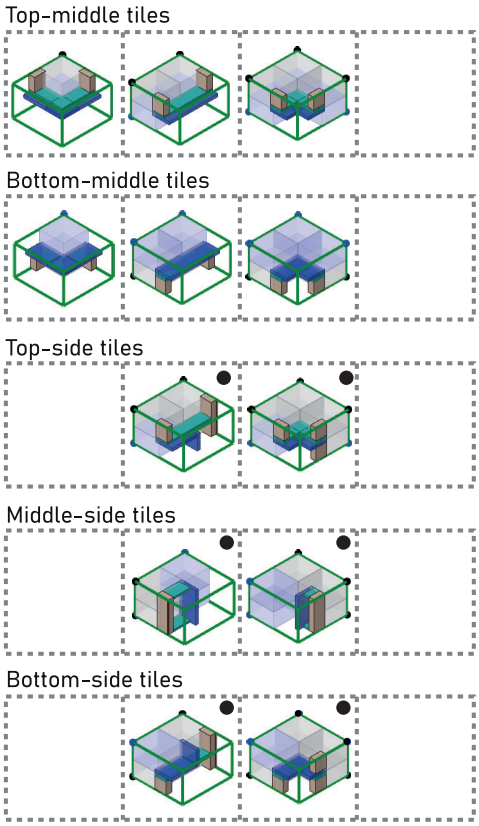


Fig.65: Categorization of window tilesets. (Author)

The tilesets of the blue octant voxel represent the window tiles. As a design rule, The base voxels surround the window tiles. The number of tiles is ten and is equivalent to 72 situations of horizontal rotation and mirroring.

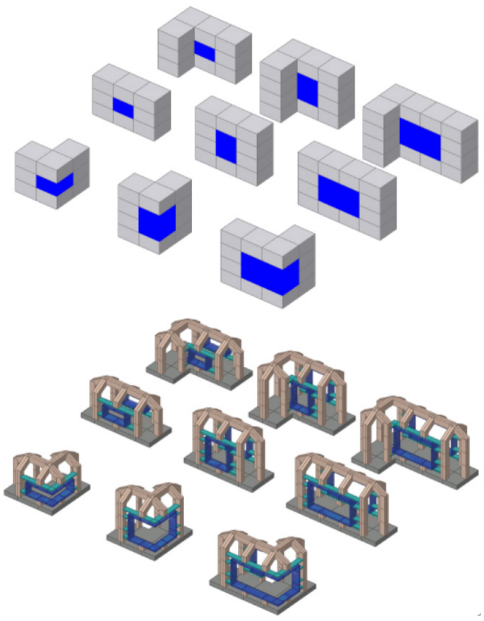


Fig.66: possible placement of window voxels and resulting envelope

The number of possible cube configurations of the window tileset is only 72, meaning that the window voxels have some restrictions in placing and, as a rule (3.5), need to be surrounded by base voxels to bring the cube configurations assigned to them.

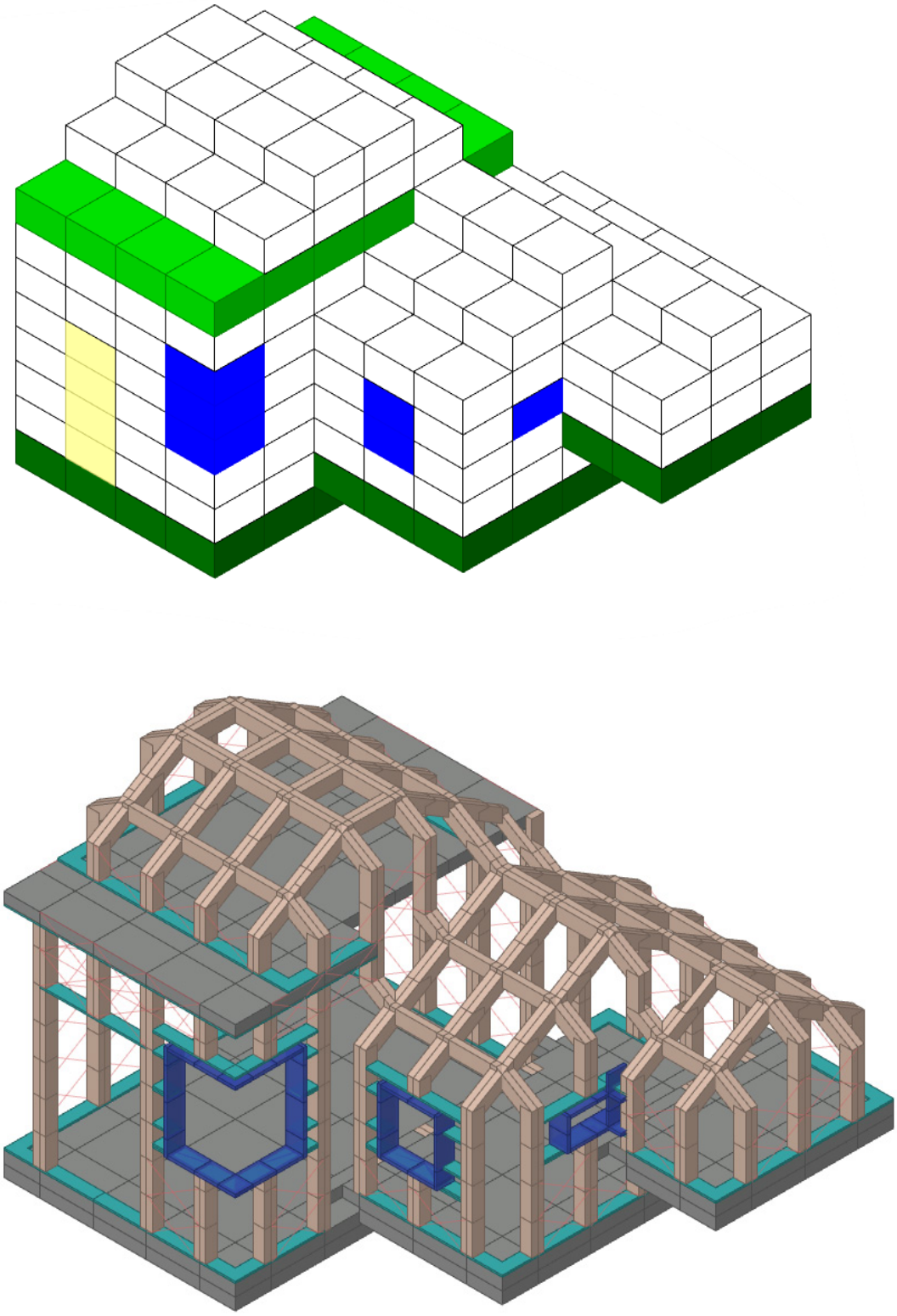


Fig.67: Resulting polygonization from every tilesets with given input array of voxels (Author)

3.4 Algorithm

3.4.2 Algorithm Overview

The Houscaper algorithm is developed under a grasshopper environment using both grasshopper components and python scripting. The script categorizes into Cube ID, Cube options, Cube placing, and structural analysis.

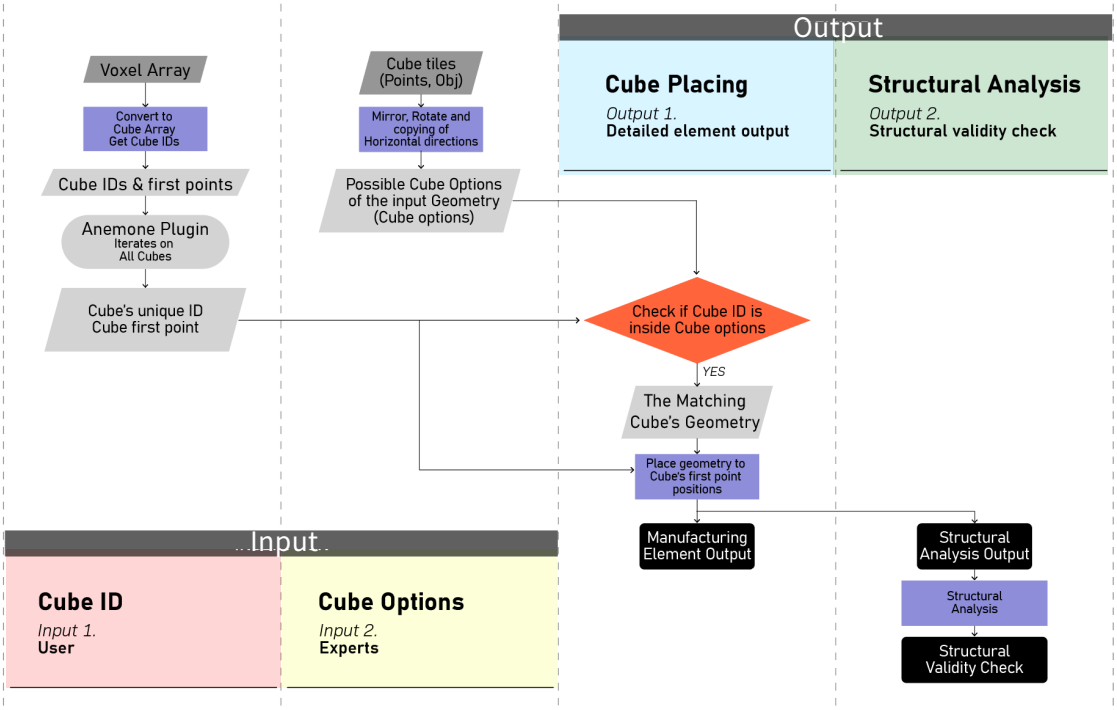


Fig.68: BMC Algorithm diagram of the thesis project (Author)

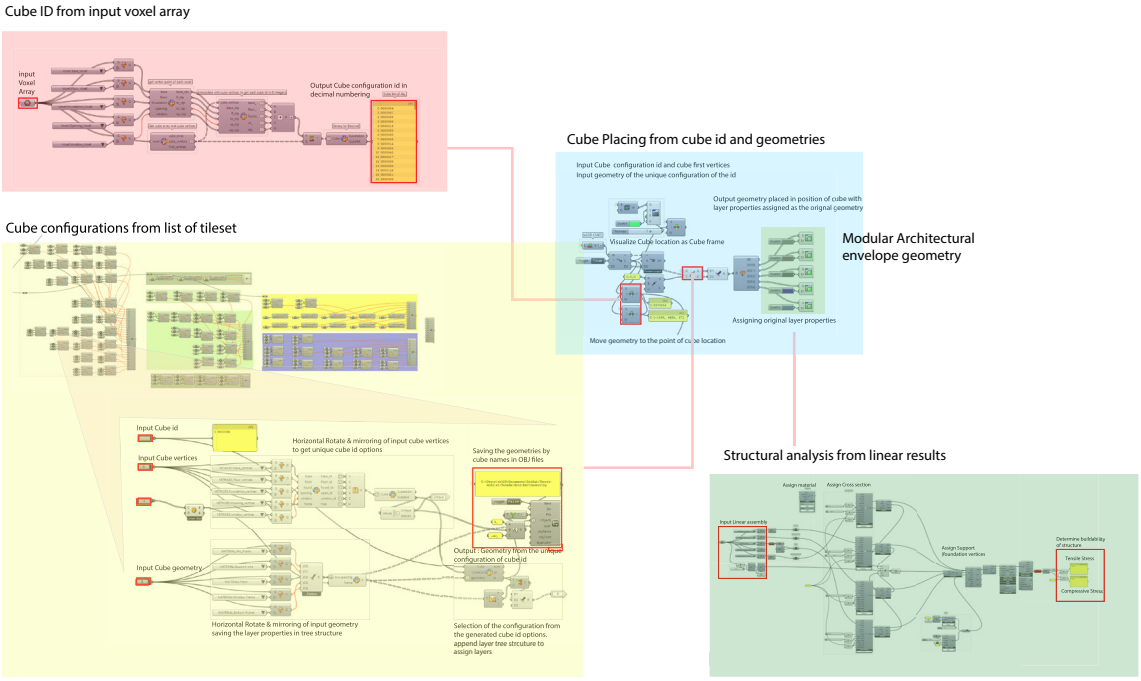
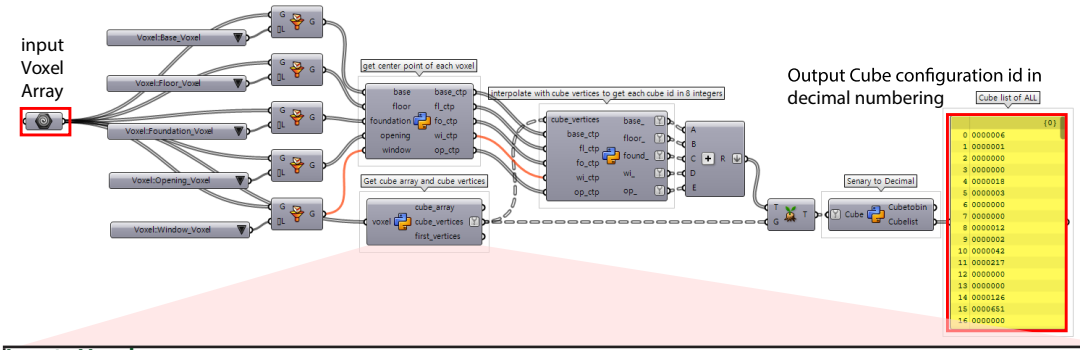


Fig 69: BMC Algorithm diagram of the thesis project in grasshopper environment (Author)

3.4.2 Cube ID algorithm

Cube Array - The first part of the Cube id algorithm is converting the input Voxel array into a Cube array. Inside grasshopper, the python script component enables the scripting of the algorithm. Below is a pseudo-code of the cube array algorithm generating the cube array list and first vertices list from the input array. The original codes are in appendix 3.



Input : Voxel array
Output : Cube array, Cube vertices, first vertices of the cube array

make empty list of centerpoints
for elements in **voxel:**
  append centroids of the **voxel** in the center point list

get a voxel as onebox
get a bounding box of the voxels as boundingbox

deconstruct onebox to get X.Y.Z value of onebox
deconstruct boundingbox to get X.Y.Z value of the boundingbox

boundingbox's X[1] - X[0] divide by onebox's X[1] - X[0] gives the X count, add 1 to it for cube array count X
boundingbox's Y[1] - Y[0] divide by onebox's X[1] - X[0] gives the Y count, add 1 to it for cube array count Y
boundingbox's Z[1] - Z[0] divide by onebox's X[1] - X[0] gives the Z count, add 1 to it for cube array count Z

assign 'start' the first point of onebox[0]
assign 'centerpoint' the centroid of onebox

subtract start with centerpoint to get a vector
move onebox using the vector as 'moved'

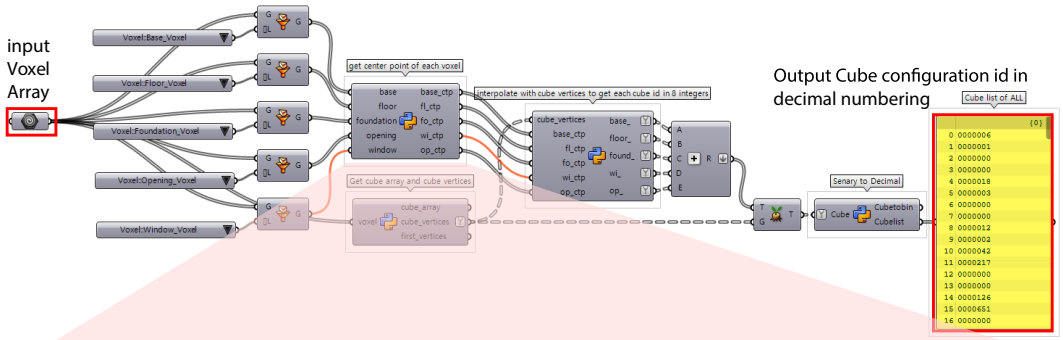
make an array of boxes using the moved box by the cube array count of X.Y.Z = **Cube array**

create an empty list of first cube_vertices
create an empty list of cube vertices

for elements in **cube array,**
  get box corners of cube array
  append first cube vertices the first elemnt of elements in **first cube_vertices**
  append all cube vertices in the **cube vertices** list

Fig.70: Pseudocode of the cube array algorithm (Author)

Cube's id assigning - In the algorithm, the center points of the voxel are read as an integer, and a code consisting of a total of 8 integers indicates the composition of the subvoxel in the cube. The pseudo-code below extracts the cube configuration of eight integers from the cube array, converts the heximal integers into decimals, and lists them. (original code in appendix 4)




```
input : Cube vertices, centerpoints of voxels by types
Output : cube id in 8 integers

define intersect function getting (centerpoints and cube_id_number)
  create an emptylist
  for elements in the cube vertices
    Member index of elements by the centerpoints as an indexlist
      create an binary_list
      for elements in indexlist
        if elements is []
          element is 0
          append element in binary_list
        elif elements is not []
          elememnts is the cube_id_number
          append elements in binary_list
    return the value of binary_list

define by_eight function getting (binarylist)
  for elements in binary_list
    separate all elements by 8



run intersect on base voxel centerpoints (base centepoint,1)
run intersect on floor voxel centerpoints (floor centerpoint, 2)
run intersect on foundation voxel centerpoints (foundation centerpoint, 3)
run intersect on opening voxel centerpoints (opening centerpoint, 4)
run intersecton window voxel centerpoints (window centerpoint, 5)

run by_eight  on base intersect
run by_eight  on floor_intersect
run by_eight  on foundation_intersect
run by_eight  on opening_intersect
run by_eight  on window_intersect
```

The combinations of algorithms of 'cube array' and 'assigning cube id' provides the unique cube id of every cube configurations inside the given voxel array.

Fig.71: Pseudo code for Cube id assigning algorithm (Author)

3.4.3 Cube Options (tileset) Algorithm

Cube options - The generated cube id needs to be searched from a list cube configuration. In the thesis, the list is a tileset described in previous chapters. Tileset made in rhino grasshopper is saved in grasshopper and called when the necessary cube ID is specified. However, for future use and reproducibility, the tile has set up an algorithm to store outside the Rhino. Below is an overview of the tileset generation algorithm.

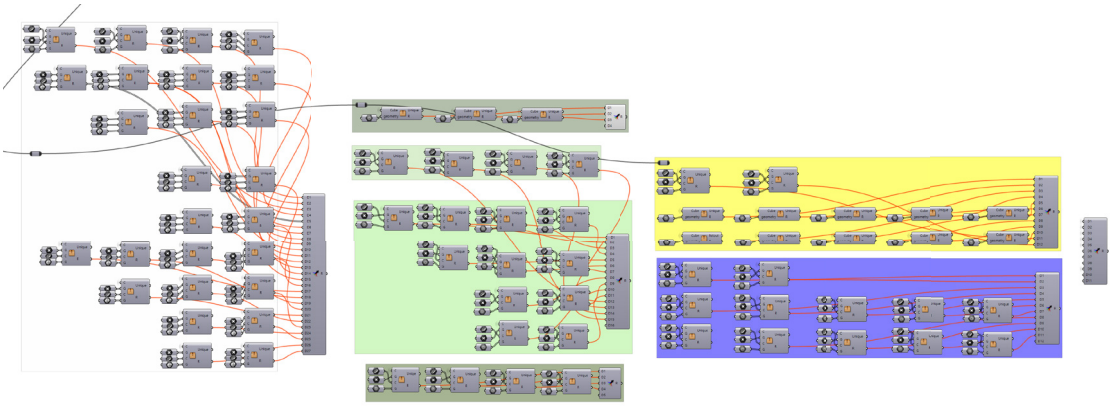


Fig.72: Overview of Cube options (tileset) generating algorithm

Inside each cluster, the input objects and points are rotated and mirrored in a 90-degree horizontal direction. After member indexing of the vertices, translate the index into the eight digits of heximal numbers and assign the unique id of the number in decimal.

The tileset geometry is saved as an obj file in the assigned directory outside the Rhino environment with a click of the bake button. When a cube id matches with a generated cube option, the algorithm searches for the geometry of the cube id and sends it over to the cube placing algorithm.

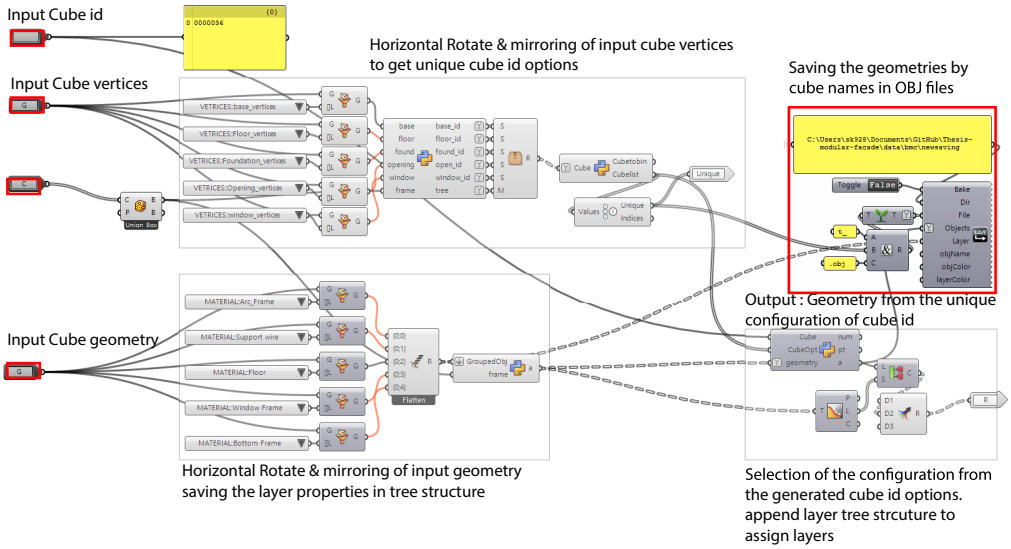


Fig.73: Overview of Cube options inside clusters (tileset) generating algorithm

The pseudo-code for cube option generation is as follows, original code is in appendix 6.

```
Input: Cube frame, Sub-voxels as vertices
Output: Cube options
Define function of 'mirror_rotate' # to rotate and mirror get 'equivalent upto' vertices
  Create an empty list
    for range of 4:
      Rotate objects 90 degrees four times
      append in the empty list
    Mirror the the last rotated object in YZ plane for horizontal mirror
    for range of 4:
      Rotate the mirrored object 90 degrees four times
      Append in the empty list
    Return the result list

Run function 'mirror_rotate' on base vertices
Run function 'mirror_rotate' on floor vertices
Run function 'mirror_rotate' on foundation vertices
Run function 'mirror_rotate' on opening vertices
Run function 'mirror_rotate' on window vertices

define intersect function getting (centerpoints and cube_id_number) # intersect with cube vertices to get id
  create an emptylist
  for elements in the cube vertices
    Member index of elements by the centerpoints as an indexlist
      create an binary_list
      for elements in indexlist
        if elements is []
          element is 0
          append element in [binary_list]
        elif elements is not []
          elememnts is the cube_id_number
          append elements in binary_list
    return the value of binary_list

define by_eight function from binarylist
  for elements in binary_list
    separate all elements by 8

run intersect on base vertices (base centepoint,1)
run intersect on floor vertices (floor centerpoint, 2)
run intersect on foundation vertices (foundation centerpoint, 3)
run intersect on opening vertices (opening centerpoint, 4)
run intersect on window vertices (window centerpoint, 5)

add the heximal list in to one added list
convert the added list in to Deicmal to make cube options
```

Fig.74: Pseudo code for Cube option generation algorithm (Author)

```
Input: Cube frame, tileset geometry
Output: Geometry Options
origin_point = Point3d(0,0,0)
cube_cornerpoint = get the first point from the frame[0]
create vector from origin_point to cube_cornerpoint
Move object to the origin point using the vector # for later saving as OBJ files

Define function of 'mirror_rotate' # generate 'equivalent upto' tiles of given geometry
  Create an empty list
    for range of 4:
    Rotate objects 90 degrees four times
      append in the empty list
  Mirror the the last rotated object in YZ plane for horizontal mirror
    for range of 4:
    Rotate the mirrored object 90 degrees four times
    Append in the empty list
    Return the result list

mirror_rotate the moved object # 90 degreee rotate mirror and save the geometry
```

Fig.75: Pseudo code for geometry option generation algorithm (Author)

```
Input : Cube ID, Cube Options, Cube Geometry
Output : Cube geometry of an assigned cube id

for each option in cube options:
  if the option equals the cube id:
    get the index of the cube option from the cube options
    get the cube geometry from the cube options using the cube id
    return the cube geometry of the cube id

the returned element is the cube geometry of cube id
```

Fig.76: Pseudo code for searching for cube id in cube options and assigning geomtery

The cube options are checked for an intersection with the cube id received from the earlier stage. as shown in the pseudo-code of comparing script below, When a cube id matches with one of the cube options stored the algorithm takes the exact configurations to be loaded and located to the cube position. (Appendix 8)

3.4.4 Cube placement Algorithm

Cube geometry placing - The later stage is the placement of the cube geometry to the cube position. An iterative plugin called 'anemone' is used to visualize the process more intuitively. From the cube options, all the cube geometry options are moved to the origin point of (0,0,0) and saved as obj files. In the algorithm, the geometries are transferred to the first point of the cube and, using the tree structure of the geometry, assign original layer properties when baked.

Input Cube configuration id and cube first vertices
Input geometry of the unique configuration of the id

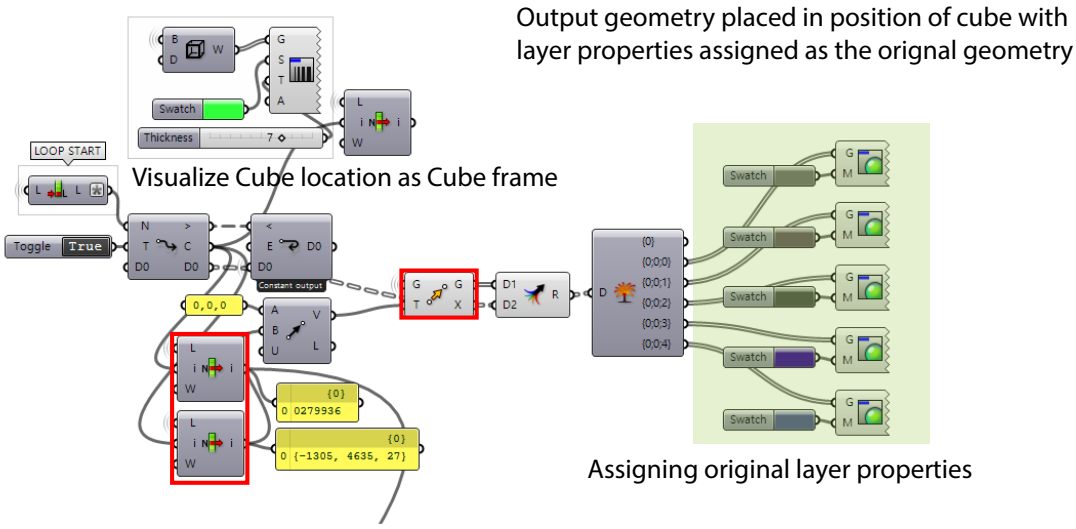


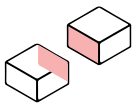
Fig.77: Overview of Cube placing algorithm(Author)

Architectural elements consist of various materials with different traits and properties. Dividing the layer properties provides the freedom of differentiating materiality to enable the separation of geometry. However, in grasshopper, when geometries are grouped, the layer properties are ignored and merge into one. The layer properties need to be re-assigned to the grouped objects to solve the issue. This is enabled by separating materials using layer filter components and saving the object layer division in layers as a tree structure. After selecting the matching cube configuration from the options, the tree structure is applied to the configuration to separate the material layers. The figure shows the separation of these layers in detail and how a configured cube information is divided using the algorithm in grasshopper.

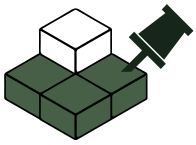
3.5 Interactivity

3.5.1 Design rules for voxels (normal users)

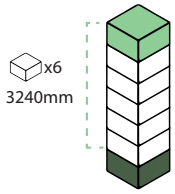
The participatory design of an architectural envelope requires design rules set. Of the cube options generated as tilesets, the cube configurations made do not cover most of the cube configurations as there are 16796166(6^8) possibilites and the generated configurations are only 493. Although the thesis aims to reduce and minimize the complexity of architecture into a more straightforward participatory design for users by 'assist sculpting,' The voxels need some rules to address users to build a structure within the scope of tilesets.



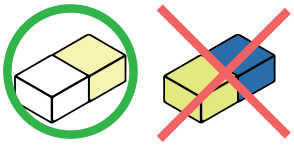
1) All voxels must surface connect



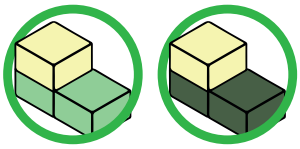
2) Foundation voxels should be placed at the bottom of the voxel array to create the foundation of the architecture.



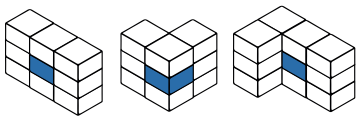
3) One-floor height is established with more than five stacking of base tiles and one floor tiles, to achieve floor to ceiling height of 3060mm. (more stacks increase by module height of 540mm)



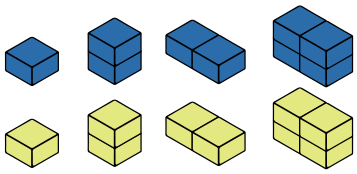
4) Color voxels (yellow, blue, green, dark green) cannot be placed neighbor to each other, need white voxels in between with one exceptional case.



5) One exceptional case is the yellow voxel on top of green and dark green voxels. the configuration of such is assigned in opening cube configurations (3.3.2.4)



6) The window voxels are always surrounded by the base voxels and the minimum units available for placement figure on the figure on the left.



7) Rectangular placement of blue and yellow voxels results in generating bigger openings for the window and opening frames.

Fig.78: Voxel placement ruleset (Author)

3.5.1 Design rules for tilesets (tileset designers)

The possible cube configurations increase exponentially with the increase of kinds of the tileset. This increase brings more possibilities and demand for professional designer/ architect users to create more configurations and own designs for the system. In the grasshopper environment, such tilesets are saved as cube options within each cluster and loaded when the cube id and cube options are a match. The script also enables the saving of geometries as obj files outside the Rhino grasshopper environment for other use cases.

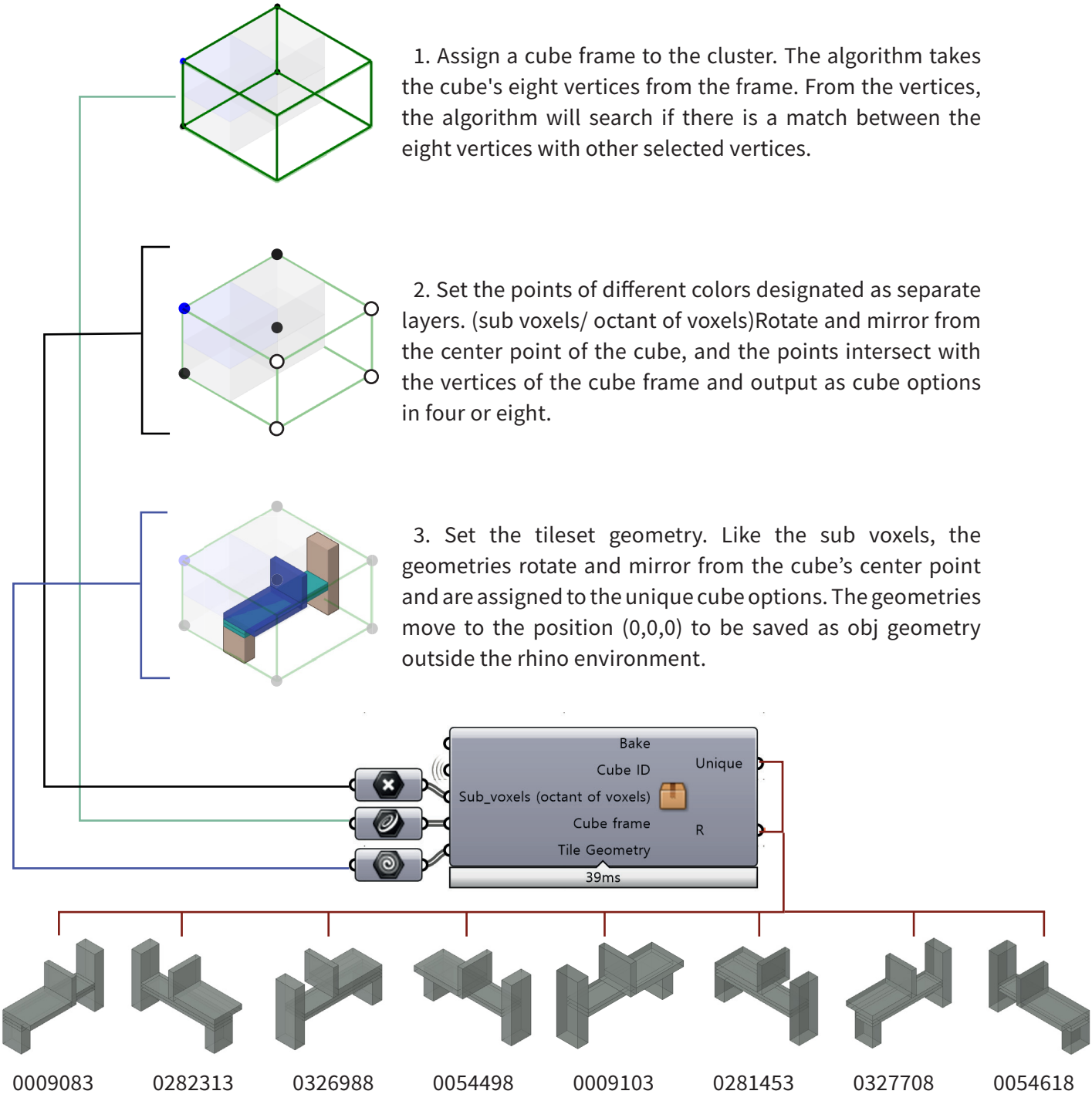
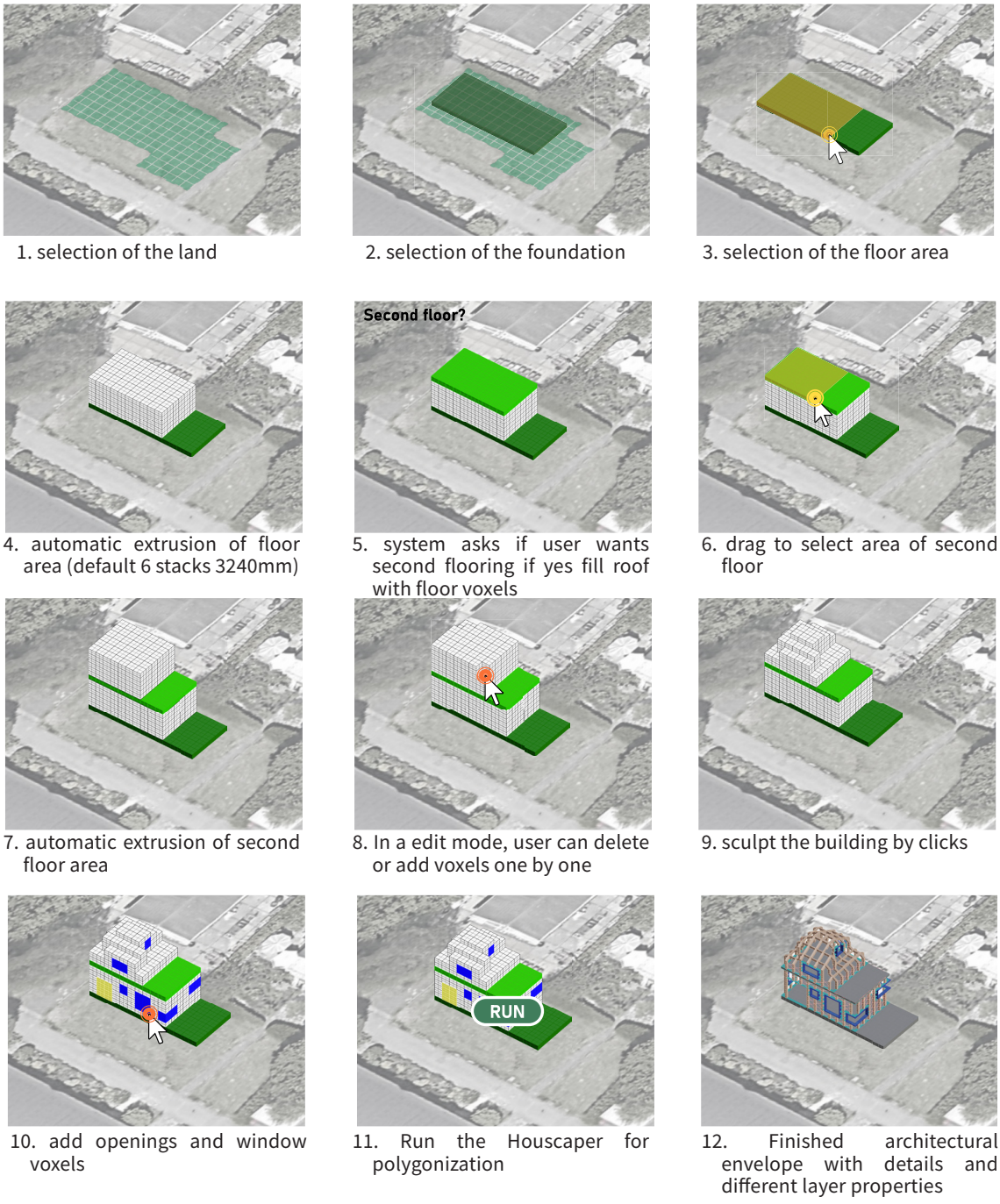


Fig.79: Tileset design rules (Author)

3.5.2 User case scenario

It is essential to address users to achieve their voxel design while keeping the design rules. Using the participatory design method or stacking the voxels using the design rules can generate endless combinations of voxel arrays. In other words, those endless combinations can end in combinations that are not so realistic. Figure 79 shows the procedural process of the use case scenario of Houscaper and methods developed to address users to design under the ruleset.



1. selection of the land
2. selection of the foundation
3. selection of the floor area
4. automatic extrusion of floor area (default 6 stacks 3240mm)
5. system asks if user wants second flooring if yes fill roof with floor voxels
6. drag to select area of second floor
7. automatic extrusion of second floor area
8. In a edit mode, user can delete or add voxels one by one
9. sculpt the building by clicks
10. add openings and window voxels
11. Run the Houscaper for polygonization
12. Finished architectural envelope with details and different layer properties

Fig.80: User case scenario procedure (Author)

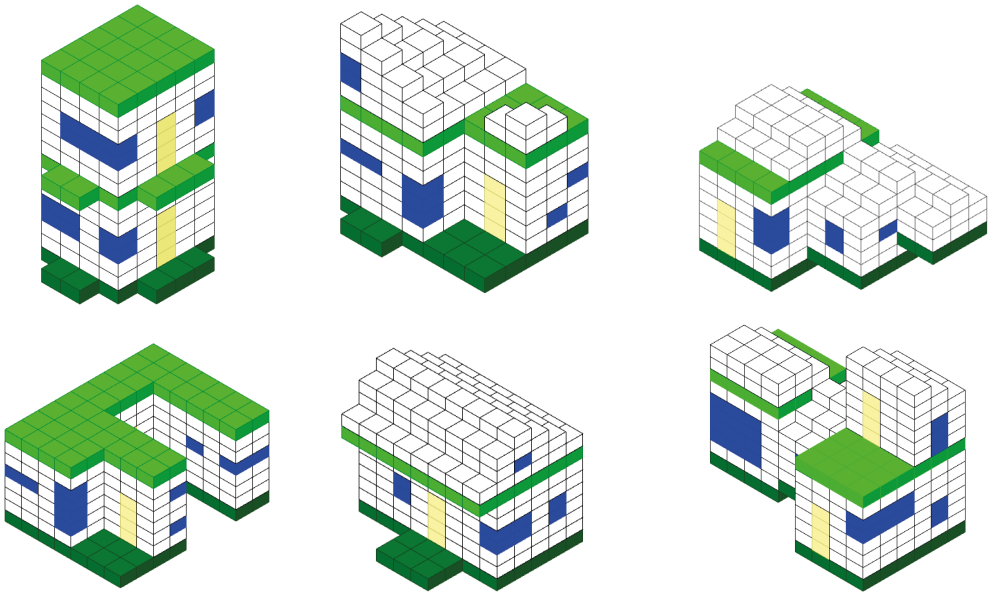
Chapter 4

4 Results

4.1 Architectural envelope

One of the goals of the house caper is to provide non-expert users with generating a modular architecture envelope with simple clicks to offer architectural design freedom within a modular design framework. The algorithm and user case scenario shows the possibility of the process. The following figures show some voxel designs resulting in an architectural envelope under the Houscaper algorithm.

Input voxel arrays



Output architectural envelope

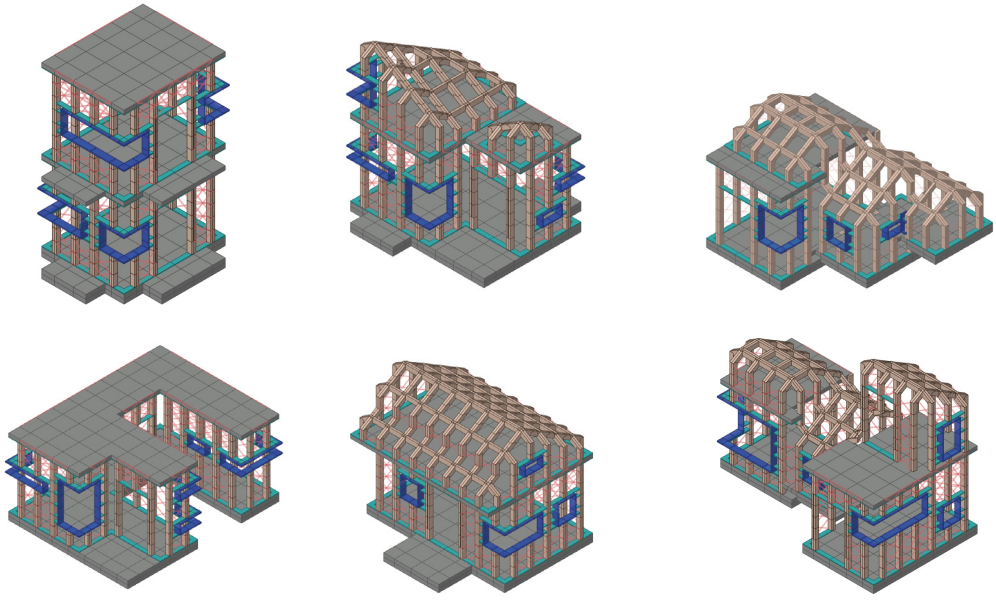


Fig.81: Input and output envelope geometry (Author)

4.2 Structural valididty check

4.2.1 Structural envelope

The structural validity check of the generated envelope is an essential aspect of architecture and a key saving point for the users. The Houscaper offers a structural analysis of the envelope combining the structure and skin system of the Open-building concept. The building type and size are limited to only the small and medium-sized self-standing envelope buildings. (The interior systems and zoning configurations can be applied as another layer of voxel array. such generative design method can be developed under the Go_Design framework.)

There are multiple reasons why Houscaper runs structural analysis on the architectural envelopes. First, there is only one type of voxel array applied on Houscaper, generating a single envelope without a structural system inside. (Adding more voxel configurations such as interior and structural voxel arrays will further increase the possibility of the system enabling more extensive architecture to undergo the structural analysis.) Second, without interior load-bearing walls, the system offers reconfigurability and brings more design options to the interior for the users to freely configure themselves. Third, the material, cost, and embodied energy are reduced using the exterior load-bearing systems, which Impacts less on the environment and increases efficiency over the interior load-bearing systems.

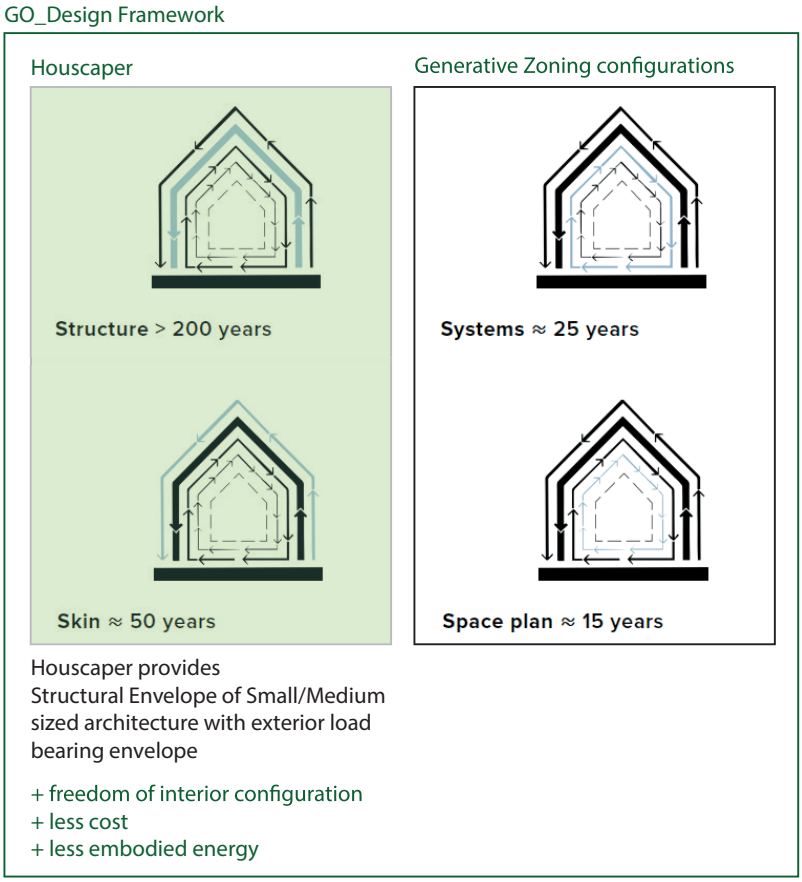
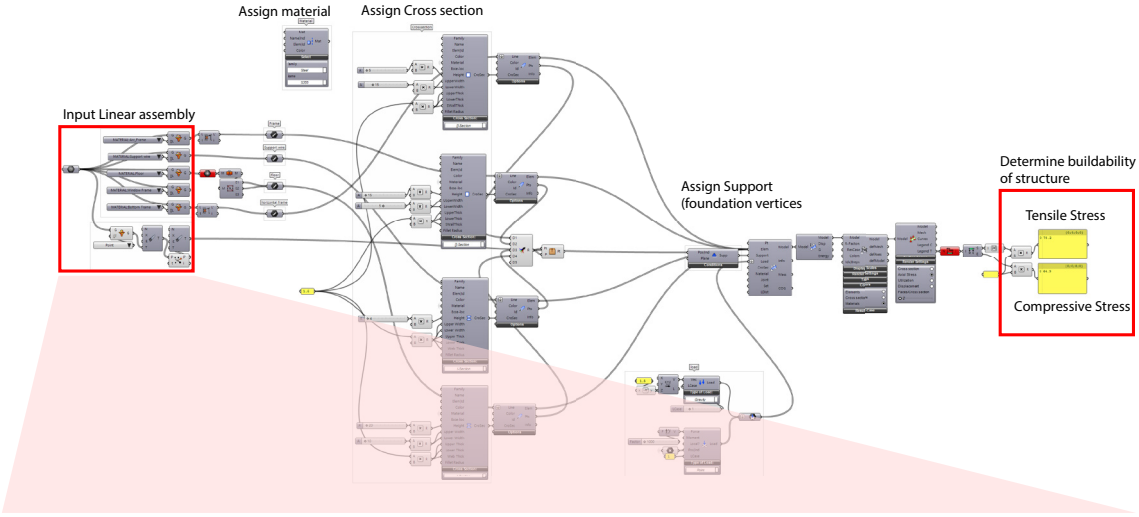


Fig.82: Structural envelope analysis in Houscaper(Author)

4.2.2 Structural analysis script

For structural analysis, the Houscaper requires another tileset of linear elements. The linear tileset is applied in the same method as architectural envelope generation with different tilesets appendix 9. The generated line structure runs in a Karamba script to check on the structural validity of the shape.



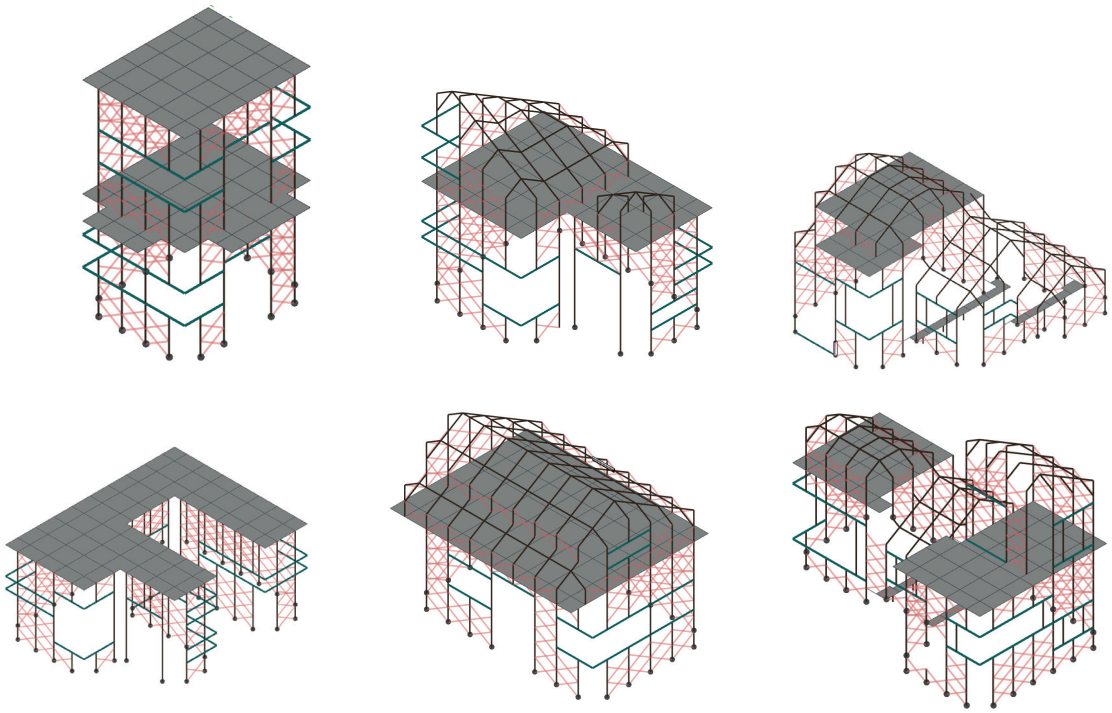
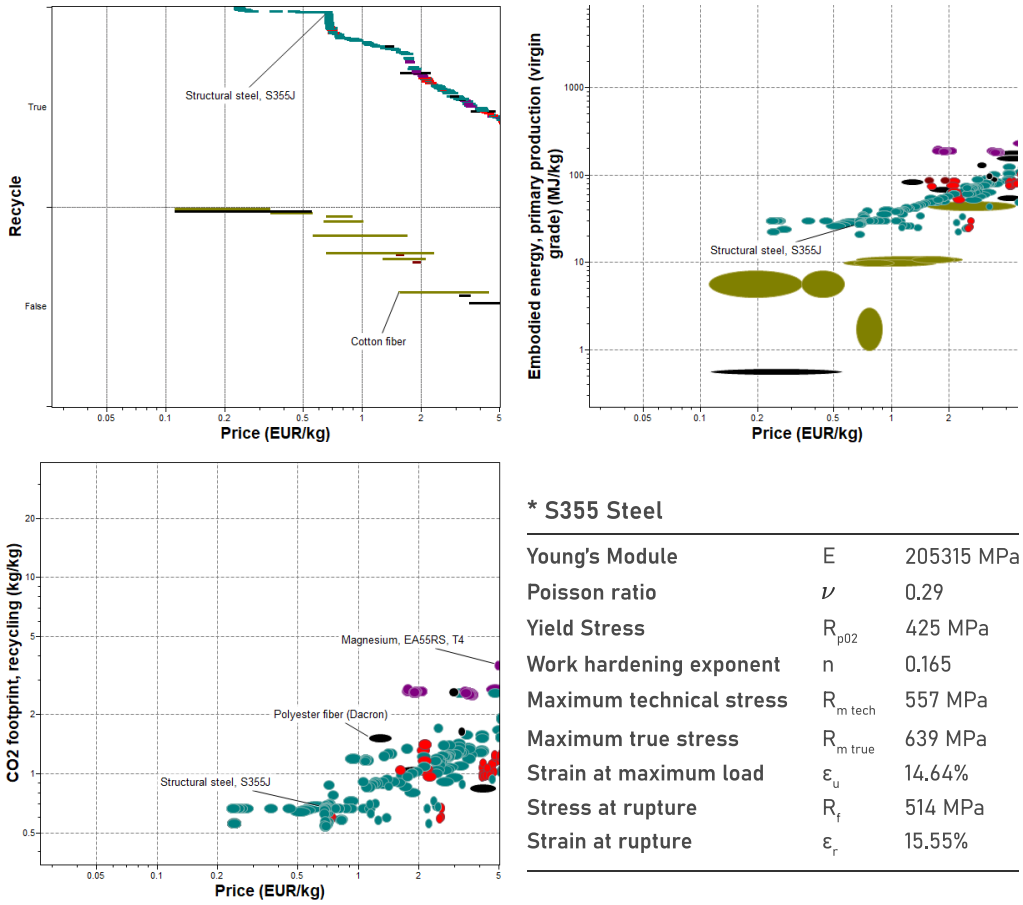


Fig.83: Overview Structural analysis script in Houscaper and input linear geometry (Author)

Material properties - Karamba enables users to assign properties of material and crosssections of the elements. In the script, layers of generated geometry divide the crosssections. The S355, non-alloy European standard structural steel, is used in the example structural analysis. The steel is commonly used after S235, where more strength is needed. The structural properties of S355 show the tensile strength of S355 is 470 ~680MPa. Below are graphs of S355 compared to other materials in Recyclability, Embodied energy for primary production, and CO2 footprint recycling.



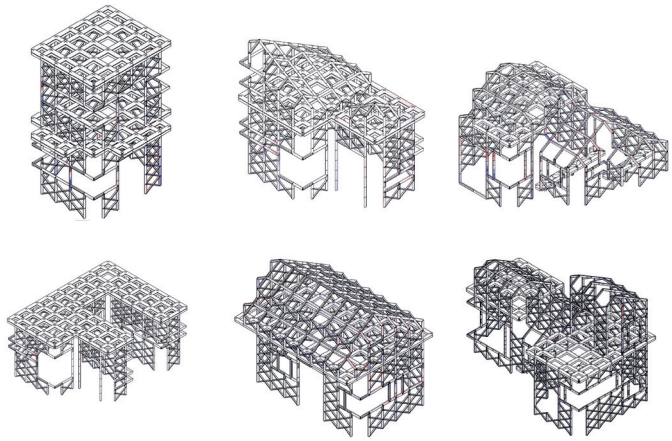
*** S355 Steel**

Young's Module	E	205315 MPa
Poisson ratio	ν	0.29
Yield Stress	R_{p02}	425 MPa
Work hardening exponent	n	0.165
Maximum technical stress	$R_{m\ tech}$	557 MPa
Maximum true stress	$R_{m\ true}$	639 MPa
Strain at maximum load	ε_u	14.64%
Stress at rupture	R_f	514 MPa
Strain at rupture	ε_r	15.55%

Fig.84: material properties of S355 (Author)

In general, steel frame houses are lighter, more durable, and more cost-effective to assemble than building with other materials such as timber. However, the steel frame shows relatively poor insulation and low energy efficiency. It is essential to understand that the Houscaper does not only work with steel frame houses and can select other linear materials such as wood or CLT to perform structural analysis.

In the script, expert users can assign different cross-sections on the beam in separate layers. Below are the cross-section properties set for regular steel frames.

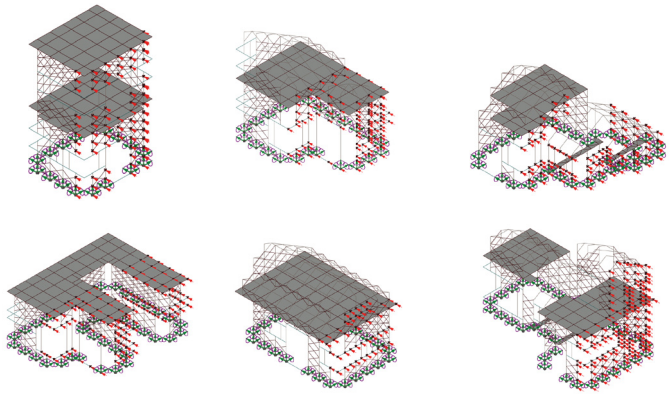


Cross-sections

Vertical Frame	Rectangular section	Height 150 cm Width 50 cm Thickness 1.6 mm
Horizontal Frame	Rectangular section	Height 50mm Width 150 mm Thickness 1.6 mm
Support Frame	Rectangular section	Height 40 mm Width 16 mm Thickness 1.6 mm
Floor Frame	I section	Height 200mm Upper width 100 mm Lower width 100 mm Thickness 1.6 mm

Fig.85: Cross-sections applied for frames (Author)

For gravity and wind loads applied to the structure, expert users can control more detailed levels of using loads. The script internally applies a standard load of 1.5 gravity load and 1000N/m for wind load on one side of a building.



Load Applied

1.5 G Gravity load
1000N/m Windload on one side

Fig.86: load case applied on linear geomgetry (Author)

As a general rule of beam structural analysis, the tensile and compressive stress of the beam defines the structural validity of the beam structure. The structure's max compressive and tensile stress should be placed under half of the material's compressive and tensile stress. These structural validity checks can be processed as an algorithm check of yes or no, enabling results to be delivered clearly and fast. The figure shows a pseudo code of the algorithm.

Input : structure's axial stress
Output: Validity of the structure

If structure's **compressive, tensile stress** is less than
  ½ of the material's **compressive stress** (278 n/mm2)
  ½ of the material's **tensile stress** (278 n/mm2),
  the structure is valid.
Else
  the structure is not valid.

Fig.87: Pseudo code for the structural analysis to determine buildability

As applied in the algorithm, the structural analysis results in a simple yes or no for the architecture to be buildable or not. Expert users can make variations in material and crosssections.

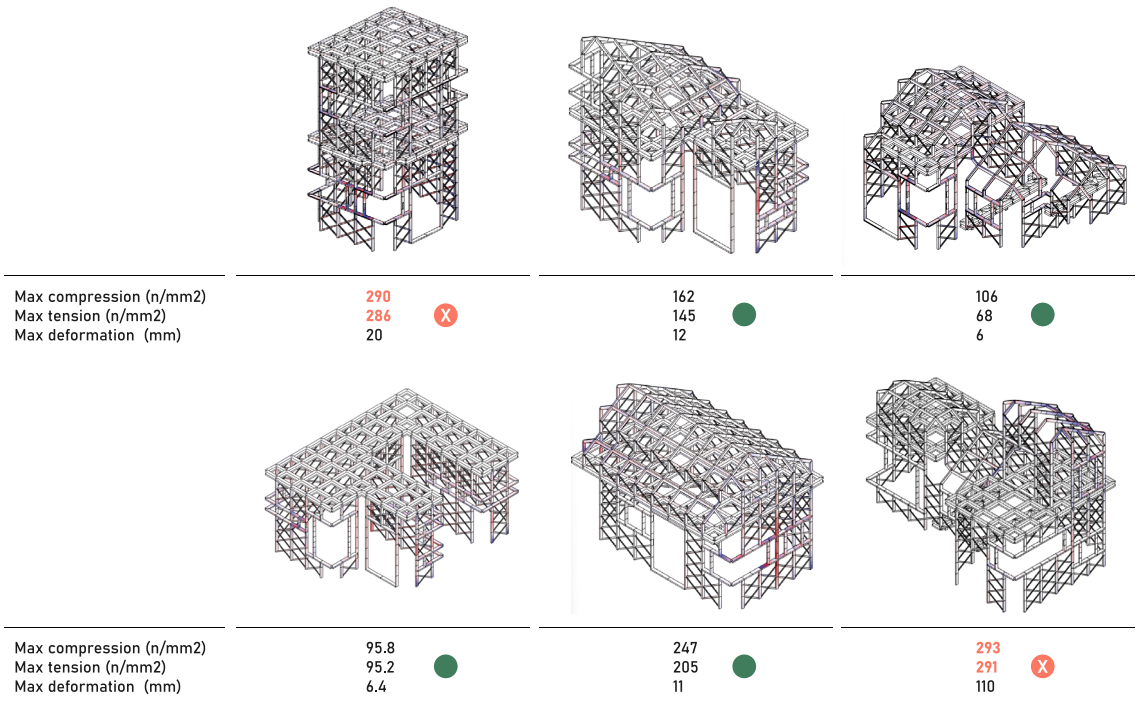


Fig.88: Result of Structural analysis indicating the stresses in red and blue colour(Author)

Chapter 5

5. Conclusion

Conclusion

The thesis presents a participatory design method of Modular Construction combined with the concept of 'assisted sculpting' using the BMC algorithm, enabling the non-expert users to design and check the validity of the structure. The modular construction has been considered a duplicate architecture that the end-users are not considered in the design process. The thesis counters the underlying perception and aims to bring Mass customization to the participatory design process, proving that modular architecture is not a connection of components but merged components based on modular digital parts. The tool generates polygonization of the voxel array using the tileset. As all parts are completely modular, the finished geometry can be precisely calculated with the CPQ (configure, price, quotation) of the material used for the frame generated.

Houscaper is a participatory design tool using a modification of the BMC algorithm. Instead of using Boolean (0 and 1), the tool uses multiple options depending on the number of voxel types. Each of these multiple options refers to a unique type of voxel in the configuration. Given the combinatorial nature of the discrete configurations, the number of possible configurations is astronomical, and designing all of them is inefficient and time-consuming. By using a set of design rules in placing the voxels, users can work with a reasonable number of tilesets, as suggested in the thesis. To limit the number of elements in each of these tilesets, the suggested method exploits various symmetries in architectural settings. More specifically, the modules and cube configurations are designed based on the 'equivalent up to' horizontal geometries (rotation and mirroring) Total of 493 configurations from 109 architectural tilesets are reduced to 72 under the design rule of the possible placement of voxel arrays.

As a guide to designing tilesets, the thesis offers three different kinds of tilesets, two surface tilesets and the architectural tileset. Expert users can work in a 3D modeling environment based on these template tilesets to produce more tilesets. The expert users can modify the cube configurations to add more options of tilesets based on the rule written and test their tilesets on the Houscaper.

Under the assumption the building owns a structural envelope, The tool can visualize the finished architectural frame and run a structural analysis, providing an instant validity check of the structure. The structural analysis script allows alternation of material and crosssections of linear building elements providing expert users with freedom of choice in materiality.

Discussion

As a nature of modular construction, the use of material exceeds the standard construction method. The need for a universal connection of components increases the manufacturing cost and material. However, this limitation can be solved as the modular construction method achieves the economy of scale.

limitation

The research has mainly focused on the polygonization aspect of modular housing using a voxel array. The modeling of the parts remains basic to give users a simple visual impression of the tool, but with more advanced modeling the tool is capable of professional-level performance.

The structural analysis also works with linear modular components by assigning crosssections and material properties in FEM analysis. However, site property variations and specifications are ignored to bring more generality to the project.

Other methods of generating cube configurations were researched, but the generality and predictability are valued more and proceeded with the current method.

future works

Future work concerns further experimentation with the proposed workflow.

1. A web version, a lighter version of the tool designed for the mass as a publishable SaaS (software as a Service), enables public access to the device for both the mass and the experts. With more people's access to the tool, the Houscaper can become a platform to share their ideas and tilesets to bring possibilities of modular construction further.

2. Development of a generative design method for the voxel array design. Already optimization methods introduced in GoDeisgn show possibilities to generative design the zone configuration. Under the GoDesign framework, the envelope voxel generating method can be developed and applied to automatize the Houscaper further.

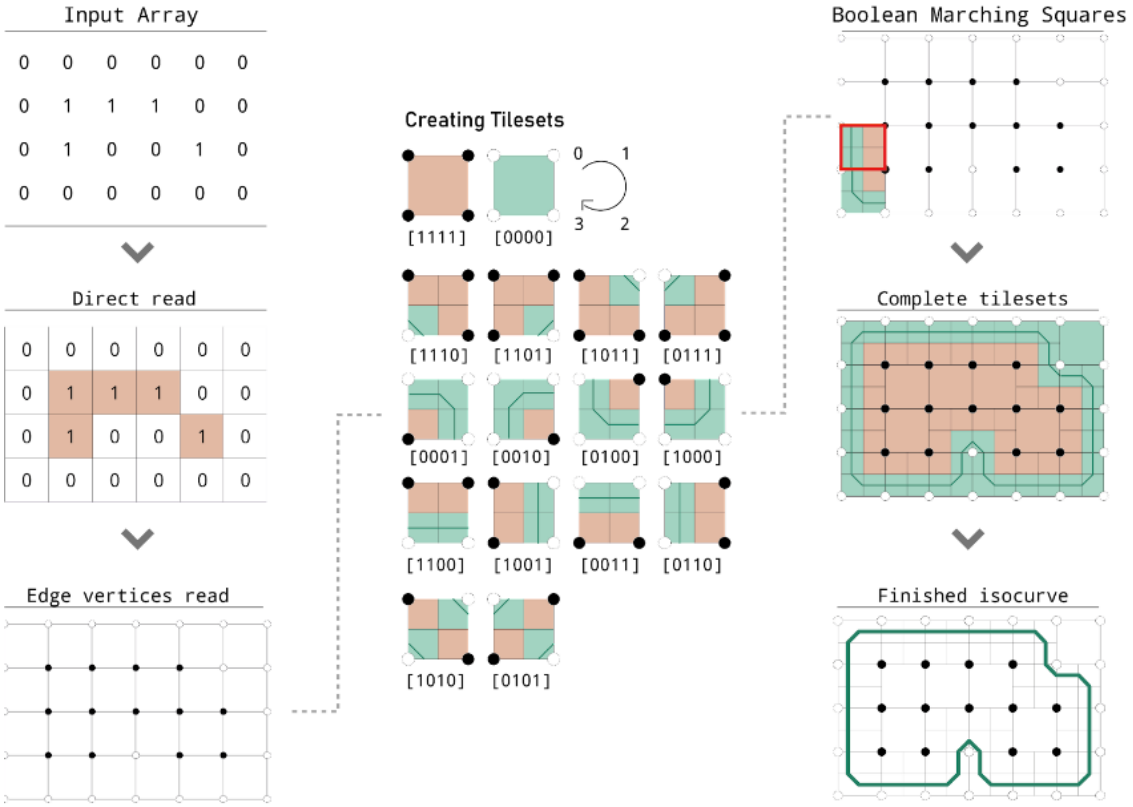
3. Develop a script enabling the parts outside of Module sizes, such as the truss inside the wall and roof. The trusses connecting edge-to-edge point of frame sizes often exceed the modules' size. In the thesis, the trusses fit in the size of modules (which is sufficient for structural validity), but this increases material usage. Further research is to develop the algorithm for a better way to generate trusses exceeding the module size.

Appendix

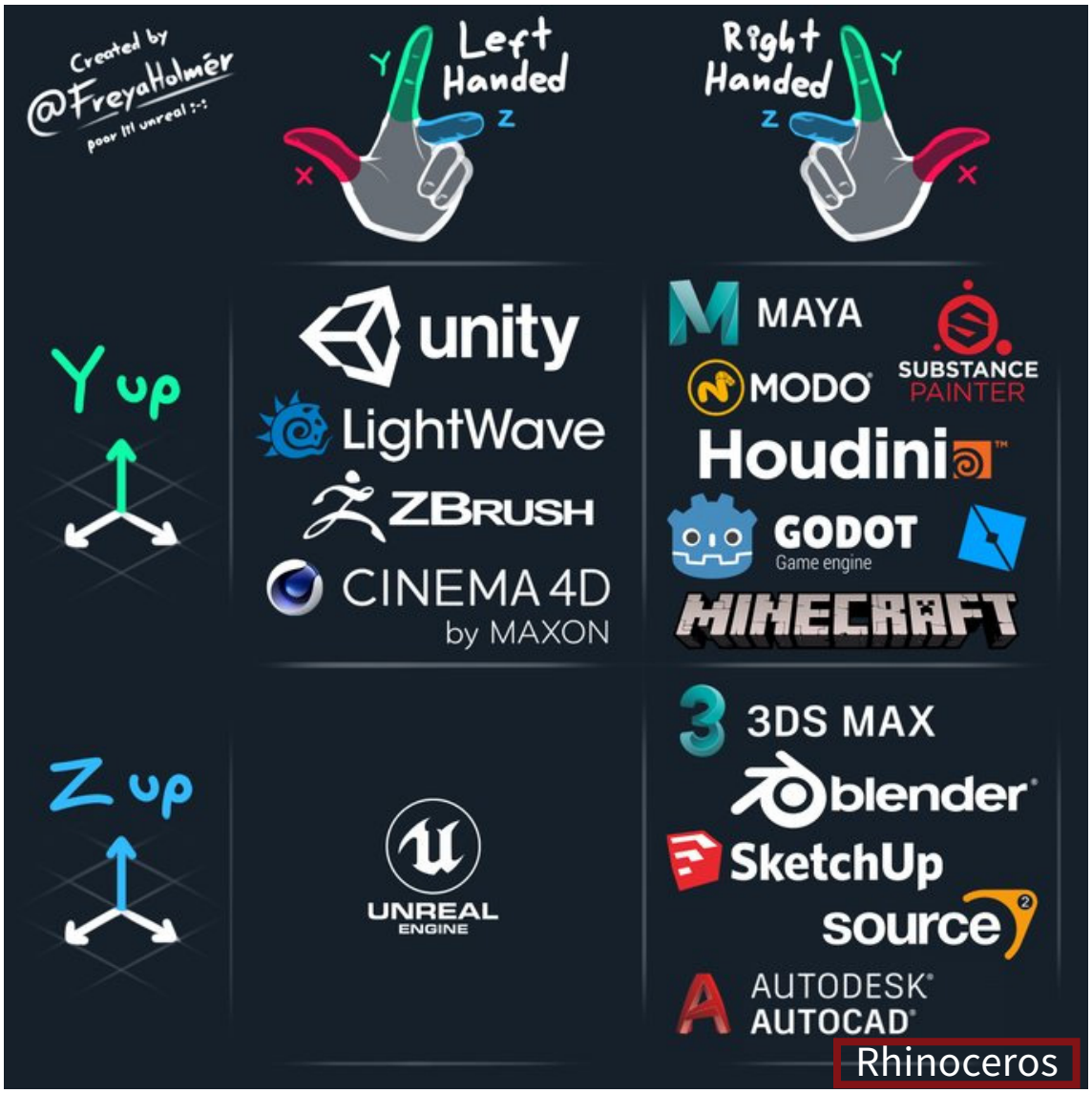
Boolean Marching square algorithm

The BMS (Boolean Marching Square) algorithm is an algorithm that generates a contour line for a two-dimensional rectangular array or a scalar field. The BMS algorithm is addressed to understand the BMC algorithm used throughout the thesis. Instead of the three-dimensional voxel array input of BMC, the BMS algorithm takes the two-dimensional pixel array as an input and outputs a single contour isocurve. The basic procedures of the algorithm explained in the diagram followed.

1. A two-dimensional rectangular boolean array is given as an input.
2. Direct reading from the input results in four pixels.
3. The boolean array of the four edge points gives a unique id for each square thus creating tilesets (Filled points = 1 and empty points = 0).
4. Boolean marching squares Iterating on all squares in the grid, the square marches, placing matching tiles from the tileset of the unique square id configurations. The possible tile counts are 16 since two possible options of 4 points exist (2^4).
5. the process completes on all squares.
6. A finished contour isocurve is generated from the input boolean array



Appendix 2. Different programs using different coordinate systems by Freya Holmer



Appendix 3. Cube arry from voxel array

```
import ghpythonlib.components as gc
import ghpythonlib.treehelpers as th
import rhinoscriptsyntax as rs
import Rhino.Geometry as rg

# get centerpoints of boxes
centerpoints = []
for i in voxel:
    centerpoints.append(rg.AreaMassProperties.Compute(i).Centroid)

oneb = rs.BoundingBox(voxel[0])
onebox = rg.Box(rs.WorldXYPlane(),oneb)

box = rs.BoundingBox(voxel)
bbox = rg.Box(rs.WorldXYPlane(),box)

debox= gc.DeconstructBox(bbox)
deboxOne = gc.DeconstructBox(onebox)

ox = deboxOne[1]
oy = deboxOne[2]
oz = deboxOne[3]

x = debox[1]
y = debox[2]
z = debox[3]

xx = (x[1]-x[0])/(ox[1]-ox[0]) +1
yy = (y[1]-y[0])/(oy[1]-oy[0]) +1
zz = (z[1]-z[0])/(oz[1]-oz[0]) +1

# get vector to move cell to starting point
start = box[0]
cent = gc.Volume(onebox)['centroid']
vec = start-cent
moved = gc.Move(onebox,vec)

# create array of cube from modules
array = gc.BoxArray(moved[0],onebox,xx,yy,zz)
cube_array = array['geometry']

# from the array of cubes
# get all the corner points as tree
# get first vertices as first_vertices
allpt = []
first_vertices = []
for i in cube_array:
    e = gc.BoxCorners(i)
    allpt.append(e)
    first_vertices.append(e[0])


cube_vertices = th.list_to_tree(allpt)
```

Appendix 4.intersect the vertices with cube frame edge points to get cube id

```
import ghpythonlib.treehelpers as th
import rhinoscriptsyntax as rs
import ghpythonlib.components as gc

cube_vert = th.tree_to_list(cube_vertices,None)

def intersect(pts,id):
    list2 = []
    for i in cube_vert:
        a = gc.trees.MemberIndex(pts,i)
        newlist=th.tree_to_list(a['index'])
        list = []
        for j in newlist:
            if j == []:
                j = 0
                list.append(j)
            elif j != []:
                j = id
                list.append(j)
        list2.append(list)
    return (list2)

def byeight(list):
    result = [list[i * 8:(i + 1) * 8] for i in range((len(list) - 1 + 8) // 8 )]
    result_ = th.list_to_tree(result)
    return result_

base = intersect(base_ctp,1)
fl= intersect(fl_ctp,2)
fo = intersect(fo_ctp,3)
op = intersect(op_ctp,4)
wi = intersect(wi_ctp,5)

base_ = byeight(base)
floor_= byeight(fl)
found_ = byeight(fo)
wi_ = byeight(wi)
op_ = byeight(op)
```

Appendix 5.converting heximal to decimal

```
import rhinoscriptsyntax as rs
import ghpythonlib.treehelpers as th

def Hextodec(lst):
    result=[]
    for i in lst:
        d = ''.join(map(str,i))
        e = int(d,6)
        result.append(e)
    return result

Cubelist= th.tree_to_list(Cube,None)
Cubetobin = Hextodec(Cubelist)
th.list_to_tree(Cubetobin)
Cubelist = th.list_to_tree(Cubelist)
Cubelist = [str(i).zfill(7) for i in Cubetobin]
```

Appendix 6.Rotate mirror vertices to get cube options options

```
import rhinoscriptsyntax as rs
import ghpythonlib.treehelpers as th
import ghpythonlib.components as gc
import Rhino.Geometry as rg
import math
# get centroid of bounding box
CP = gc.Volume(frame)['centroid']
vec = rs.CreateVector(0,0,90)
# make plane for XYZ plane
x = rg.Point3d(0.01,0,0)
y = rg.Point3d(0,0.01,0)
z = rg.Point3d(0,0,0.01)
# make XYZ oriented from centerpoint
newx = CP+x
newy = CP+y
newz = CP+z

## move to top 8 times
def move(obj):
    result=[]
    for i in range(8):
        a = gc.Move(obj,vec*i)
        result.append(a[0])
    return result

# rotate_mirror_rotate
def mirrorrotatemove(obj):
    result = []
    for i in range(4):
        a = rs.RotateObjects(obj,CP,90*i,None,True)
        b = rs.MoveObjects(a,vec*i)
        result.append(b)
    mir = rs.MirrorObjects(obj,newy,newz,False)
    for i in range(4):
        d = rs.RotateObjects(mir,CP,90*i,None,True)
        e = rs.MoveObjects(d,vec*(i+4))
        result.append(e)
    return result

base_id = th.list_to_tree(mirrorrotatemove(base))
floor_id = th.list_to_tree(mirrorrotatemove(floor))
found_id = th.list_to_tree(mirrorrotatemove(found))
open_id = th.list_to_tree(mirrorrotatemove(opening))
window_id = th.list_to_tree(mirrorrotatemove(window))

RF = move(frame)
RF2 = th.list_to_tree(RF)

allpts=[]
startpt1= []
for i in RF:
    e = gc.BoxCorners(i)
    allpts.append(e)
    startpt1.append(e[0])

tree = th.list_to_tree(allpts)
```

Appendix 7.Move geometry to origin point and rotate mirror to get geometry options

```
import rhinoscriptsyntax as rs
import ghpythonlib.components as gc
import ghpythonlib.treehelpers as th
import Rhino.Geometry as rg
import math

# move obj to origin[0,0,0]
origin_point = rg.Point3d(0,0,0)
cube_cornerpoint = gc.BoxCorners(frame)[0]
vec_o_to_c= rs.VectorCreate(origin_point,cube_cornerpoint)
obj_ol=gc.Move(GroupedObj,vec_o_to_c)[0]
cube_cp = gc.Move(frame,vec_o_to_c)
# get centerpoint of the moved frame
CP = gc.Volume(cube_cp['geometry'])['centroid']

# make plane for XYZ plane
x = rg.Point3d(0.01,0,0)
y = rg.Point3d(0,0.01,0)
z = rg.Point3d(0,0,0.01)
# make XYZ oriented from centerpoint
newx = CP+x
newy = CP+y
newz = CP+z

plane = rs.PlaneFromPoints(CP,newx,newy)

def mirrorrotatemove(obj):
    result = []
    for i in range(4):
        a = rs.RotateObjects(obj,CP,90*i,None,True)
        result.append(a)
    mir = rs.MirrorObjects(obj,newy,newz,False)
    for i in range(4):
        d = rs.RotateObjects(mir,CP,90*i,None,True)
        result.append(d)
    return result

a =mirrorrotatemove(obj_ol)
a = th.list_to_tree(a)

print a
```

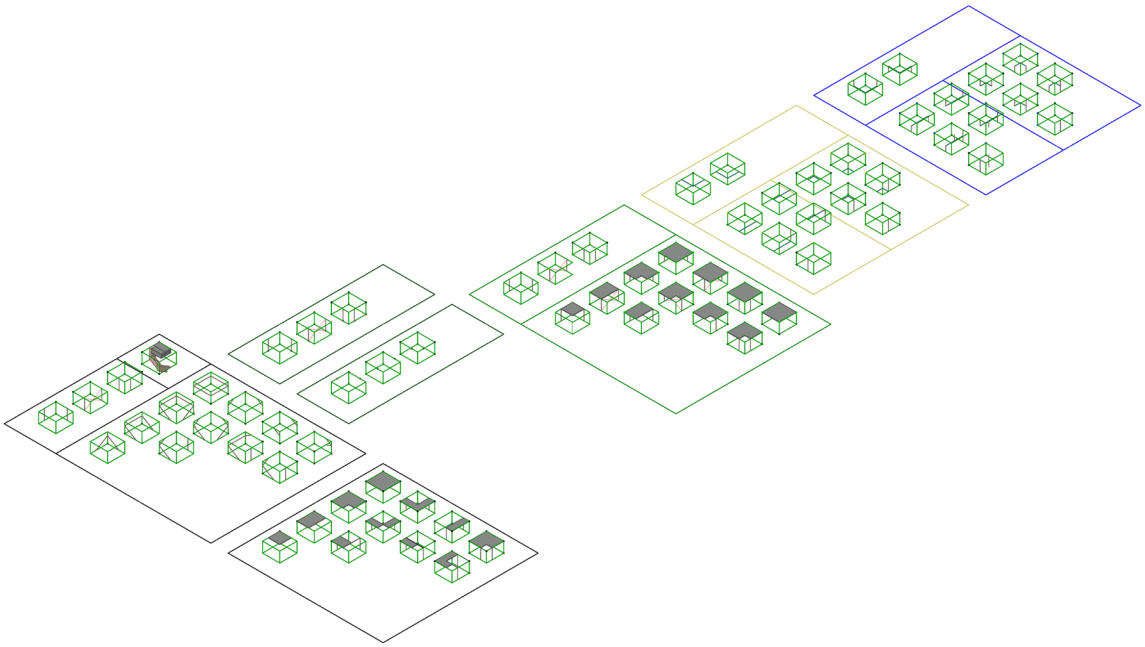
Appendix 8.intersect check on Cube and Cube opt

```
import rhinoscriptsyntax as rs
import ghpythonlib.treehelpers as th
from itertools import chain

CubeOpt = th.tree_to_list(CubeOpt,None)
obj = th.tree_to_list(geometry,None)
list = []
for i in CubeOpt[0]:
    if i == Cube[0]:
        list.append(CubeOpt[0].index(Cube[0]))
num = list[0]

a = obj[num]
```

Appendix 9. Linear tileset



list of figures

References

[1] S. Azadi and P. Nourian, "A modular generative design framework for mass-customization and optimization in architectural design," p. 11.

[2] "Modular Construction Market Size & Share Report, 2028." https://www.grandviewresearch.com/industry-analysis/modular-construction-market (accessed Mar. 31, 2022).

[3] "Materials Practice Guide Modular Construction - DESIGN FOR MODULAR CONSTRUCTION: AN INTRODUCTION FOR," StuDocu. https://www.studocu.com/en-us/document/florida-international-university/contract-and-project-management/materials-practice-guide-modular-construction/7301075 (accessed Mar. 30, 2022).

[4] "Spotlight Modern Methods of Construction." Savils. [Online]. Available: https://pdf.euro.savills.co.uk/uk/spotlight-on/spotlight-modern-methods-of-construction-spring-2020.pdf

[5] "Maximizing Modular Fabrication," Modular Building Institute, Jan. 01, 2021. https://www.modular.org/2021/01/01/maximizing-modular-fabrication/ (accessed Mar. 17, 2022).

[6] "A new edition of 'Delivery Platforms for Government Assets,'" Bryden Wood. https://www.brydenwood.co.uk/projects/a-new-edition-of-delivery-platforms-for-government-assets/s115366/ (accessed Apr. 05, 2022).

[7] R. Doe, "Integration of computational design tools in the design and production of prefabricated homes," Thesis, 2017. Accessed: Mar. 18, 2022. [Online]. Available: https://ses.library.usyd.edu.au/handle/2123/18768

[8] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87, Not Known, 1987, pp. 163–169. doi: 10.1145/37401.37422.

[9] "Solving the global housing crisis." https://www.worldfinance.com/infrastructure-investment/solving-the-global-housing-crisis (accessed Apr. 05, 2022).

[10] "Construction Costs Will Keep Rising. Here's How Much.," Commercial Property Executive, Oct. 07, 2021. https://www.commercialsearch.com/news/construction-costs-will-keep-rising-heres-how-much/ (accessed Apr. 18, 2022).

[11] "The Digital in Architecture: Then, Now and in the Future," SPACE10, Nov. 12, 2019. https://space10.com/project/digital-in-architecture/ (accessed Mar. 18, 2022).

[12] "Manifesto — Open Building." https://www.openbuilding.co/manifesto (accessed Jun. 15, 2022).

[13] S. Oliveira, J. Burch, K. Hutchison, Olalekan Adekola, S. Jaradat, and M. Jones, "MMC modes of delivery in housing: Effects on Housing Association clients," 2017, doi: 10.13140/RG.2.2.20524.21127.

[14] "A Brief History on Modular Architecture," GKV Architects. https://www.gkvarchitects.com/news/a-brief-history-on-modular-architecture (accessed Mar. 19, 2022).

[15] L. Solonickne, "Why Did Sears Stop Selling Houses?" http://www.sears-homes.com/2016/07/why-did-sears-stop-selling-houses_11.html (accessed Apr. 19, 2022).

[16] "Building the 'Secret Cities' | National Trust for Historic Preservation." https://savingplaces.org/stories/building-the-secret-cities-pre-fab-architecture-of-the-manhattan-project (accessed Apr. 19, 2022).

[17] de H. T.M, "Developing a computational method for including a client's input in a mass customizable 3D module building industry," Thesis, TU Delft.

[18] "Finch Buildings - Product," Finch Buildings. https://finchbuildings.com/en/product/ (accessed Apr. 18, 2022).

[19] "BONE Structure - Steel Construction System for Net-Zero Energy Ready Homes," BONE Structure. https://bonestructure.ca/en/home/ (accessed Apr. 19, 2022).

[20] “Passive House,” GO LOGIC. https://www.gologic.us/passive-house (accessed May 19, 2022).

[21] “Gablok Nederland modulaire energieneutraal bouwsysteem - Gablok Nederland BV.” https://www.gablok-nederland.nl/ (accessed May 14, 2022).

[22] “Diensten - Sustainer Homes.” https://sustainer.nl/diensten/ (accessed Apr. 19, 2022).

[23] T. Alphin, author of the bestselling book T. L. Architect, and which explores how to build 7 styles of architecture using L. T. page shows the calculations behind my article in issue #13 of B. Magazine, “LEGO figures in Scale models,” BRICK ARCHITECT. https://brickarchitect.com/scale/ (accessed Jun. 24, 2022).

[24] P. Nourian, S. Azadi, and F. van Andel, “Open Source Participatory Design and Construction of Open Buildings: Affordable ‘Haute Couture’ for the Masses by means of Design-Build Games:,” Mar. 16, 2021. doi: 10.13140/RG.2.2.30315.46888.

[25] “Getting started with advanced LEGO building,” Swooshable. https://swooshable.com/how-to-start-building-lego (accessed May 10, 2022).

[26] B. Durhuus and S. Eilers, “On the entropy of LEGO ®,” J. Appl. Math. Comput., vol. 45, no. 1–2, pp. 433–448, Jun. 2014, doi: 10.1007/s12190-013-0730-9.

[27] “All Models of 2x4 LEGO Bricks.” https://c-mt.dk/counting/ (accessed Jun. 06, 2022).

[28] “Mobaco,” » Museum Kinderwereld, Feb. 11, 2013. https://www.museumkinderwereld.nl/mobaco/ (accessed Apr. 19, 2022).

[29] “Townscaper,” Townscaper. https://www.townscapergame.com (accessed May 12, 2022).

[30] “Marching cubes.” http://www.scientificlib.com/en/ComputerGraphics/MarchingCubes.html (accessed May 10, 2022).

[31] “Generative Design with the Wave Function Collapse Algorithm,” welcometoabode.com, Oct. 03, 2021. https://welcometoabode.com/generative-design-with-the-wave-function-collapse-algorithm/ (accessed May 13, 2022).

[32] A. Savov, R. Winkler, and O. Tessmann, “Encoding Architectural Designs as Iso-surface Tilesets for Participatory Sculpting of Massing Models,” 2020, pp. 199–213. doi: 10.1007/978-3-030-29829-6_16.

[33] Boris, “Marching Cubes 3d Tutorial,” BorisTheBrave.Com, Apr. 15, 2018. https://www.boristhebrave.com/2018/04/15/marching-cubes-3d-tutorial/ (accessed Jun. 06, 2022).

[34] J.-H. Hou, “Object Modeling and Proper Abstractions: The Case of Stair Design,” p. 280.

[35]“Expanding Wave Function Collapse with Growing Grids for Procedural Content Generation - Project Library, Aalborg University.” https://projekter.aau.dk/projekter/en/studentthesis/expanding-wave-function-collapse-with-growing-grids-for-procedural-content-generation(34c55797-b45e-42c3-bb1e-1f3865ec8308).html (accessed Jun. 28, 2022).

[36]E. Holmlid, “Manifold Contouring of an Adaptively Sampled Distance Field,” undefined, 2010, Accessed: Jun. 28, 2022. [Online]. Available: https://www.semanticscholar.org/paper/Manifold-Contouring-of-an-Adaptively-Sampled-Field-Holmlid/6f7041364cd8891daa778e4d7659354916a6890f