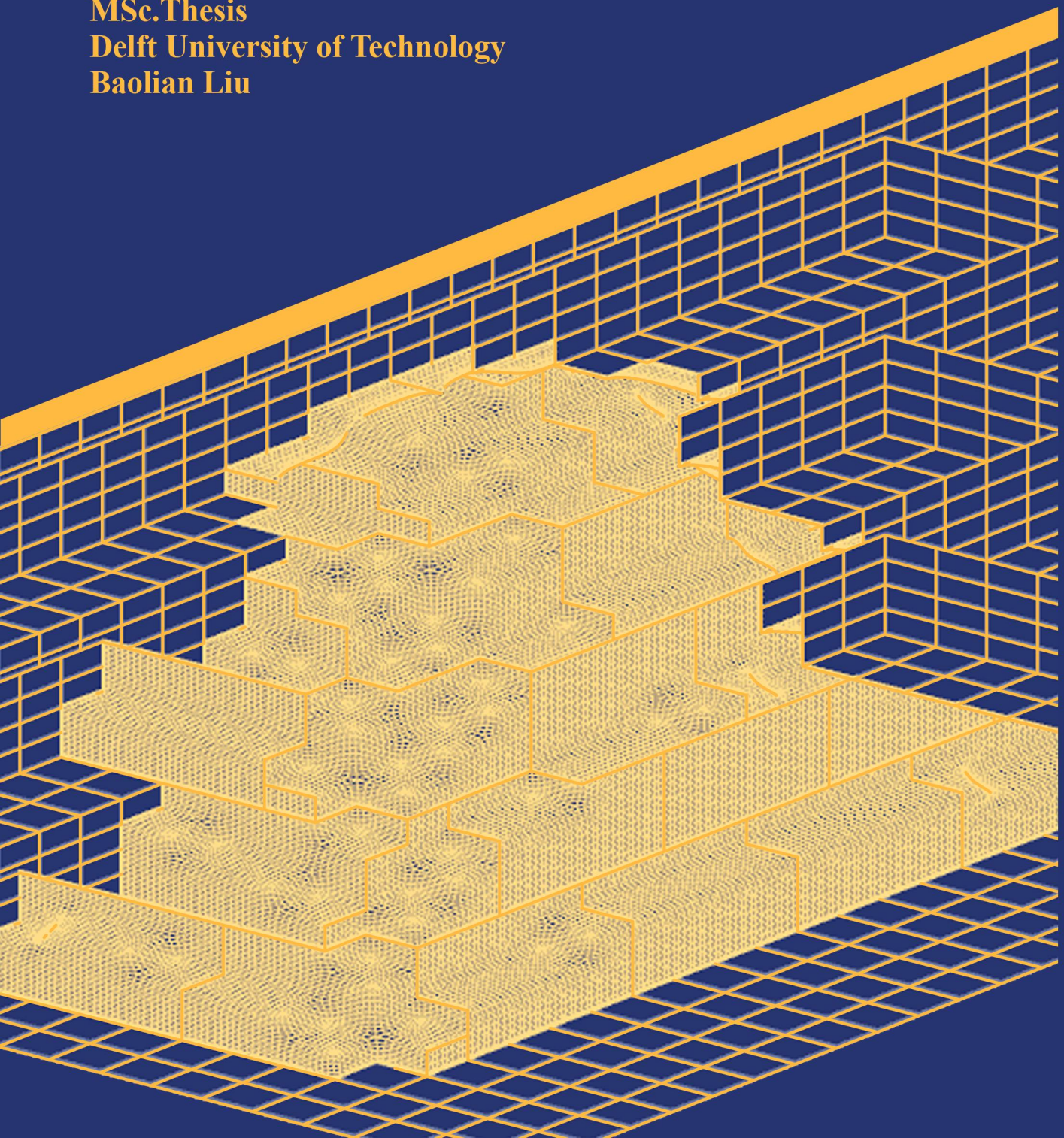
Topological Stereotomic Design of System of Interlocking Stackable Modular Blocks for Constructing Multi-Storey Masonry Buildings

MSc.Thesis
Delft University of Technology
Baolian Liu



Master Thesis

Topological Stereotomic Design of System of
Interlocking Stackable Modular Blocks for Constructing
Multi-Storey Masonry Buildings
01/07/2022

Student:
Baolian Liu | 5244404

Delft University of Technology
Faculty of Architecture and Built Environment
MSc. Architecture, Urbanism and Building Sciences
Track **Building Technology**

First Mentor:
Dr. Ir. Pirouz Nourian
Architectural Engineering +Technology | Design Infromatics

Second Mentor:
Dr. Ir. Simona Bianchi
Architectural Engineering +Technology | Structural Design & Mechanics

Advisor:
Dr. Ir. Anjali Mehrotra
Civil Engineering and Geosciences | Applied Mechanics

Ir. Shervin Azadi
Architectural Engineering +Technology | Design Infromatics

External Examiner:
Geert Coumans
Architecture | Form, Space & Type

Chapter 3

Space-filling Grid Volumetric Design

3.1 Introduction

As aforementioned, the design process is divided into two parts: Space-filling Grid Volumetric Design and Shell Structure Approximation Algorithm Design. This Chapter elaborates about the first part, to design a 3D interlocking grid system as a base grid for the next part of algorithm design.

3.2 Design Methodology of Space-filling Grids

In this Chapter, three types of space-filling grids are designed based on the concept of Topological Interlocking, a design principal mentioned in Chapter 2. Two of the grids are chosen for structural verifications in Abaqus [30] and 3DEC [31] respectively, with one of the grids selected as the most promising one for the subsequent algorithm design. It is worth mentioning that necessary structural verifications are conducted as auxiliary tools to compare the structural behaviour between the grids. An overview of the design methodology can be seen in Figure 3.1.

In cooperate with Qinglu Chen, the design of three types of grids are the cooperation results together as well as the preliminary structural analyses results of grid comparison in Abaqus. The rest part of the structural verification process, including grid comparison in 3DEC, proportion comparison, joint design and self-supporting comparison are finished by the author individually.

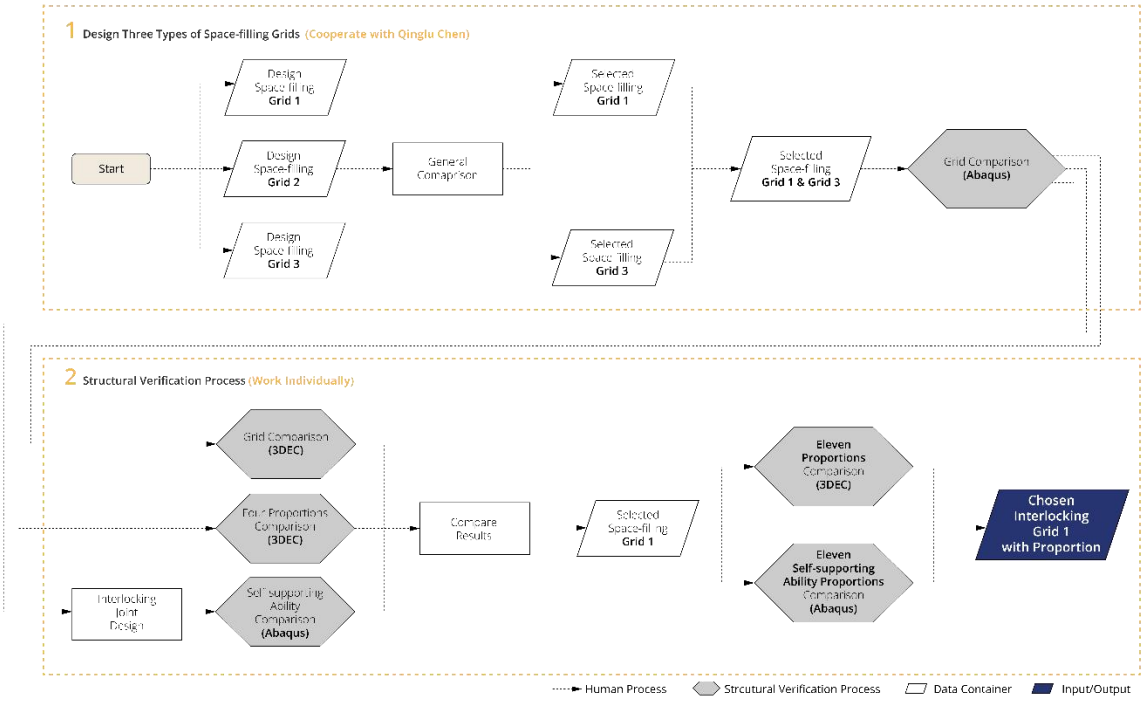


Figure 3.1 *Design Methodology of space-filling grids*

3.3 Design of Three Types of Space-filling Grids

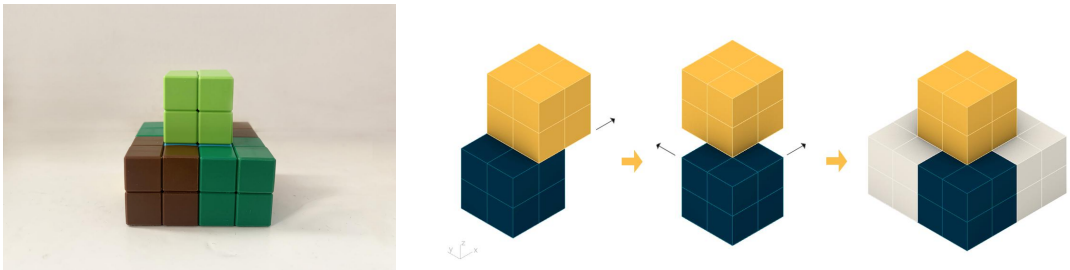
3.3.1 Overview of Three Types of Space-filling Grids

To ensure a space-filling volumetric grid, three types of grids are generated from a base Topological Interlocking Grid described above and developed from a smaller voxel grid (Figure 3.2). These grids represent the topological relationship of the modular blocks for the construction of the vaults. The design started with playing with magnetic voxels representing the smaller voxel grid of each block. By placing voxels in different positions, different layering methods and topology of modular blocks can be generated.

Figure 3.2 *Overview the three types of space-filling grids (Source from Qinglu Chen and Baolian Liu)*

3.3.2 Developments of Space-filling Grid 1

Grid 1 learns part from the base Topological Interlocking Grid described in Section 2.2 and also borrowed from the traditional brick layering method when the upper block stands on half of the lower block. By moving the upper block horizontally in the other direction, Grid 1 is generated based on the assumption that the upper block is supported by one-quarter of each of its lower blocks (Figure 3.3), therefore forming a space-filling system which ensures a three-dimensional dual graph like a pyramid of the force flow path. However, this grid results in inadequate interlocking between the elements.



(a) (b)

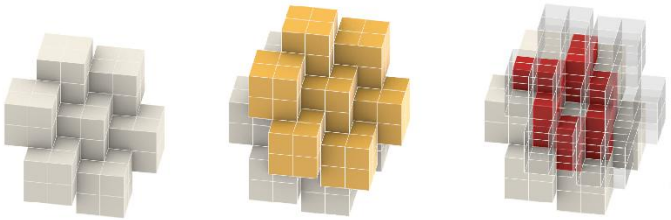
Figure 3.3 *(a) Space-filling Grid 1 (Source from Qinglu Chen and Baolian Liu) (b) Development of Grid 1 from the traditional brick layering method*

3.3.3 Developments of Space-filling Grid 2

Grid 2 (Figure 3.4(a)) starts from the base grid of the second type of Topological Interlocking Grid (Figure 3.4(b)), this type of interlocking can prevent the middle block from moving constraint by its six neighbours. However, it has the drawback of creating voids when a second layer is added (Figure 3.5).

(a) (b)

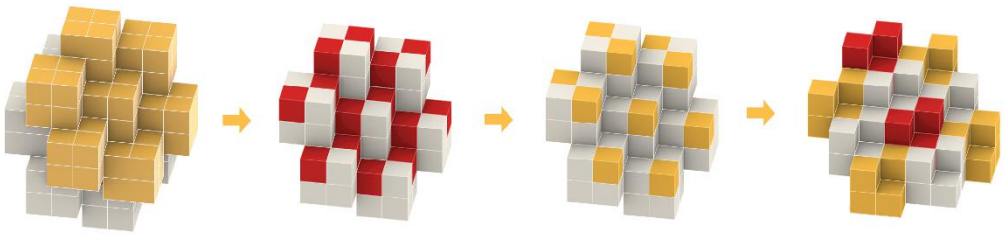
Figure 3.4 *(a) Space-filling Grid 2 (Source from Qinglu Chen and Baolian Liu) (b) Base Topological Interlocking Grid*



(a) (b) (c)

Figure 3.5 *(a) original interlocking grid (b) add a second layer on top (c) voids in between the layers (Source from Qinglu Chen and Baolian Liu)*

In order to solve the problem of the voids, several movements of small voxels within the blocks are conducted. First, the special parts highlighted in red colors are the parts need to be reserved, while the opposite parts highlighted in yellow within each block are the reason causing the gap, thus are determined to be removed. After removing the yellow voxels, the final tessellation is able to fulfill the space, then the topology of Grid 2 is thus determined (Figure 3.6).



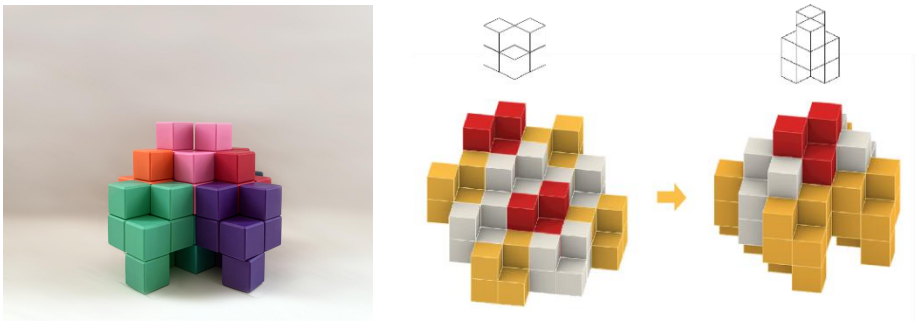
(a) (b) (c) (d)

Figure 3.6 *Design Process of space-filling Grid 2 (Source from Qinglu Chen and*

Baolian Liu)

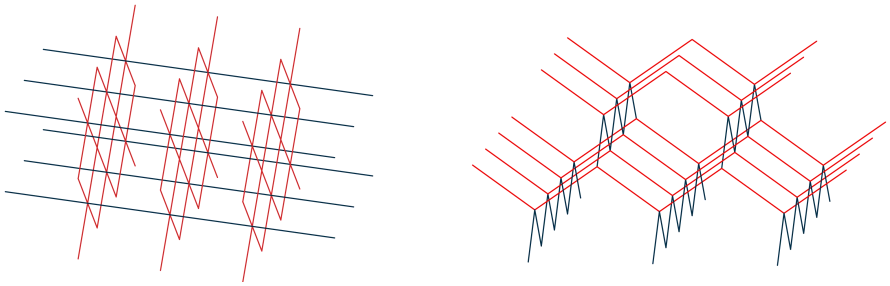
3.3.4 Developments of Space-filling Grid 3

After the discovery of space-filling Grid 2, some drawbacks appeared. One obvious problem is the grid of the system. When connecting the center point of every block, there are obvious stratification causing the incoherent structural system (Figure 3.8(a)). To deal with this issue, the third type was explored to solve the problem. By adding another two small blocks on the bottom to achieve balance within each modular block (Figure 3.7(b)), Grid 3 (Figure 3.7(a)) is able to ensure a space-filling pattern, form a three-dimensional dual graph when connecting their central point, as well as ensure topological interlocking (for the monolithic coloured blocks) (Figure 3.8 (b)).



(a) (b)

Figure 3.7 *(a) Space-filling Grid 3 (Source from Qinglu Chen and Baolian Liu) (b) Generation of Grid 3*



(a) (b)

Figure 3.8 *Dual graph of (a) Grid 2 and (b) Grid 3 (Source from Qinglu Chen and Baolian Liu)*

3.3.5 General Comparison

After the generation of three types of space-filling grids, a general comparison (Figure 3.9) is conducted to compare these three grids. By comparing their dual graph and force flow path (a.k.a. a thrust network), vertical and horizontal performance, load path angle and if they can ensure interlocking, two grids are selected for the following steps.

Considering the dual graph, the dual graphs of Grid 1 and Grid 3 are three-dimensional, while the obvious stratification of the dual graph for Grid 2 causing an incoherent structural system, therefore both Grid 1 and Grid 3 are suitable for compression-only structures, while Grid 2 needs cable or other support to link the system. As for interlocking performance, both Grid 2 and Grid 3 can ensure pre-set interlocking by the geometry of modular block itself, while no interlocking performance are shown in Grid 1, resulting in less advantage of Grid 1 under horizontal loads. However, this drawback can be redesigned afterwards to ensure post-interlocking in the following step, but the stratification of Grid 2 is not changeable. Therefore, Grid 1 and Grid 3 are selected for further structural verifications.

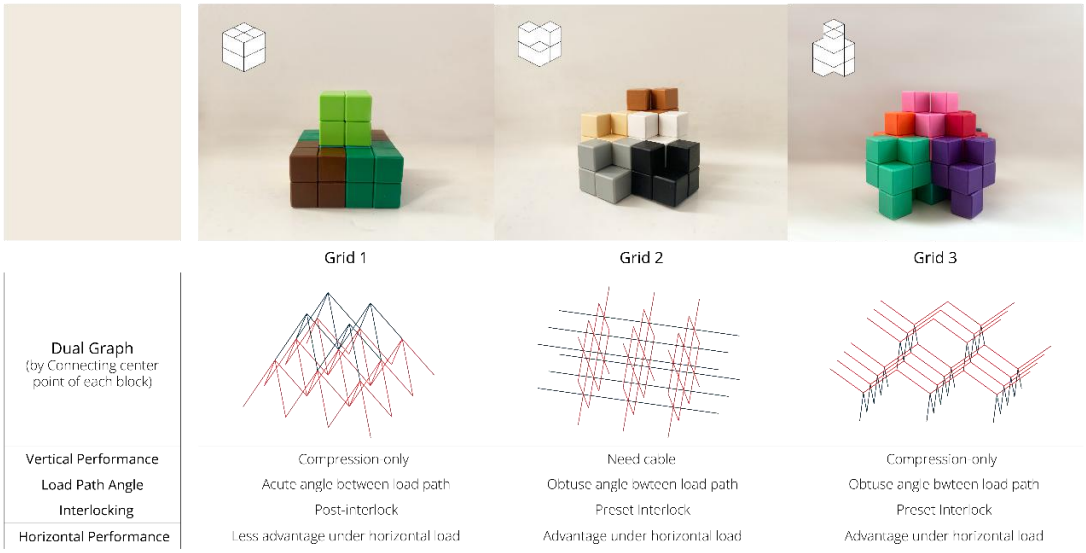


Figure 3.9 *General Comparison between three Grids (Source from Qinglu Chen and Baolian Liu)*

3.4 Structural Comparison between Two Grids

3.4.1 Introduction

In order to test the stability of the designed space-filling grids, structural verifications are conducted to compare which grid performs better for different hypothetical load cases using FEM and DEM, respectively utilizing Abaqus [30] and 3DEC [31]. Three kinds of structural analyses are conducted for grid comparison, proportion comparison, and ability to self-support during construction. The grid which performs better during structural verification is then chosen as the foundation for the design of the algorithm for approximating vault structures.

3.4.2 Workflow for Structural Verification Process

The workflow of structural analyses can be seen in Figure 3.10. The process for initializing the analysis for both Abaqus and 3DEC began with set limits, then modular blocks were created within the software. After material property is assigned to the modular blocks, assembly process started to combine several modular blocks need to be analyzed together. After that, interaction properties were assigned to the contact surfaces and different load cases are applied on corresponding surfaces, then steps are set for running the analysis. The input property of materials and load cases (Appendix 1) are based on scientific paper and Eurocode [32].

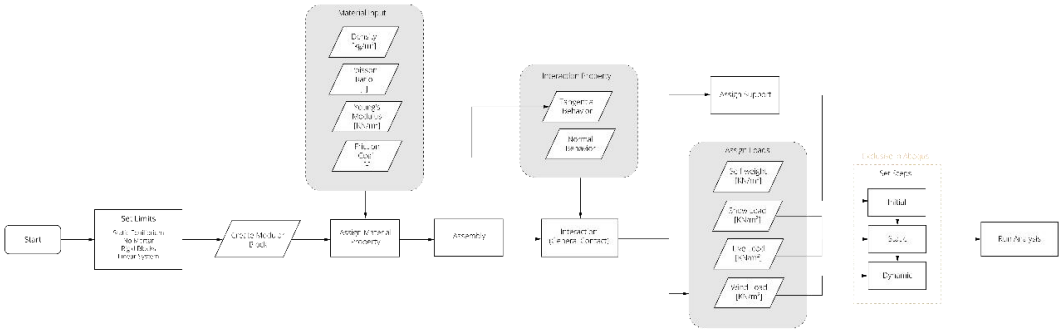


Figure 3.10 *Workflow for Structural Analyses Process (Source from Qinglu Chen and Baolian Liu)*

3.4.3 Grid Comparison

The first set of analyses focuses on comparing the performance of Grid 1 and Grid 3 under different load cases, with the simulations conducted using both FEM in Abaqus and DEM in 3DEC. In this phase of simulation, every voxel has the proportion of 1:1 to ensure the comparability of both grids. The model of Grid 1 consists of 6 modular blocks (with 8 voxels within each block) (Figure 3.11(a)), while Grid 3 contains 9 modular blocks (with 8 voxels within each block as well) (Figure 3.11(b)). Two types of load conditions are applied to the model, one for vertical loads only (5175N/m²), while the other for both vertical loads (5175N/m²) and horizontal loads (525N/m²) (Appendix 2), and the results are evaluated in terms of principal stresses and displacements.

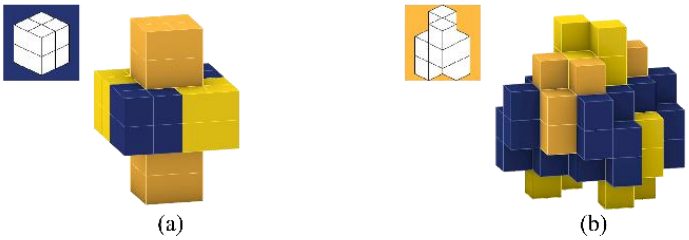
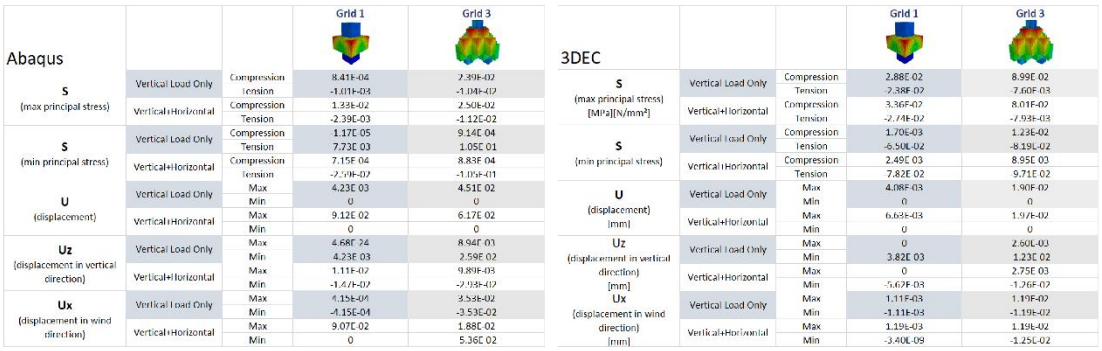


Figure 3.11 *(a) 6 blocks of Grid 1 (48 voxels) (b) 9 blocks of Grid 3 (72 voxels)*



Abaqus

			Grid 1	Grid 3
S (max principal stress)	Vertical Load Only	Compression	8.41E-04	2.39E-02
		Tension	-1.01E-03	-1.04E-02
	Vertical+Horizontal	Compression	1.33E-02	2.50E-02
		Tension	-2.39E-03	-1.12E-02
S (min principal stress)	Vertical Load Only	Compression	1.17E 05	9.14E 04
		Tension	7.73E 03	1.05E 01
	Vertical+Horizontal	Compression	7.15E 04	8.83E 04
		Tension	-2.59E-07	-1.05E-01
U (displacement)	Vertical Load Only	Max	4.23E 03	4.51E 02
		Min	0	0
	Vertical+Horizontal	Max	9.12E 02	6.17E 02
		Min	0	0
Uz (displacement in vertical direction)	Vertical Load Only	Max	4.68E-24	8.94E-03
		Min	4.23E 03	2.59E 02
	Vertical+Horizontal	Max	1.11E-02	9.89E-03
		Min	-1.47E-07	-2.93E-02
Ux (displacement in wind direction)	Vertical Load Only	Max	4.15E-04	3.53E-02
		Min	-4.15E-04	-3.53E-02
	Vertical+Horizontal	Max	9.07E-02	1.88E-02
		Min	0	5.36E 02

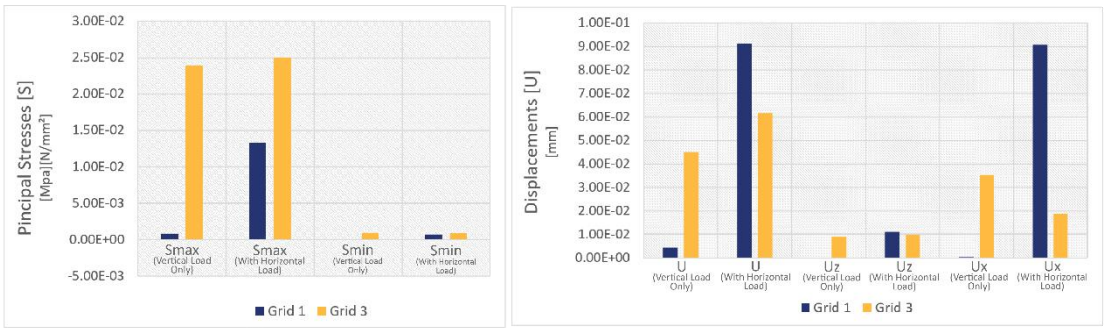
(a)

3DEC

			Grid 1	Grid 3
S (max principal stress) [MPa][N/mm²]	Vertical Load Only	Compression	2.88E 02	8.99E 02
		Tension	-2.38E-02	-7.60E-03
	Vertical+Horizontal	Compression	3.36E-02	8.01E-02
		Tension	-2.74E-02	-7.93E-03
S (min principal stress)	Vertical Load Only	Compression	1.70E-03	1.23E-02
		Tension	-6.50E-02	-8.19E-02
	Vertical+Horizontal	Compression	2.49E 03	8.95E 03
		Tension	7.82E 02	9.71E 02
U (displacement) [mm]	Vertical Load Only	Max	4.08E-03	1.90E-02
		Min	0	0
	Vertical+Horizontal	Max	6.63E-03	1.97E-02
		Min	0	0
Uz (displacement in vertical direction) [mm]	Vertical Load Only	Max	0	2.60E-03
		Min	3.82E 03	1.23E 02
	Vertical+Horizontal	Max	0	2.75E 03
		Min	-5.67E-03	-1.26E-02
Ux (displacement in wind direction) [mm]	Vertical Load Only	Max	1.11E-03	1.19E-02
		Min	-1.11E-03	-1.19E-02
	Vertical+Horizontal	Max	1.19E-03	1.19E-02
		Min	-3.40E-09	-1.25E-02

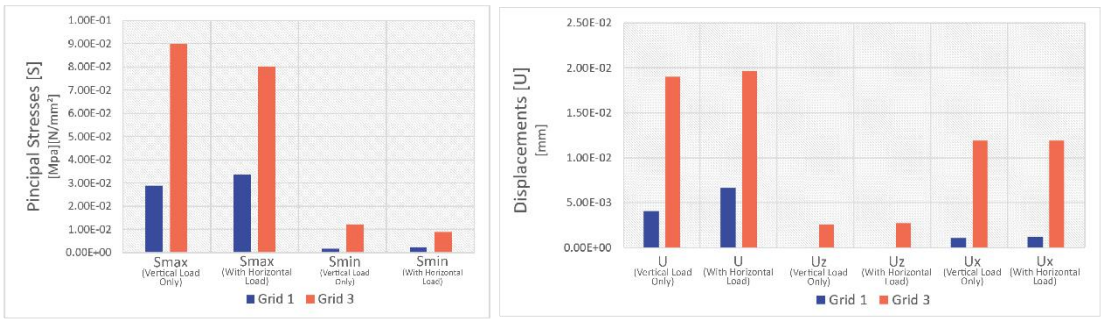
(b)

Figure 3.12 *Data from (a) Abaqus and (b) 3DEC*



(a) (b)

Figure 3.13 *Results in Abaqus (a) Principal stresses and (b) displacements for Grid 1 and Grid 3 (Source from Qinglu Chen and Baolian Liu)*



(a) (b)

Figure 3.14 *Results in 3DEC (a) Principal stresses and (b) displacements for Grid 1 and Grid 3*

Table in Figure 3.12 shows the simulation results of Grid 1 and Grid 3 under the same vertical load cases and horizontal load cases. In both results, ten values are considered for comparison, namely S (maximum principal stress), S (minimum principal stress), U (total displacement), Uz (displacement in vertical direction), Ux (displacement in wind direction), Uy (displacement in another horizontal direction). It can be observed that for both load cases in both results of Abaqus and 3DEC, Grid 1 experiences smaller principal stresses than Grid 3 (Figure 3.13(a) and Figure 3.14(a)), but comparable displacements ((Figure 3.13(b) and Figure 3.14(b)). As is shown in the charts, two software has some differences regarding the displacements due to some

settings are basically different inherited in two softwares. For vertical load-only conditions, Grid 1 has smaller displacements, however, Grid 3 performs better under the influence of horizontal loads in Abaqus. However, Grid 1 performs better than Grid 3 regarding displacements in 3DEC. Also, considering the material use, Grid 3 needs more material to achieve the same height as Grid 1. Therefore, Grid 1 is more efficient for structures experiencing predominately vertical loading, while Grid 3 might be a good choice when applied to a location where horizontal loads are high.

3.4.3 Proportion Comparison (First Round)

In the last phase of grid comparison, 1:1 proportion of small voxels is under assumption of both grids. However, the proportion of block, that is, the ratio between its width and height is of extreme importance because it determines the angle of the force flow path, as well as influences the structural behavior regarding stability, rigidity and spatial geometry in the process of space formation. Therefore, in the first phase of exploring proportion, 4 different variations are tested in simulation by 3DEC, respectively proportion 1:2, 1:1, 2:1 and 3:1 (Figure 3.15). These four proportions is aiming at trying as many variations as possible, but at the same time avoiding extreme proportions which would result in less rigidity of modular block and difficulty in manufacturing.

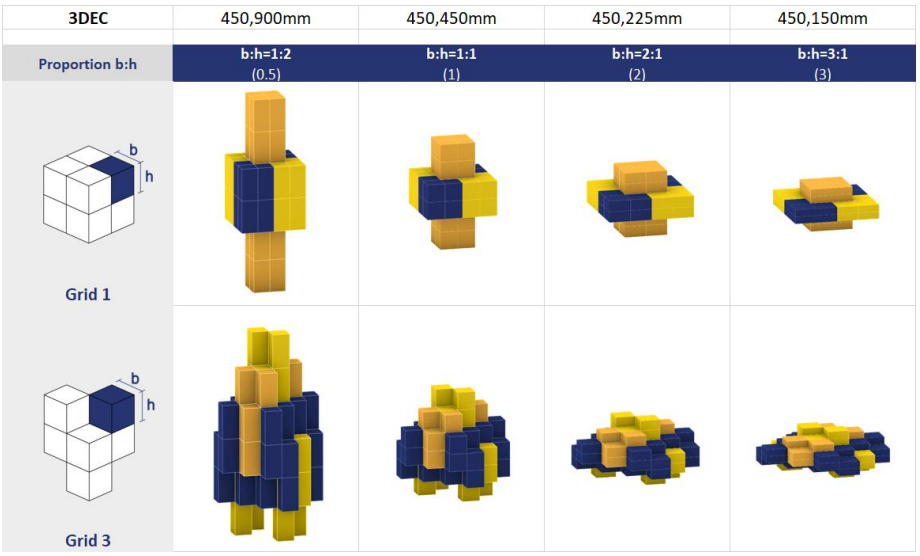


Figure 3.15 *Four Proportions of Grid 1 and Grid 3*

Considering that each modular block is consisted of 8 small voxels and it's hard to define the proportion under the same circumstances due to the fact that the heights of two grids are consisted of different number of voxels, respectively 2 and 3, the proportion exploration is within each small voxel to test the proportion between the width(b) and the height(h) (Figure 3.16).

Also, because every upper block relies on one-quarter of the lower block, the base surface for both grids should keep as square shape to remain isotropic in both x and y direction, thus only one width(b) is considered in this step.

Figure 3.16 *Proportion between width (b) and height (h) of a voxel*

In the simulation of first phase, two kinds of vertical load cases are considered. One is under 30 Pa (N/m²), the other is under 5175 Pa (N/m²). The aim of doing two kinds of vertical load is to test the stability of both grids when there is barely load as well as there are live loads on the top. Also, the two conditions of vertical load only and with vertical and horizontal loads remains the same with the previous simulations. The table of results can be seen in Figure 3.17.

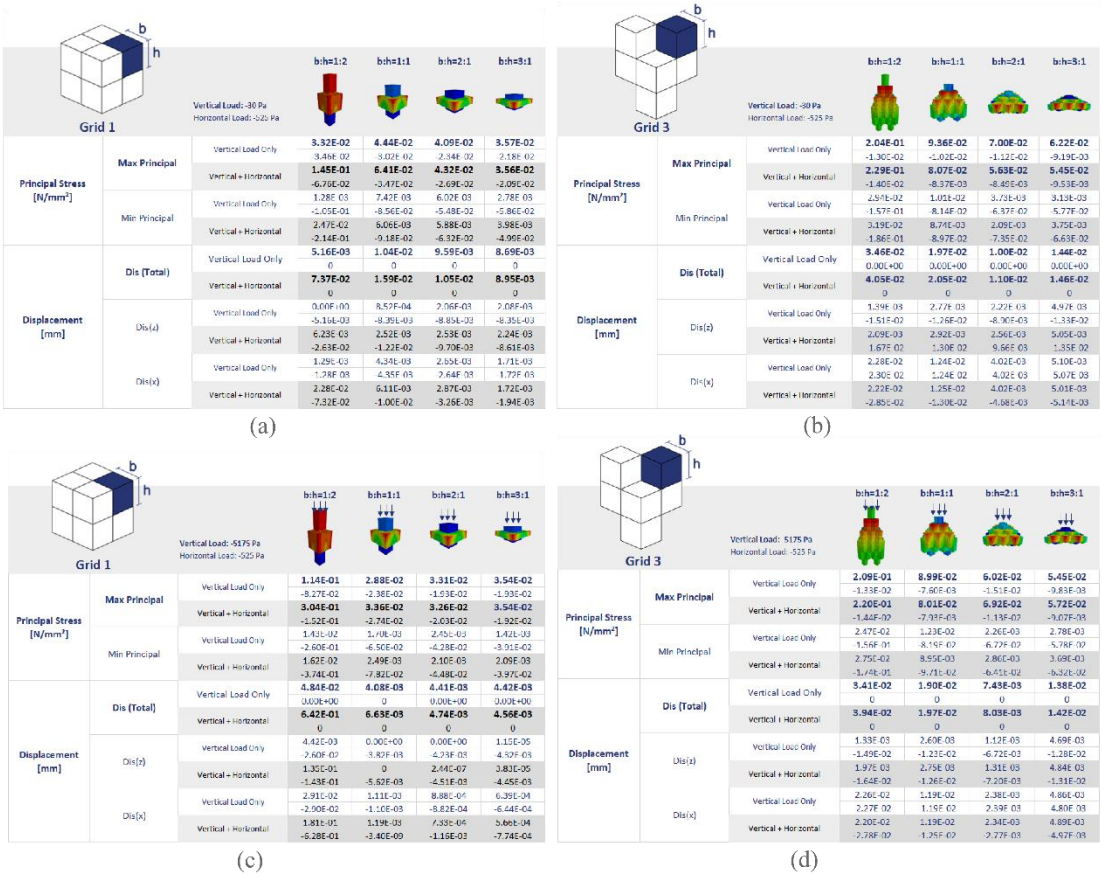


Figure 3.17 *Data of Grid 1 for vertical load case (a)30N/㎡ and (b) 5175N/㎡; Grid 3 for vertical load case (c)30N/㎡ and (d) 5175N/㎡*

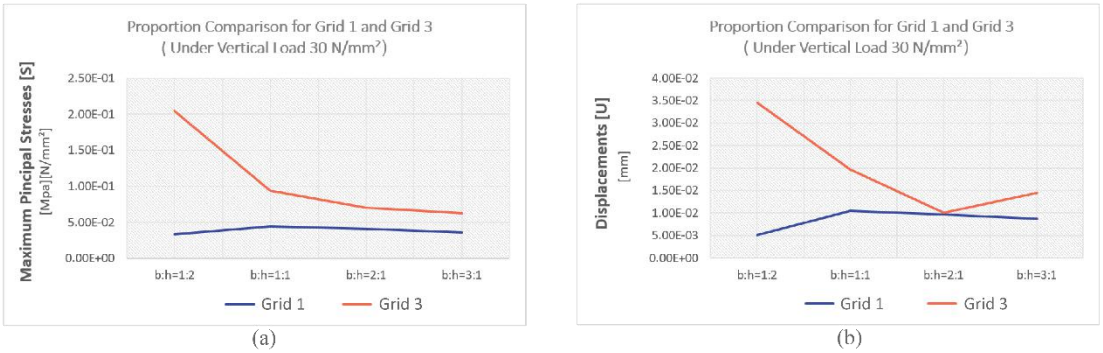


Figure 3.18 *(a) Principal Stress and (b)Displacements of four proportions of Grid 1 and Grid 3 under vertical load of 30 Pa*

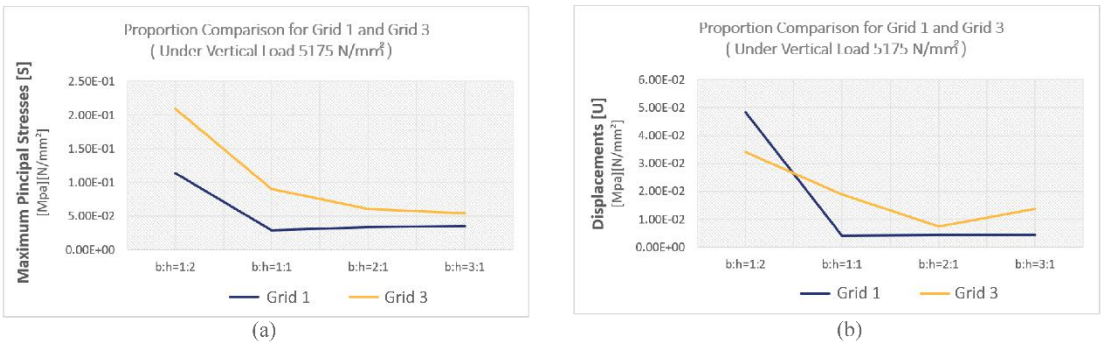


Figure 3.19 *(a)Principal Stress and (b)Displacements of four proportions of Grid 1 and Grid 3 under vertical load of 5175 Pa*

The chart of results of the first-round simulation are shown in Figure 3.18 and 3.19. Keeping the same with the previous one, principal stresses and total displacements are the main consideration when comparing results. It can tell that for both vertical load conditions (under vertical load of 30 N/m² and 5175 N/m²), Grid 1 has less maximum principal stress and displacements in most cases of the compared four proportions, though there are special case in displacements result of proportion b:h = 1:2 when Grid 3 has less displacements than Grid 1. When considering the tendency of proportion change, it is not hard to tell that when proportion becomes larger (from b:h = 1:2 (0.5) to b:h = 3:1 (3)), both maximum principal stresses and displacements have the tendency of becoming smaller, though there are strange peak points existing at proportion b:h = 1:1 and 2:1. Overall, Grid 1 has better performance regarding both principal stresses and displacements in most cases under both load conditions and the tendency for proportion change has a trend to become smaller when the proportion becomes larger.

3.4.4 Ability of Self-Supporting (First Round)

In this sub-section, joints for connection between modular blocks are designed to test its ability of self-supporting during construction. For both

Grid 1 and Grid 3, the upper block stands on one-quarter of the lower block, consequently, the area of the contact surface is not large enough for the upper block to stand without support. Inspired from a paper regarding interlocking element design by Nabila Aff.etc [33], two types of interlocking joints are designed for both grids, namely pin joint and groove joint respectively. In the paper described, several interlocking joints were prototyped and tested, and groove joint are the type most suitable for this project due its advantage of good self-supporting ability and good for construction [33]. Another type – pin joint- is developed from the Lego joints, with a pin joint in between the blocks. For Grid 1, two types of joints are offset from the edge to ensure a safe area when collapse happens, while both types of joints are at the edge of Grid 3 to ensure the largest contact surface of between the blocks (Figure 3.20).

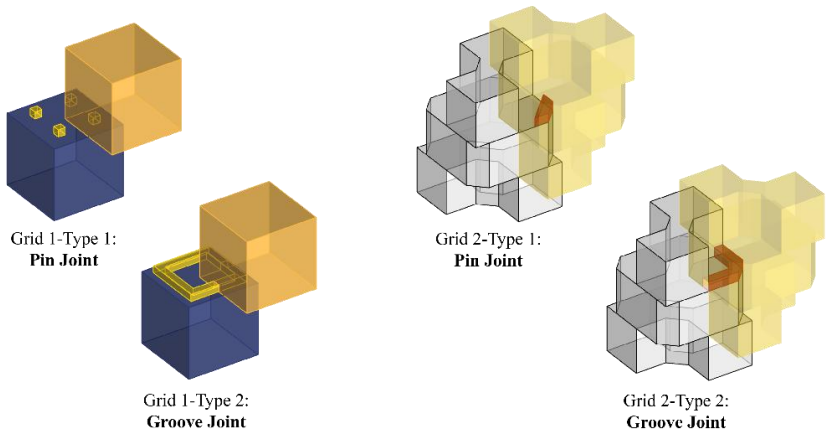
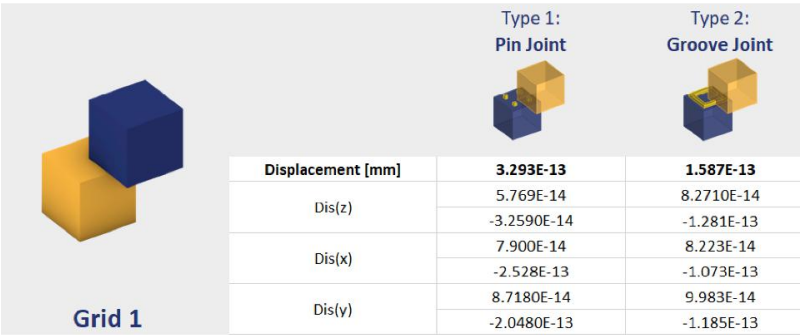
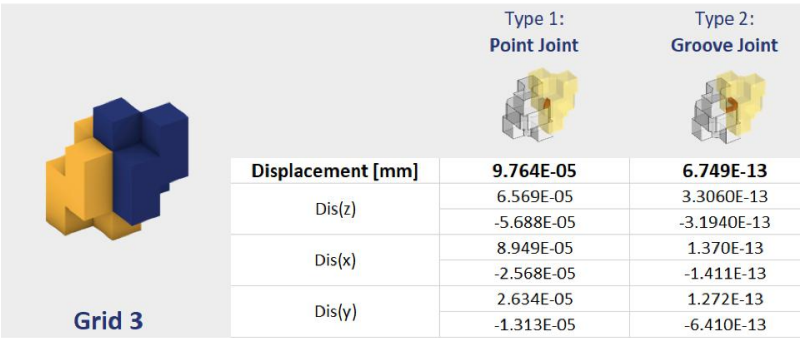


Figure 3.20 *Two types of interlocking joint for Grid 1 and Grid 3*



		Type 1: Pin Joint	Type 2: Groove Joint
Grid 1	Displacement [mm]	**3.293E-13**	**1.587E-13**
	Dis(z)	5.769E-14	8.2710E-14
		-3.2590E-14	-1.281E-13
	Dis(x)	7.900E-14	8.223E-14
		-2.528E-13	-1.073E-13
	Dis(y)	8.7180E-14	9.983E-14
		-2.0480E-13	-1.185E-13

(a)



		Type 1: Point Joint	Type 2: Groove Joint
Grid 3	Displacement [mm]	**9.764E-05**	**6.749E-13**
	Dis(z)	6.569E-05	3.3060E-13
		-5.688E-05	-3.1940E-13
	Dis(x)	8.949E-05	1.370E-13
		-2.568E-05	-1.411E-13
	Dis(y)	2.634E-05	1.272E-13
		-1.313E-05	-6.410E-13

(b)

Figure 3.21 *Displacements of two types of joints for (a) Grid 1 and (b) Grid 3*

The simulation results are shown as table in Figure 3.21 and it is worth noting that this simulation has no applied load, aiming at testing its ability of self-supporting during construction process. It is not hard to tell that Grid 1 has much less displacements than Grid 3 for both types of joints shown in Figure 3.22. When comparing between the two types of joints, groove joints have larger advantage than pin joints for Grid 3, while comparable displacements for Grid 1 (Figure 3.23). Nevertheless, Grid 1 has better performance regarding the displacements when in comparison with both types of joints.

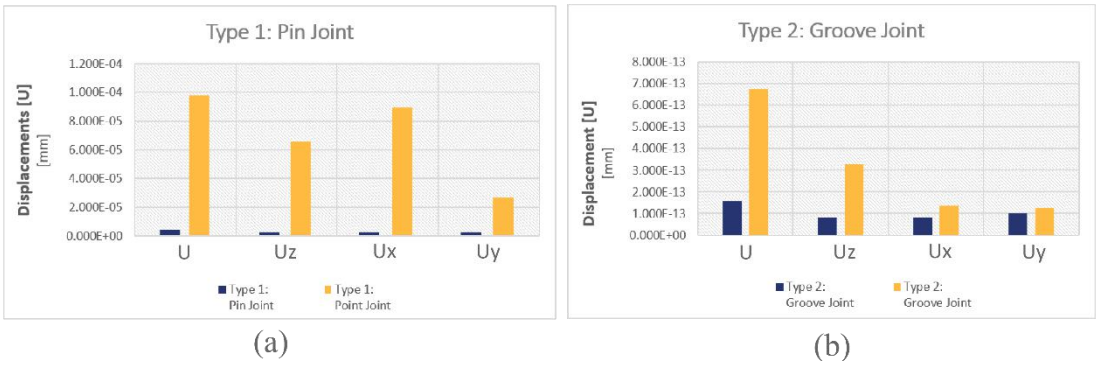


Figure 3.22 *Displacements comparison between Grid 1 and Grid 3 for (a) Pin Joint and (b) Groove Joint*

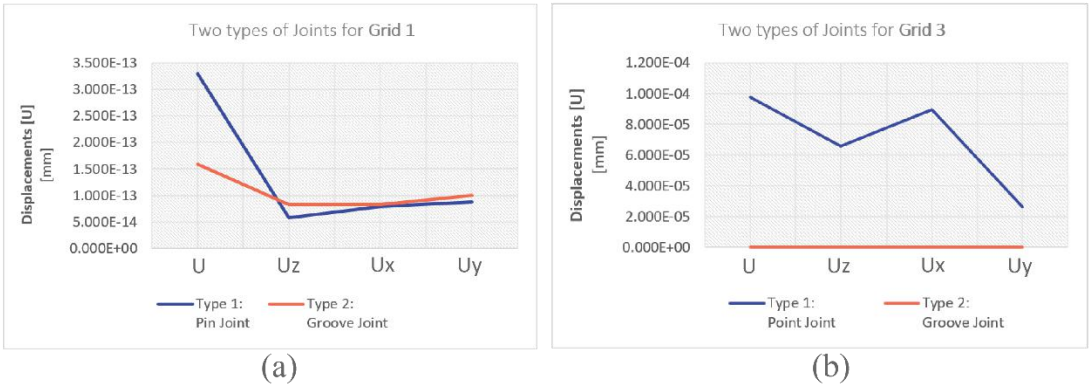


Figure 3.23 *Displacements comparison between two types of joints for (a) Grid 1 and (b) Grid 3*

3.4.5 Conclusion of Chosen Grid

In this section, three types of simulations are conducted to compare the structural performance between Grid 1 and Grid 3. In all the simulations, Grid 1 seems to have a better performance regarding grid comparison, proportion exploration as well as ability of self-supporting. Therefore, Grid 1 is chosen for further exploration.

3.5 Structural Exploration of Grid 1

3.5.1 Introduction

In previous section, Grid 1 is chosen since it has better structural performance than Grid 3. To have a more comprehensive exploration regarding Grid 1, two second-round simulations are thus conducted for more proportion explorations and to test which proportion performs better with interlocking joint. In the end, Grid 1 with a proper proportion is chosen as the base for the algorithm design.

3.5.2 Proportion Exploration (Second Round)

After exploration of four proportions for Grid 1, there exists strange peak points in the results. Therefore, more proportions are added in the second-round simulation. In this case, eleven proportions are simulated in 3DEC with the same two load conditions as before (Figure 3.24). The proportions are varied from b:h=1:2 to b:h=3:1, as the objective is to comprehensively assess as many proportions as possible. It should be noted that due to the extreme results of proportion b:h=1:2, the results in the plots (Figure 3.25) only include the last ten proportions.

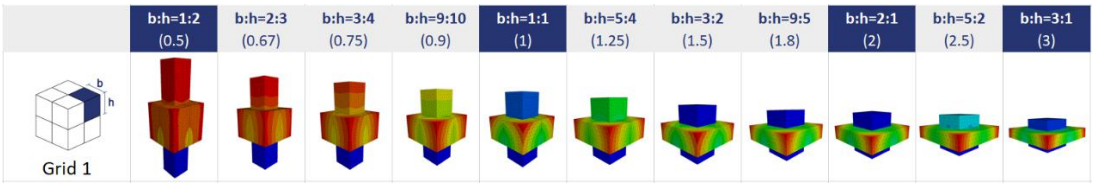


Figure 3.24 *Eleven proportions of Grid 1 for simulation*

The results in terms of principal stresses and displacements are shown in Figure 3.25 and the charts are shown in Figure 3.26 and 3.27 respectively. They have a similar tendency for the two load conditions, with a big gap for smaller proportions which gradually reduces as the proportion becomes larger. For the 'vertical loads only' condition, both principal stresses and displacements slightly decline first, then increase again as the proportion increases. When the horizontal loads are involved, the trend for the declining stage is steeper. Nevertheless, larger proportions tend to have a more stable behaviour with respect to both principal stresses and displacements. To that end, proportions b:h = 9:5, 2:1, 5:2, and 3:1 are taken into consideration for further analysis. It must be noted that we have restricted to rational proportions with integer dimensions as the base grid is a regular and isotropic voxel grid.

Grid 1
(vertical load: 30 Pa; Horizontal Load: 525 Pa)

			b:h=1:2 (0.5)	b:h=2:3 (0.67)	b:h=3:4 (0.75)	b:h=9:10 (0.9)	b:h=1:1 (1)	b:h=5:4 (1.25)	b:h=3:2 (1.5)	b:h=9:5 (1.8)	b:h=2:1 (2)	b:h=5:2 (2.5)	b:h=3:1 (3)
Principal Stress [N/mm²]	Max Principal	Vertical Load Only	3.32E-02	3.43E-02	4.22E-02	4.37E-02	4.44E-02	4.66E-02	4.55E-02	4.60E-02	4.09E-02	3.73E-02	3.57E-02
			-3.46E-02	-3.43E-02	-3.54E-02	-3.48E-02	-3.02E-02	-2.80E-02	-2.55E-02	-2.46E-02	-2.34E-02	-2.19E-02	-2.18E-02
		Vertical + Horizontal	1.45E-01	1.04E-01	9.41E-02	7.79E-02	6.41E-02	5.53E-02	4.79E-02	4.26E-02	4.32E-02	3.72E-02	3.56E-02
			-6.76E-02	-5.08E-02	-4.51E-02	-3.75E-02	-3.47E-02	-2.79E-02	-2.67E-02	-2.43E-02	-2.69E-02	-2.24E-02	-2.09E-02
	Min Principal	Vertical Load Only	1.28E-03	2.76E-03	4.10E-03	3.33E-03	7.42E-03	6.82E-03	6.35E-03	7.34E-03	6.02E-03	6.89E-03	2.78E-03
			-1.05E-01	-9.03E-02	-8.82E-02	-7.89E-02	-8.56E-02	-7.69E-02	-6.75E-02	-6.65E-02	-5.48E-02	-4.64E-02	-5.86E-02
		Vertical + Horizontal	2.47E-02	1.59E-02	1.64E-02	1.24E-02	6.06E-03	7.71E-03	5.56E-03	8.48E-03	5.88E-03	6.13E-03	3.98E-03
			-2.14E-01	-1.36E-01	-1.29E-01	-1.03E-02	-9.18E-02	-7.99E-02	-6.83E-02	-6.27E-02	-6.32E-02	-5.74E-02	-4.99E-02
Displacement [mm]	Dis (Total)	Vertical Load Only	5.16E-03	8.12E-03	9.30E-03	1.02E-02	1.04E-02	1.05E-02	1.02E-02	9.85E-03	9.59E-03	9.24E-03	8.69E-03
			0	0	0	0	0	0	0	0	0	0	0
		Vertical + Horizontal	7.37E-02	3.07E-02	2.36E-02	1.81E-02	1.59E-02	1.29E-02	1.17E-02	1.08E-02	1.05E-02	9.66E-03	8.95E-03
			0	0	0	0	0	0	0	0	0	0	0
	Dis(z)	Vertical Load Only	0	0	0	5.21E-04	8.52E-04	1.48E-03	1.77E-03	1.98E-03	2.06E-03	2.17E-03	2.08E-03
			-5.16E-03	-6.52E-03	-7.28E-03	-8.07E-03	-8.39E-03	-8.94E-03	-8.97E-03	-8.93E-03	-8.85E-03	-8.73E-03	-8.35E-03
		Vertical + Horizontal	6.23E-03	3.79E-03	3.24E-03	2.69E-03	2.52E-03	2.35E-03	2.40E-03	2.48E-03	2.53E-03	2.45E-03	2.24E-03
			-2.63E-02	-1.80E-02	-1.58E-02	-1.32E-02	-1.22E-02	-1.08E-02	-1.02E-02	-9.83E-03	-9.70E-03	-9.13E-03	-8.61E-03
	Dis(x) (wind direction)	Vertical Load Only	1.29E-03	3.62E-03	4.21E-03	4.39E-03	4.34E-03	3.98E-03	3.46E-03	2.92E-03	2.65E-03	2.12E-03	1.71E-03
			-1.28E-03	-3.62E-03	-4.21E-03	-4.43E-03	-4.35E-03	-3.98E-03	-3.46E-03	-2.95E-03	-2.64E-03	-2.14E-03	-1.72E-03
		Vertical + Horizontal	2.28E-02	1.21E-02	9.74E-03	7.15E-03	6.11E-03	4.58E-03	3.81E-03	3.18E-03	2.87E-03	2.19E-03	1.72E-03
			-7.32E-02	-3.03E-02	-2.18E-02	-1.32E-02	-1.00E-02	-5.82E-02	-4.64E-03	-3.67E-03	-3.26E-03	-2.47E-03	-1.94E-03

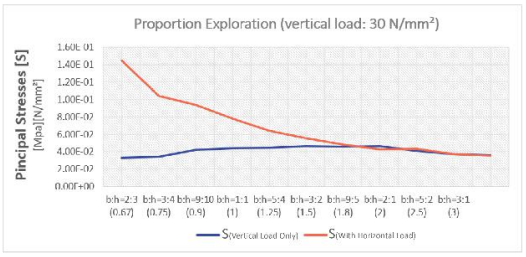
(a)

Grid 1
(vertical load: 5175 Pa; Horizontal Load: 525 Pa)

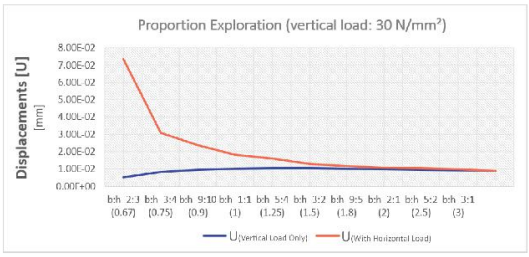
			b:h=1:2 (0.5)	b:h=2:3 (0.67)	b:h=3:4 (0.75)	b:h=9:10 (0.9)	b:h=1:1 (1)	b:h=5:4 (1.25)	b:h=3:2 (1.5)	b:h=9:5 (1.8)	b:h=2:1 (2)	b:h=5:2 (2.5)	b:h=3:1 (3)
Principal Stress [N/mm²]	Max Principal	Vertical Load Only	1.14E-01	3.25E-02	3.11E-02	2.95E-02	2.88E-02	2.93E-02	3.08E-02	3.18E-02	3.31E-02	3.45E-02	3.54E-02
			-8.27E-02	-2.70E-02	-2.73E-02	-2.49E-02	-2.38E-02	-2.29E-02	-2.08E-02	-2.06E-02	-1.93E-02	-1.92E-02	-1.93E-02
		Vertical + Horizontal	3.04E-01	4.62E-02	3.97E-02	3.87E-02	3.36E-02	3.16E-02	3.19E-02	3.26E-02	3.26E-02	3.35E-02	3.54E-02
			-1.52E-01	-3.62E-02	-3.48E-02	-3.01E-02	-2.74E-02	-2.50E-02	-2.16E-02	-2.04E-02	-2.03E-02	-1.92E-02	-1.92E-02
	Min Principal	Vertical Load Only	1.43E-02	1.53E-03	1.84E-03	1.28E-03	1.70E-03	2.14E-03	1.77E-03	1.72E-03	2.45E-03	2.79E-03	1.42E-03
			-2.60E-01	-8.73E-02	-8.15E-02	-6.85E-02	-6.50E-02	-5.64E-02	-4.89E-02	-4.55E-02	-4.28E-02	-4.05E-02	-3.91E-02
		Vertical + Horizontal	1.62E-02	1.49E-03	2.00E-03	3.13E-03	2.49E-03	2.42E-03	2.07E-03	2.43E-03	2.10E-03	2.06E-03	2.09E-03
			-3.74E-01	-1.12E-02	-9.76E-02	-7.81E-02	-7.82E-02	-5.99E-02	-5.02E-02	-4.37E-02	-4.48E-02	-3.81E-02	-3.97E-02
Displacement [mm]	Dis (Total)	Vertical Load Only	4.84E-02	4.17E-03	4.00E-03	4.03E-03	4.08E-03	4.29E-03	4.38E-03	4.40E-03	4.41E-03	4.39E-03	4.42E-03
			0	0	0	0	0	0	0	0	0	0	0
		Vertical + Horizontal	6.42E-01	1.26E-02	1.00E-02	7.59E-03	6.63E-03	5.56E-03	5.15E-03	4.82E-03	4.74E-03	4.58E-03	4.56E-03
			0	0	0	0	0	0	0	0	0	0	0
	Dis(z)	Vertical Load Only	4.42E-03	0	0	0	0	0	0	0	0	0	1.15E-05
			-2.60E-02	-4.17E-03	-3.90E-03	-3.81E-03	-3.82E-03	-4.00E-03	-4.12E-03	-4.19E-03	-4.23E-03	-4.26E-03	-4.32E-03
		Vertical + Horizontal	1.35E-01	0	0	0	0	0	0	0	2.44E-07	3.09E-05	3.83E-05
			-1.43E-01	-8.49E-03	-7.38E-03	-6.16E-03	-5.62E-03	-4.99E-03	-4.75E-03	-4.54E-03	-4.51E-03	-4.44E-03	-4.45E-03
	Dis(x) (wind direction)	Vertical Load Only	2.91E-02	9.97E-04	1.02E-03	1.09E-03	1.11E-03	1.13E-03	1.07E-03	9.56E-04	8.88E-04	7.26E-04	6.39E-04
			-2.90E-02	-9.92E-04	-1.02E-03	-1.09E-03	-1.10E-03	-1.14E-03	-1.07E-03	-9.56E-04	-8.82E-04	-7.40E-04	-6.44E-04
		Vertical + Horizontal	1.81E-01	1.52E-03	1.51E-03	1.36E-03	1.19E-03	1.07E-03	9.09E-04	7.90E-04	7.33E-04	6.37E-04	5.66E-04
			-6.28E-01	-1.15E-02	-8.08E-03	-4.56E-03	-3.40E-09	-2.19E-03	-1.67E-03	-1.31E-03	-1.16E-03	-9.23E-04	-7.74E-04

(b)

Figure 3.25 *Table of results for under vertical load of (a)30 MPa and (b) 5175 MPa*

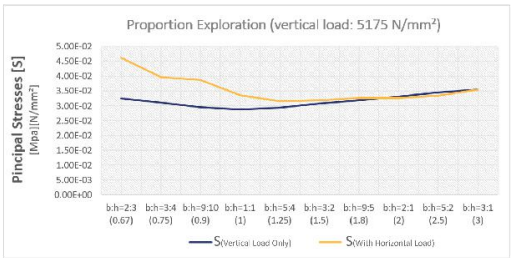


(a)

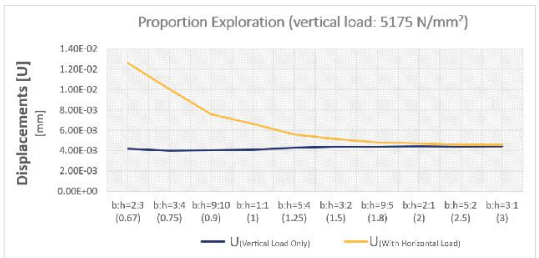


(b)

Figure 3.26 *(a)Principal stresses and (b)displacements for ten proportions of Grid 1 under vertical load of 30 MPa and Horizontal Load 525 MPa*



(a)



(b)

Figure 3.27 *(a)Principal stresses and (b)displacements for ten proportions of Grid 1 under vertical load of 5175 MPa and Horizontal Load 525 MPa*

3.5.3 Ability of Self-Supporting (Second Round)

The aim of these additional analyses is to select the most promising proportion within the modular block. To that end, the same eleven proportions of Grid 1 are again simulated with the interlocking joints (Figure 3.28). In the previous tests, the groove joints show a significant advantage with respect to the total displacement, therefore, this type of joint is selected in these round simulations.

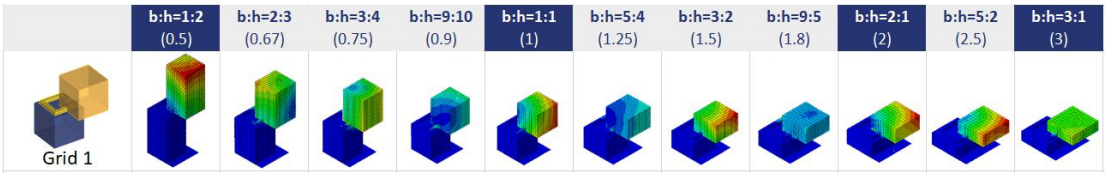
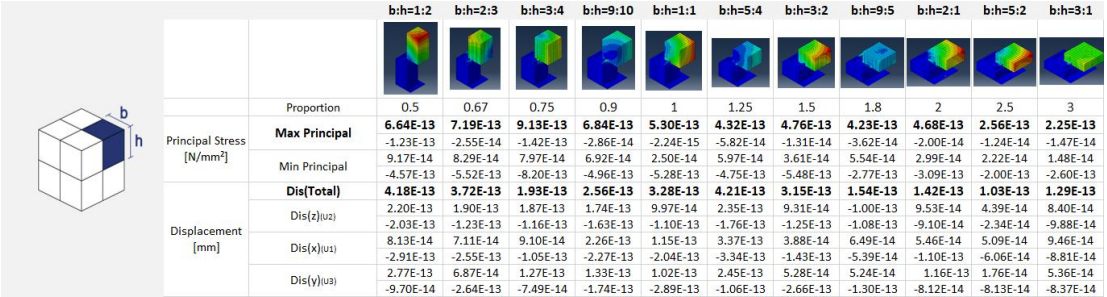


Figure 3.28 *Eleven proportions of grid 1 with groove joints*

The table of results are shown in Figure 3.29. In general, the results of the maximum principal stresses (Figure 3.30(a)) seem to decrease as the proportion increases. For displacements (Figure 3.30(b)), there are strange peak points appearing at proportion 5:4, however, the overall trend is declining as well and the smallest displacements appear at the last two proportions of 5:2 and 3:1.



		b:h=1:2	b:h=2:3	b:h=3:4	b:h=9:10	b:h=1:1	b:h=5:4	b:h=3:2	b:h=9:5	b:h=2:1	b:h=5:2	b:h=3:1
	Proportion	0.5	0.67	0.75	0.9	1	1.25	1.5	1.8	2	2.5	3
Principal Stress [N/mm²]	**Max Principal**	**6.64E-13**	**7.19E-13**	**9.13E-13**	**6.84E-13**	**5.30E-13**	**4.32E-13**	**4.76E-13**	**4.23E-13**	**4.68E-13**	**2.56E-13**	**2.25E-13**
		-1.23E-13	-2.55E-14	-1.42E-13	-2.86E-14	-2.24E-15	-5.82E-14	-1.31E-14	-3.62E-14	-2.00E-14	-1.24E-14	-1.47E-14
	Min Principal	9.17E-14	8.29E-14	7.97E-14	6.92E-14	2.50E-14	5.97E-14	3.61E-14	5.54E-14	2.99E-14	2.22E-14	1.48E-14
		-4.57E-13	-5.52E-13	-8.20E-13	-4.96E-13	-5.28E-13	-4.75E-13	-5.48E-13	-2.77E-13	-3.09E-13	-2.00E-13	-2.60E-13
Displacement [mm]	**Dis(Total)**	**4.18E-13**	**3.72E-13**	**1.93E-13**	**2.56E-13**	**3.28E-13**	**4.21E-13**	**3.15E-13**	**1.54E-13**	**1.42E-13**	**1.03E-13**	**1.29E-13**
	Dis(z)(U2)	2.20E-13	1.90E-13	1.87E-13	1.74E-13	9.97E-14	2.35E-13	9.31E-14	-1.00E-13	9.53E-14	4.39E-14	8.40E-14
		-2.03E-13	-1.23E-13	-1.16E-13	-1.63E-13	-1.10E-13	-1.76E-13	-1.25E-13	-1.08E-13	-9.10E-14	-2.34E-14	-9.88E-14
	Dis(x)(U1)	8.13E-14	7.11E-14	9.10E-14	2.26E-13	1.15E-13	3.37E-13	3.88E-14	6.49E-14	5.46E-14	5.09E-14	9.46E-14
		-2.91E-13	-2.55E-13	-1.05E-13	-2.27E-13	-2.04E-13	-3.34E-13	-1.43E-13	-5.39E-14	-1.10E-13	-6.06E-14	-8.81E-14
	Dis(y)(U3)	2.77E-13	6.87E-14	1.27E-13	1.33E-13	1.02E-13	2.45E-13	5.28E-14	5.24E-14	1.16E-13	1.76E-14	5.36E-14
		-9.70E-14	-2.64E-13	-7.49E-14	-1.74E-13	-2.89E-13	-1.06E-13	-2.66E-13	-1.30E-13	-8.12E-14	-8.13E-14	-8.37E-14

Figure 3.29 *Table of results of displacements of Grid 1 with groove joints*

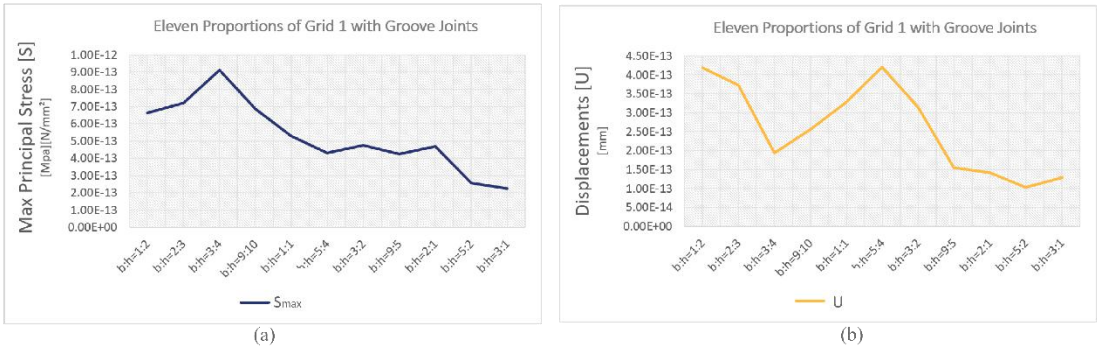


Figure 3.30 *(a) Principal Stresses and (b) displacements of Grid 1 with groove joints*

3.5.4 Conclusion of Chosen Proportion for Grid 1

From the conducted analyses, Grid 1 was observed to perform better than Grid 3. Subsequently, when varying the proportions of Grid 1, proportions b:h = 9:5, 2:1, 5:2, and 3:1 are found to behave better, while when varying the grid proportion with groove joints, proportions b:h = 5:2, 3:1 have the lowest displacements. Therefore, proportion b:h = 5:2 (Figure 3.31) is finally chosen as a base for the algorithm design.

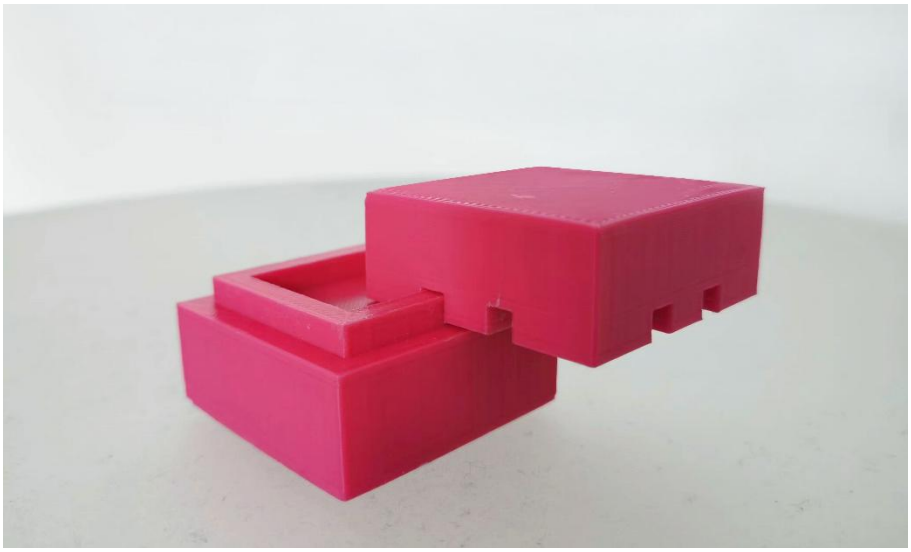


Figure 3.31 *Chosen Grid 1 with Proportion 5:2*

Chapter 4

Algorithm Design

4.1 Algorithm Design Methodology

This section elaborates on the algorithm design for approximating the shell structure. The algorithm is inspired by Selina Bitting's work on reconfigurable domes [34], and is mostly developed within the COMPAS framework coupled with GHPython in order to automatically generate the voussoir geometry for construction. The algorithm design is divided into four phases: 1) Generating a 3D grid system; 2) Funicular form-finding; 3) Approximating shell structure [35, 36]; 4) Refining the approximated shell structure by subdividing and smoothing the block shapes. These four parts are elaborated in sections 4.3, 4.4, 4.5, and 4.6 respectively, while an overview of the entire procedure can be found in Figure 4.1.

The methodology is preferably user-oriented, meaning that there are some decision variables that users can choose (i.e. the size of modular block, the size of base area, the level of intrados' smoothness. etc). In this case, the outcome of this methodology could be optimally performed from the user's point of view, thus reduce material use and increase structural efficiency to the largest extent.

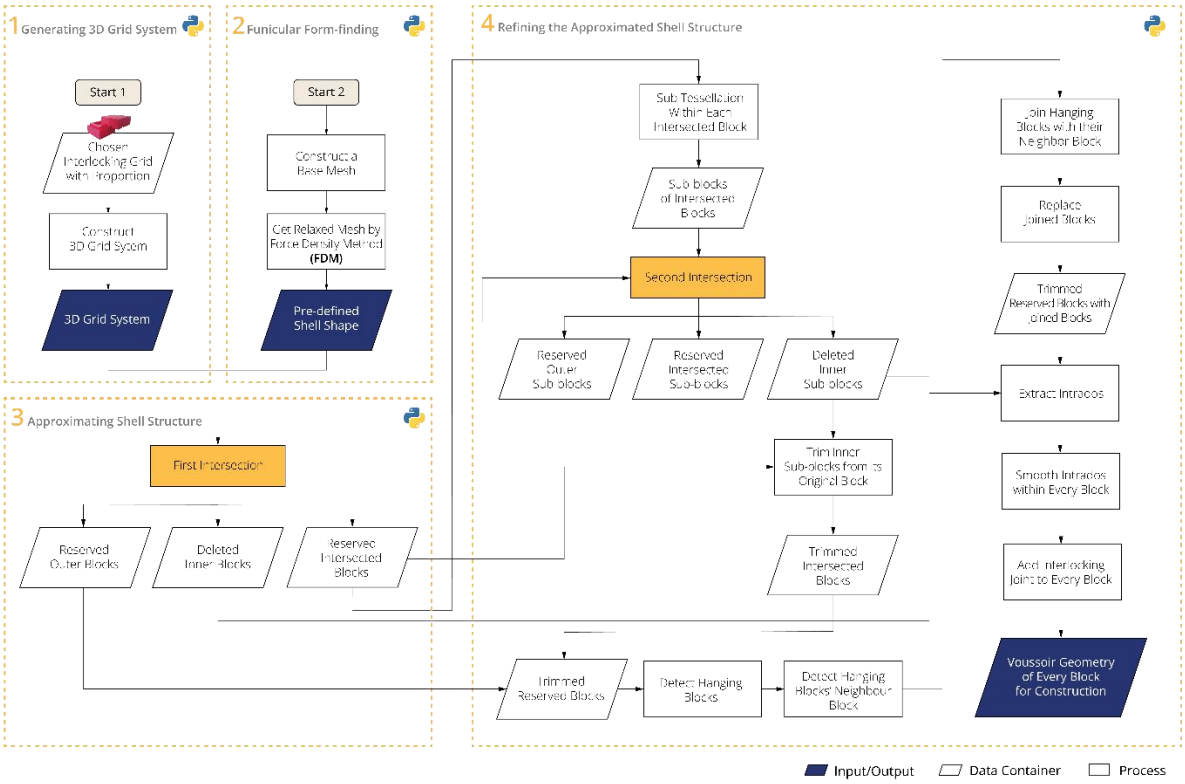


Figure 4.1 *Flowchart of Algorithm Design*

4.2 Defining the Size of Grid

Before the algorithm design, one thing needs to be defined is the size of the chosen grid. This is a decision variable for the users to determine the size of block, and the size settled here is under careful consideration to provide as a reference for the users, however, users can also change the size of block to their preference.

The first consideration of the size of block is the weight of a modular block. The aim of this project is to generate voussoir geometry for easy construction, therefore, the weight of a block can determine if it is easy for a man to carry, thus influence the extent of convenience during the construction process. Starting with Grid 1 with a proportion of 5:2, the 3D grid system can be constructed by first specifying the block size as a user-defined input. In this case, block dimensions of 150mm in width and 60mm in height are chosen to ensure that the block is light enough (in this case about 2.43kg considering the hypothetical density of normal masonry material of 1800kg/m³) to be lifted by a human being (Figure 4.2(a)).

The second consideration is its effect on the size of modular housing unit. By considering the normal wall thickness of a housing unit as 300-450mm, the wall thickness can be easily achieved by two to three modular blocks with the size of 150mm (Figure 4.2(b)).

The third consideration is the staircase for housing unit. In order to create a comfortable riser for staircase, the common value for a riser of 180mm can be easily achieved by three of the modular blocks (Figure 4.2(c)).

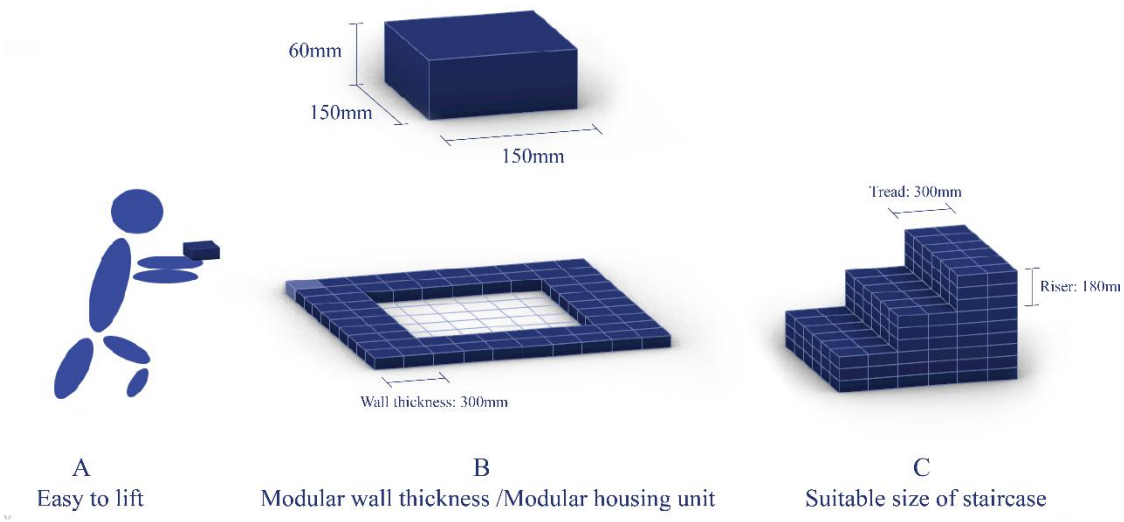


Figure 4.2 *Size of block and its considerations*

4.3 Step 1: Constructing a 3D Grid System

The first part is to construct a 3D grid system from the chosen grid and this grid system serves as the first input for the third phase of shell structure approximation. From the chosen Grid 1 with a proportion of 5:2, the 3D grid system can be constructed by first specifying the block size as a user-defined input. As decribed in the previous sub-section, the size of a modular block is settled to be 150mm*150mm*60mm. In this process, a basic voxel grid is first constructed in GH_CPython by defining the number of grids in three directions, then by selecting the centre point for each block, the 3D grid system can be built by constructing vectors at every centre point in GHPython components in Grasshopper (Figure 4.3).

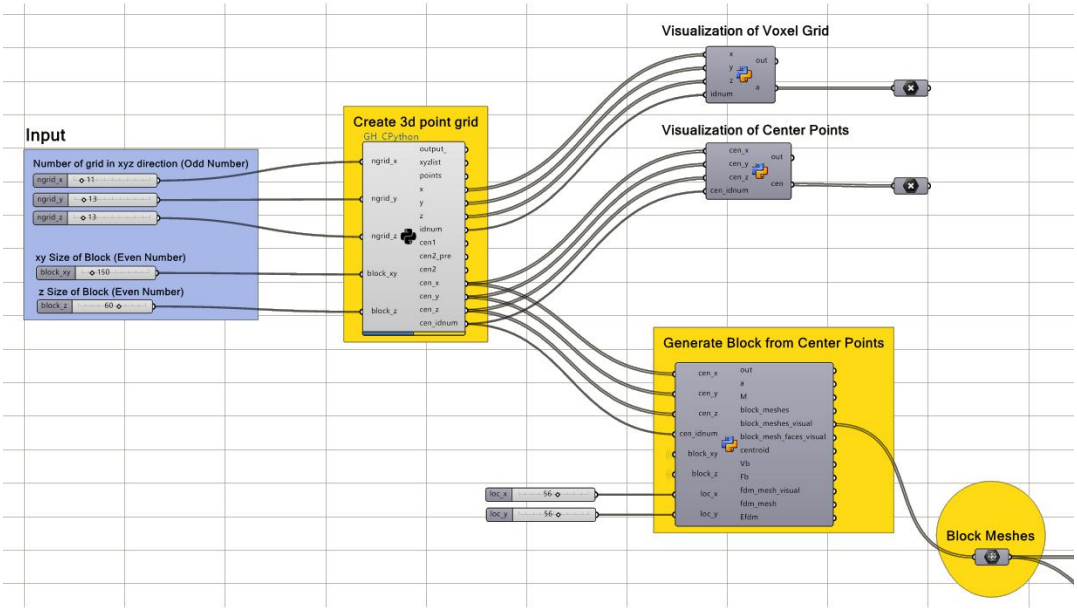


Figure 4.3 *Generating a 3D Grid System in Grasshopper Python Component*

4.2.1 Generating Voxel Grid

The first step to generate a 3D gird system is to generate a voxel grid. The voxel grid is a regular grid that can form a block when several of them are combined together (in this case, 8 voxels can form a modular block). Therefore, grid of voxel size is created according to the input data of number of grids in x, y, z direction (in this case, 11 grids in x direction, 13 grids in y direction, 13 grids in z direction) as well as size of block (in this case, 150mm in x and y direction, 60mm in z direction) (Figure 4.4). The input data are the variables that the users can define themselves, meaning that they can change the size of block as well as change the number of grids.

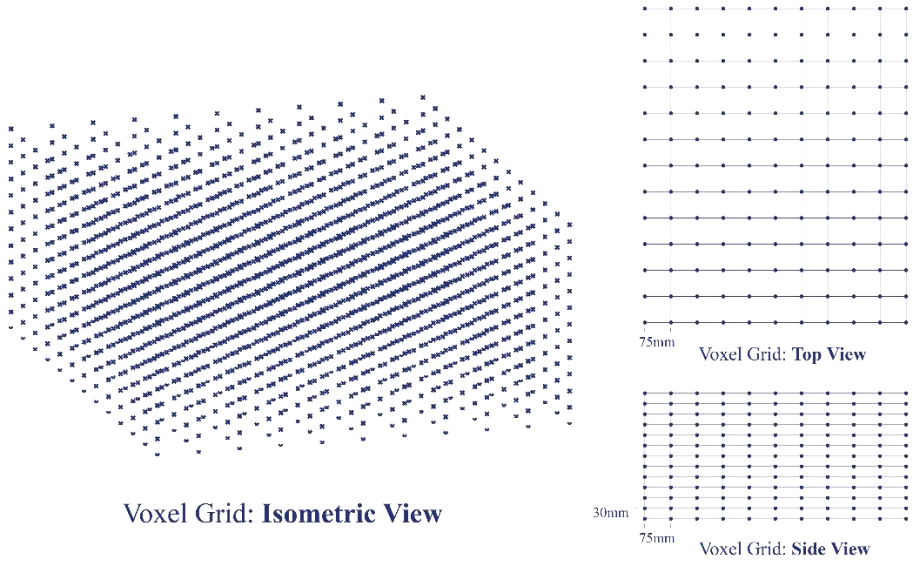


Figure 4.4 *Generating Voxel Grid*

4.2.2 Pick Centre Point for Each Block

After generating the voxel grid, an essential step is to find the centre point of each block. According to the chosen grid in Chapter 3, the upper block relies on one-quarter of the lower block, therefore, the centre point changes regularly due to the layer change. By finding the regularity of change of centre points by layer change, the centre points can be easily selected (Figure 4.5). Assuming that the layering of block start from the left corner of the grid, the center point in the first layer will locate on every second of even number grid, while the center point in the second layer will locate on every second of odd number grid.

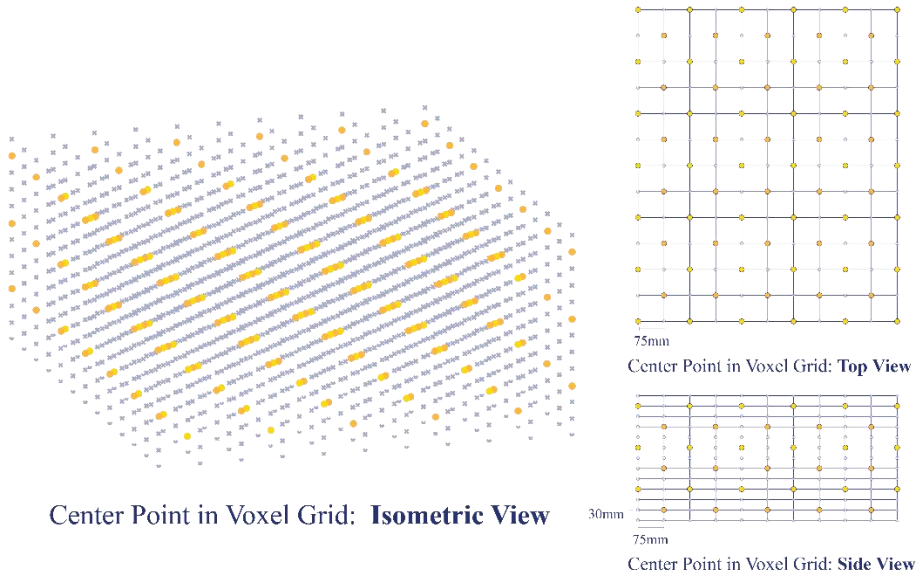


Figure 4.5 *Selecting Centre Points for Modular Blocks*

4.2.1 Generating 3D Grid System

After obtaining all the centre points of modular blocks, the 3D grid system can be thus generated. First, a modular block of size 150mm*150mm*60mm is generated using COMPAS framework, then the centre point of the constructed modular block is obtained. Afterward, by constructing vectors from the centre point of the constructed modular block to every centre point selected in the previous step (Figure 4.6), the constructed modular block can be transformed to the grid system, and thus the 3D grid system is generated (Figure 4.7). An overview of pseudo code for generating the 3D grid system can be seen in Algorithm 1.

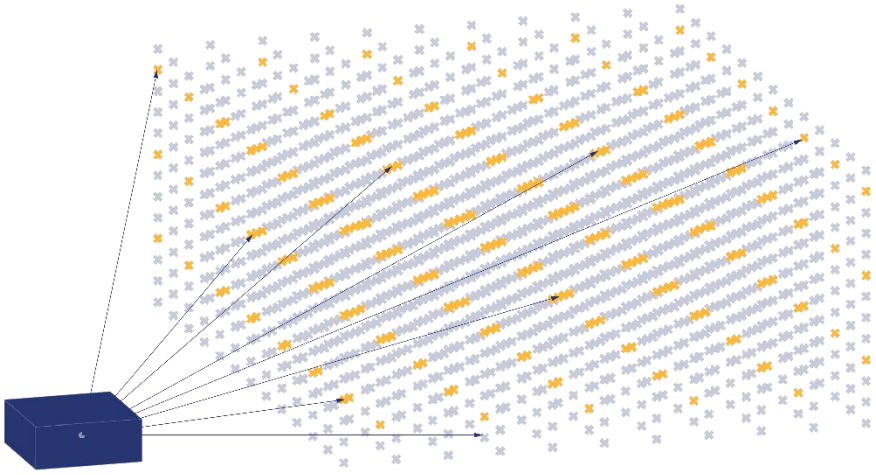


Figure 4.6 *Constructing vectors to every centre point of modular blocks*

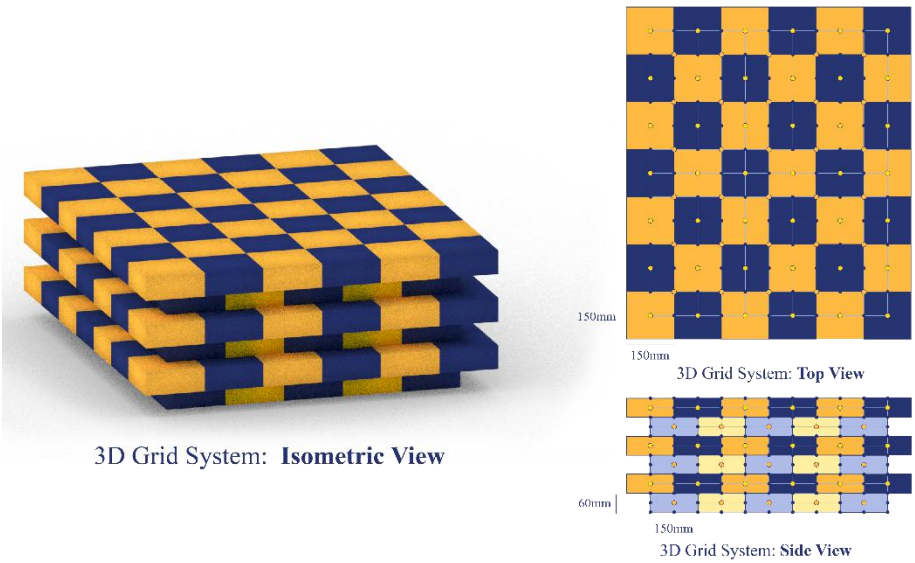


Figure 4.7 *Generating 3D Grid System*

Input: 1. ngrid_x, ngrid_y, ngrid_z #Number of grids in three directions

2. block_xy, block_z #Size of block in xy direction and in z direction

Output: 3D_grid_system #A 3D grid system consisting of modular blocks

```
# Constructing basic voxel grid
voxel_points = [0 to ngrid_x, 0 to ngrid_y, 0 to ngrid_z]
voxel_points_with_size = [voxel_points_x*block_xy/2, voxel_points_y*block_xy/2, voxel_points_z*block_z/2]
# Extract centre point for every modular block
cen1 = voxel_points_with_size [(voxel_points_x%2==1) & (voxel_points _y%2==1) & voxel_points _z%4==1)]
cen2 = voxel_points_with_size [(voxel_points_x%2==0) & (voxel_points _y%2==0) & (voxel_points _z%4==3)]
cen = cen1+cen2
# Transfer chosen grid to every block by constructing vectors
block = Block (block_xy, block_xy, block_z)
block_centroid = block.centroid()
for i in range (cen_number):
   trans_vector = Vector(cen_x[i]- block_centroid_x, cen_y[i]- block_centroid_y, cen_z[i]- block_centroid_z)
   3D_grid_system = block.transformed(trans_vector)
```

Algorithm 1 *- Pseudo Code for constructing 3D grid system*

4.4 Step 2: Constructing a Predefined Shell Shape by Form-finding Process

The second step is to generate a predefined shell shape using form-finding methods. For the form-finding process, several form-finding methods are researched in the literature review, namely Force Density Method, Dynamic Relaxation and Thrust Network Analysis. Among the form-finding methods, the Force Density Method [37] is chosen due to its multiple advantages: it requires less prescribed quantities, is easy to control, and can tackle compression-only structures efficiently.

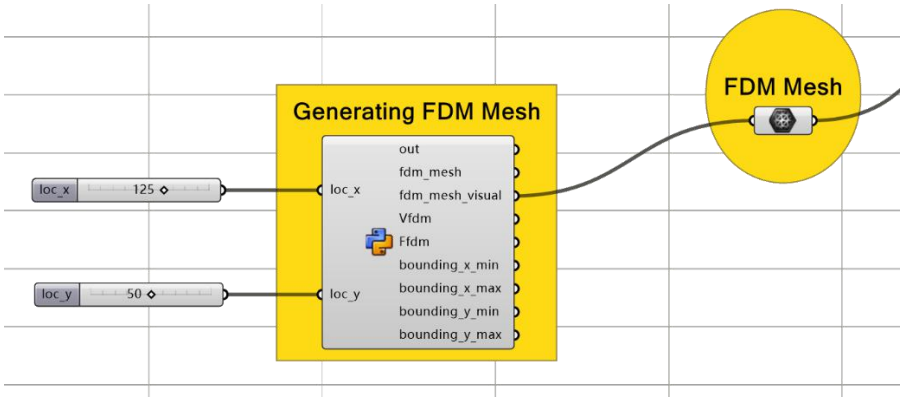


Figure 4.8 *Generating a Relaxed Mesh in Grasshopper Python Component*

To that end, a base mesh is first created within COMPAS, then by using the Force Density Method, a relaxed mesh is generated by applying loads to all the vertices and setting q (force densities, i.e. the ratio of the force/thrust over the length of each edge, roughly meaning the same thing as the stiffness of the chosen material) to every edge (Figure 4.9) (Algorithm 2).

However, it is worth noting that the choice of form-finding method is another decision variable that users can control, meaning that they can choose the most suitable form-finding method of their preference to generate a form-active shape themselves. Other decision variables within this step are the size of the base mesh (in this case, a rectangle with size of 800mm*500mm) and the location of the relaxed mesh in the 3D Grid System (loc_x and loc_y in the grasshopper component shown in Figure 4.8).

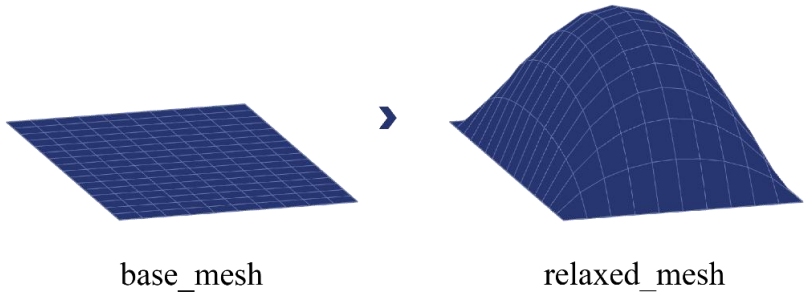


Figure 4.9 *Generating a Relaxed Mesh in Grasshopper Python Component*

Input: base_mesh

Output: relaxed_mesh

```
# Extract coordinates of vertices of base_mesh
vertices, faces = base_mesh. to_vertcices_and_faces()
# Extract edges of base_mesh
edges = base_mesn.edges()
# Find boundary vertices
boundary_vertices = base_mesh.vertices_on_boundary()
# Set loads(30kN)
loads = [[0,0,30]] * (len(vertices))
# Prescribed force densities in the edges (1kN/m³)
qpre = [1] * (len(edges))
# Generate relaxed mesh
xyz, q, f, l, r = compas.numerical.fd_numpy(vertices, edges, boundary_vertices, qpre, loads)
relaxed_mesh = compasmesh.from_vertices_and_faces(xyz, faces)
```

Algorithm 2 *– Pseudo Code for generating relaxed mesh*

4.5 Step 3: Approximating Shell Structure

4.5.1 Intersection between the Generated 3D Grid System and Predefined Shell Structure

In this phase, the predefined shell shape is approximated using the method of topological polyhedralization (i.e. finding the minimal set of necessary and sufficient blocks to represent the topology of the predefined shell shape obtained in the last step, using its wireframe network as a reference, as explained in [35] and [38]) with the generated modular blocks by intersection. The goal of this phase is to roughly obtain the shell with a combination of interlocking modular blocks so that the structure is self-supporting during the construction process. Therefore, in the intersection process, all the modular blocks intersecting with the predefined shell mesh are selected to achieve the rough approximation.

The intersection algorithm is conducted by using grasshopper component "mesh | mesh". When one block is intersected with the relaxed mesh, the intersection component allows to detect the intersection lines in between. Afterwards, "list length" component will transfer the lines intersected into a list which tells how many lines does each block has if it is intersected with the relaxed mesh. By a component called "Boolean", which could basically transfer wherever there is a number in a list into Boolean values like "True" or "False", a new list could be obtained with Boolean values representing if a block is intersect with the relaxed FDM mesh. By deleting the blocks in the list who has a "False" value, a new list with blocks with "True" values could be achieved representing blocks that are intersecting with the form-finding mesh (Figure 4.10).

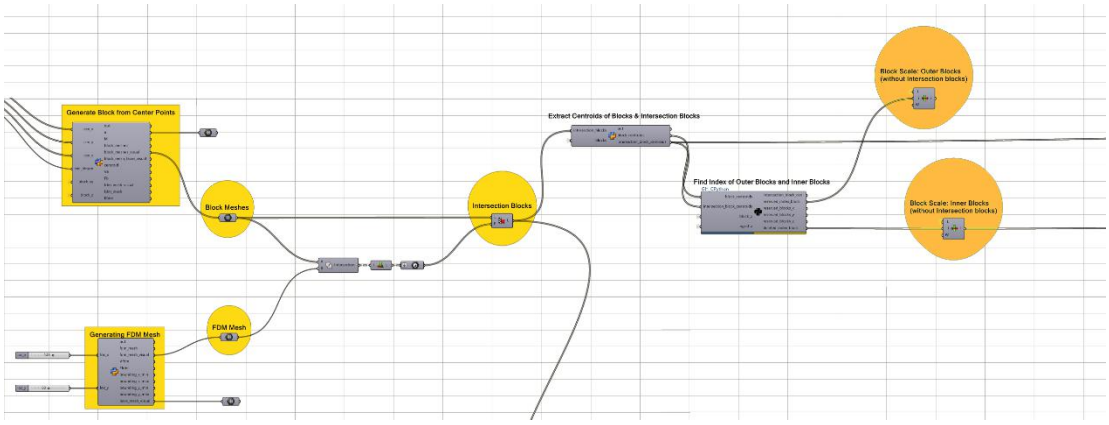


Figure 4.10 *Shell Structure Approximation in Grasshopper Python Component*

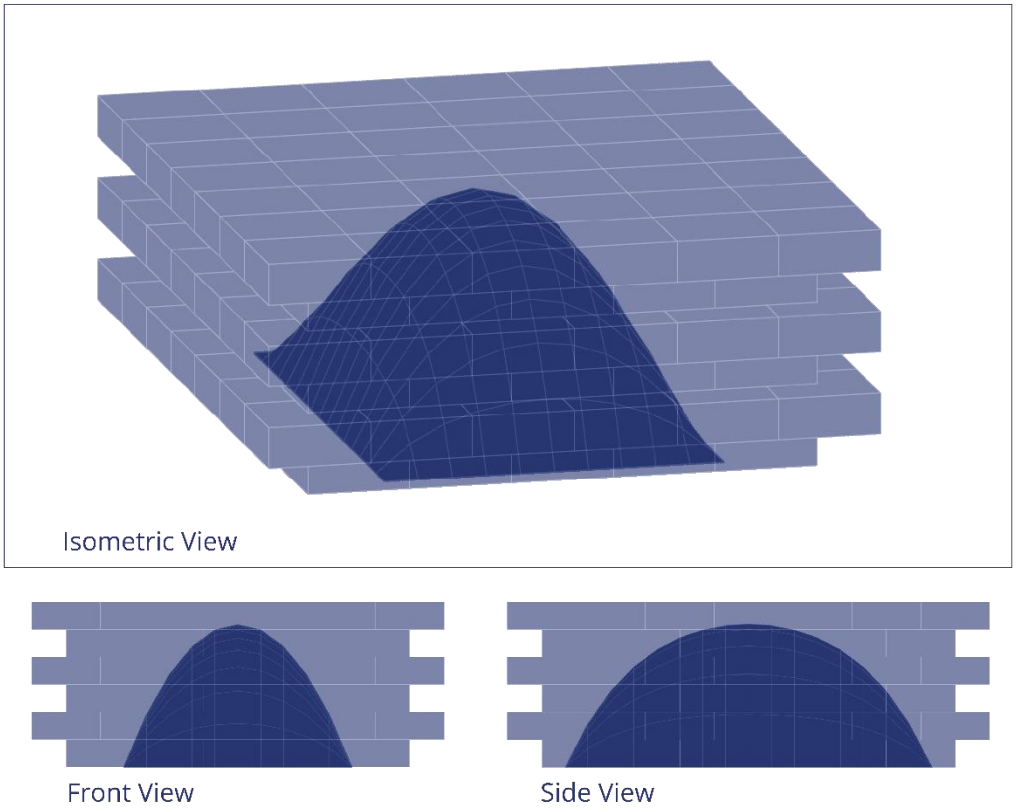


Figure 4.11 *3D Grid System and Predefined Shell Structure before intersection*

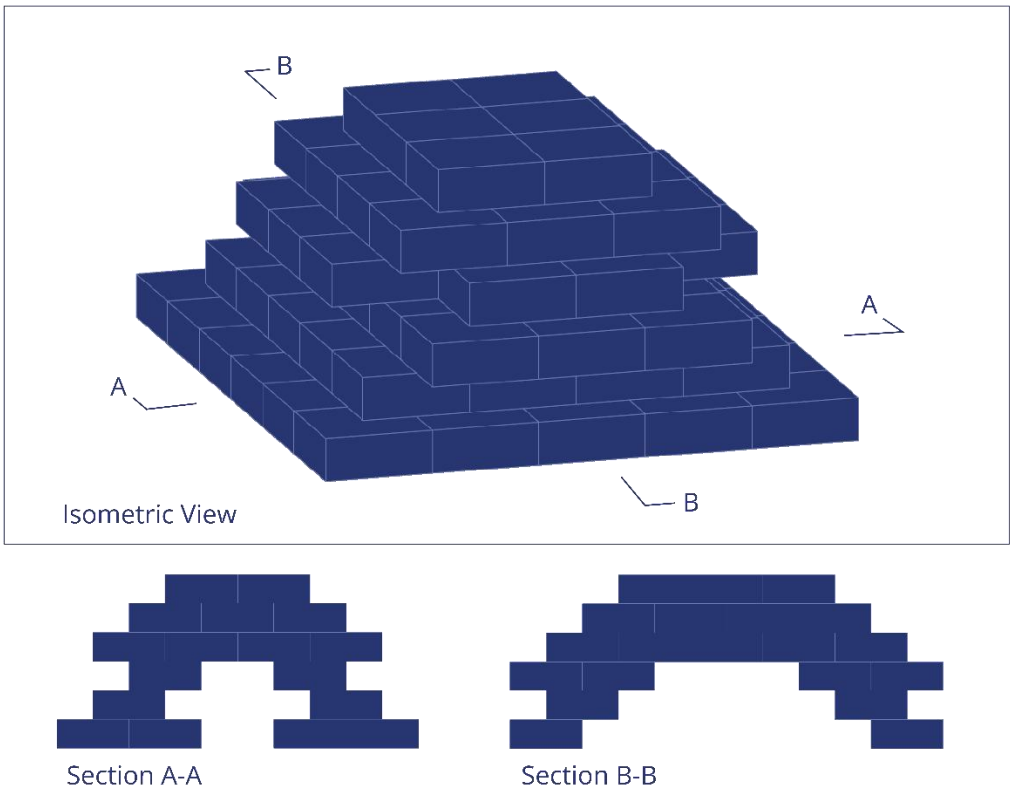


Figure 4.12 *Approximated blocks after Intersection*

4.5.2 Selection of Outer Blocks, Intersected Blocks and Inner Blocks

In the previous step, the modular blocks intersected with the relaxed mesh were selected. However, outer blocks also need to be retained to ensure a flat surface on the top of the vault for the construction of the second floor. Consequently, the outer blocks and inner blocks are respectively selected within this step. By identifying the blocks within the range of the form-found/relaxed mesh, blocks within this range can be selected. Afterward, inner blocks can be obtained by deleting the intersected blocks and outer blocks can thus be obtained by deleting both inner blocks and intersected blocks (Algorithm 3). Finally, inner blocks are removed because they are not conducive to the structure when the vault structure is already predefined by the form-finding process. Outer blocks and intersected blocks are reserved for further refinement of the shape of the voussoir components (Figure 4.14).

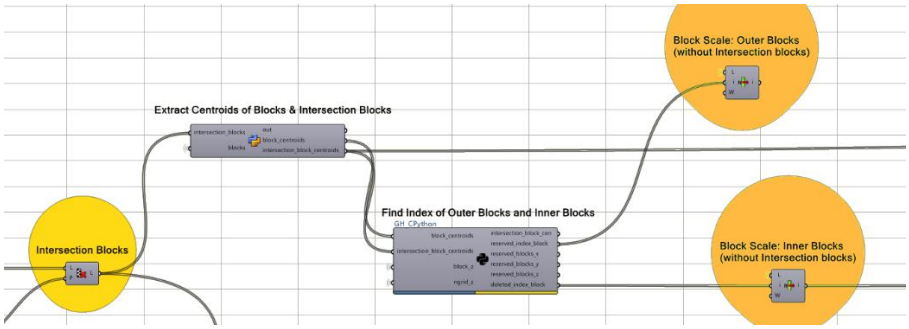


Figure 4.13 *Selection of Outer Blocks, Intersected Blocks, and Inner Blocks in Grasshopper Python Component*

Input: blocks, intersected_blocks
Output: outer_blocks, intersected_blocks, inner_blocks
Pick the centroids of blocks inside "Relaxed Mesh" inside_block_include_intersected_blocks = [] **for** p **in** blocks: **if** relaxed_mesh.IsPointInside(p,0,False) = True: inside_block_include_intersected_blocks.append(p) # Pick Inner Blocks by deleting the Intersected Blocks inner_blocks = np.delete(inside_block_include_intersected_blocks, intersected_blocks) # Pick Reserved Blocks (Outer Blocks + Intersected Blocks) by selecting reversed index from all the blocks mask = np.ones(len(blocks),bool) mask[index_inner_blocks] = 0 reserved_block_include_intersected_blocks = blocks [mask] # Pick Outer Blocks by deleting the intersected blocks outer_blocks = np.delete(reserved_block_include_intersected_blocks, intersected_blocks)

Algorithm 3 *– Pseudo Code for selecting outer blocks, intersected blocks, and inner blocks*

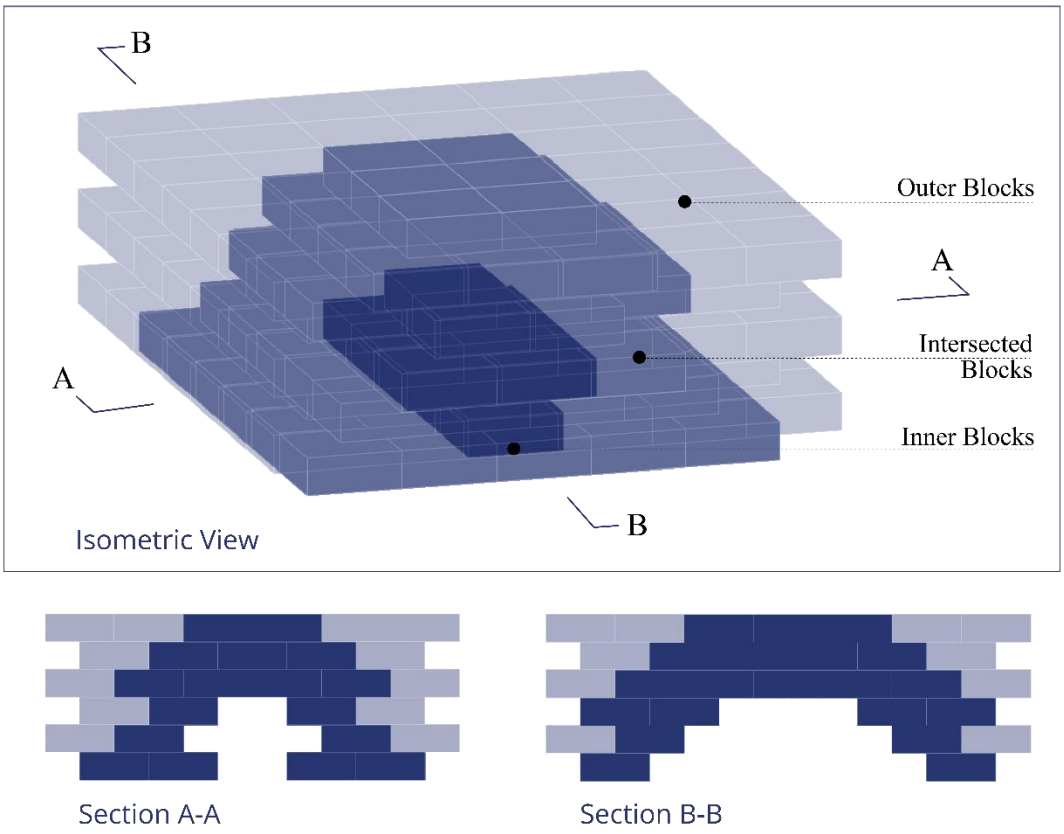


Figure 4.14 *Outer Blocks, Intersected Blocks, and Inner Blocks*

4.6 Step 4: Refining the Approximating Shell Structure

After the process of shell structure approximation, the last step involves the optimization of the intrados (Figure 4.15) by diversifying block types on a higher resolution voxel grid. When different shapes of vaults are generated from topologically different tessellations and the same settings for the form-finding process as input, the blocks for intrados can be distinct and diverse.

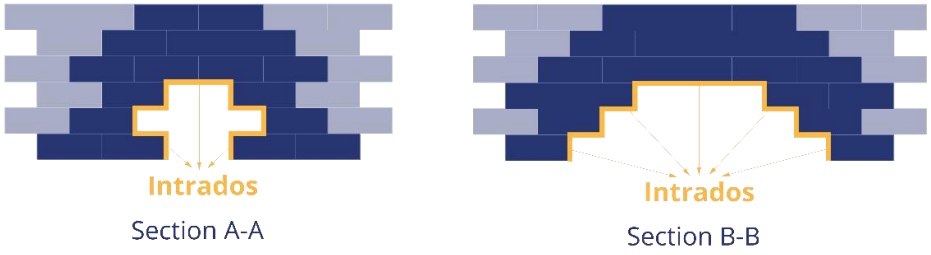


Figure 4.15 *Intrados for optimization*

4.6.1 Sub-Tessellation Within Every Intersected Block

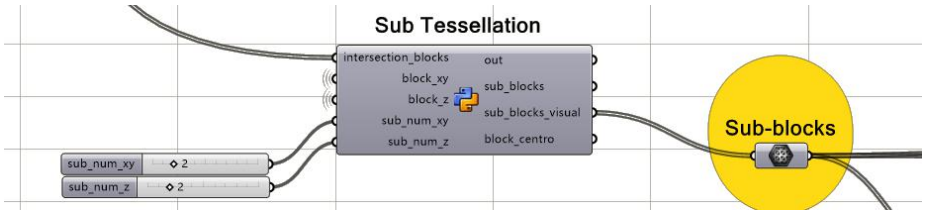


Figure 4.16 *Sub -Tessellation Process in Grasshopper Python Componenet*

Even though a stacking of the rough blocks might be acceptable as an approximated shell structure, the size of a modular block is usually large and thus not suitable to achieve an aesthetic and elegant intrados for people to live in. To break the restriction of the size of the block, the sub-tessellation process provides the opportunity to refine the intrados at a higher resolution of the grid. By subdividing each block along every edge into equal number of segments, each modular block can be sub-tessellated into smaller sub-blocks (efficiently in a deeper layer of an octree voxelated space). In the example shown in Figure 4.17, a modular block is respectively sub-tessellated into 8 sub-blocks in total (i.e. 2 segments along each edge/ a one level deep octree) and 64 sub-blocks in total (i.e. 4 segments along each edge/ a two levels deep octree). To achieve sub tessellation within every single block, a “VolMesh” grid is first created within COMPAS framework, then sub-blocks can be generated by construct vectors to every intersected block (Algorithm 4). In the following steps, sub-tessellation of level 2 (64 sub-blocks) are used as an example.

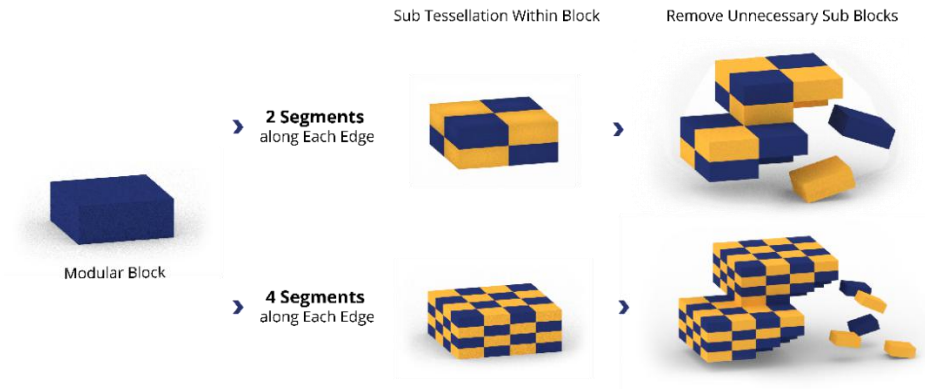


Figure 4.17 *Sub-Tessellation Within Each Modular Block*

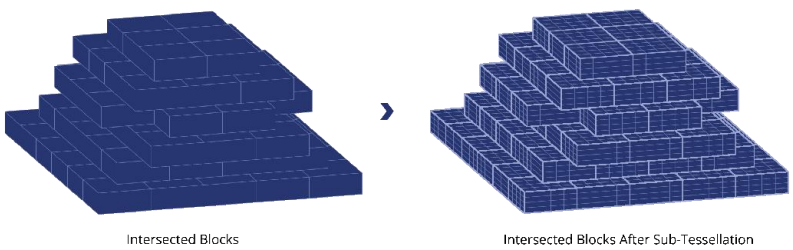


Figure 4.18 *Intersected Blocks after Sub-Tessellation*

Input: 1. intersected_blocks

2. sub_num_xy,sub_num_z #the level of sub-tessellation

Output: outer_blocks, intersected_blocks, inner_blocks

```
# create general sub-tessellation grid
sub_mesh = VolMesh (block_xy/sub_num_xy,block_xy/ sub_num_xy, block_z/ sub_num_z)
sub_centro = sub_mesh.centroid()


# transfer sub-blocks to every intersection blocks
sub_blocks = [] #the final sub-blocks in every intersected block
for i in range (len(intersected_blocks)):
    trans_vector = Vector.from_start_end(sub_centro, intersected_blocks_centro[i])
    T = Translation.from_vector(trans_vector)
    M = submeshgrid.transformed(T)
    for j in range (sub_num_xy*sub_num_xy*sub_num_z):
        sub_block = VolMesh.cell_to_mesh(M,j)
        sub_blocks.append(sub_block)
```

Algorithm 4 *– Pseudo Code for sub-tessellation process*

4.6.2 Intersection between the Sub-blocks and Predefined Shell Structure

In the previous step, sub-blocks are obtained. These sub-blocks are again intersected with the same predefined shell shape following the same process as in the shell structure approximation process in Section 4.5.1, three kinds of sub-blocks then can be categorized within the intersected blocks, respectively outer sub-blocks, intersected sub-blocks, and inner sub-blocks. and the sub-blocks that do not intersect, as well as within the range of predefined shell shape (inner sub-blocks), are selected to be removed in the next step.

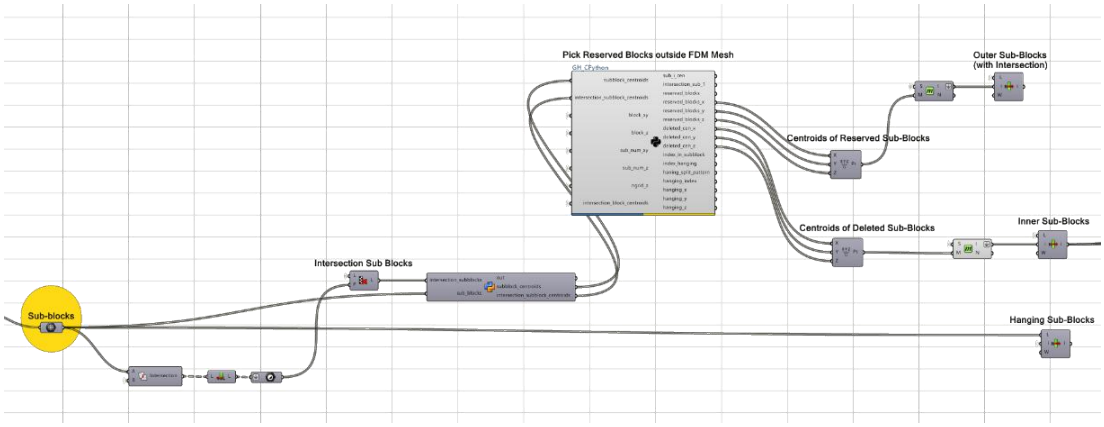


Figure 4.19 *Intersection process with Sub-blocks in Grasshopper Python Component*

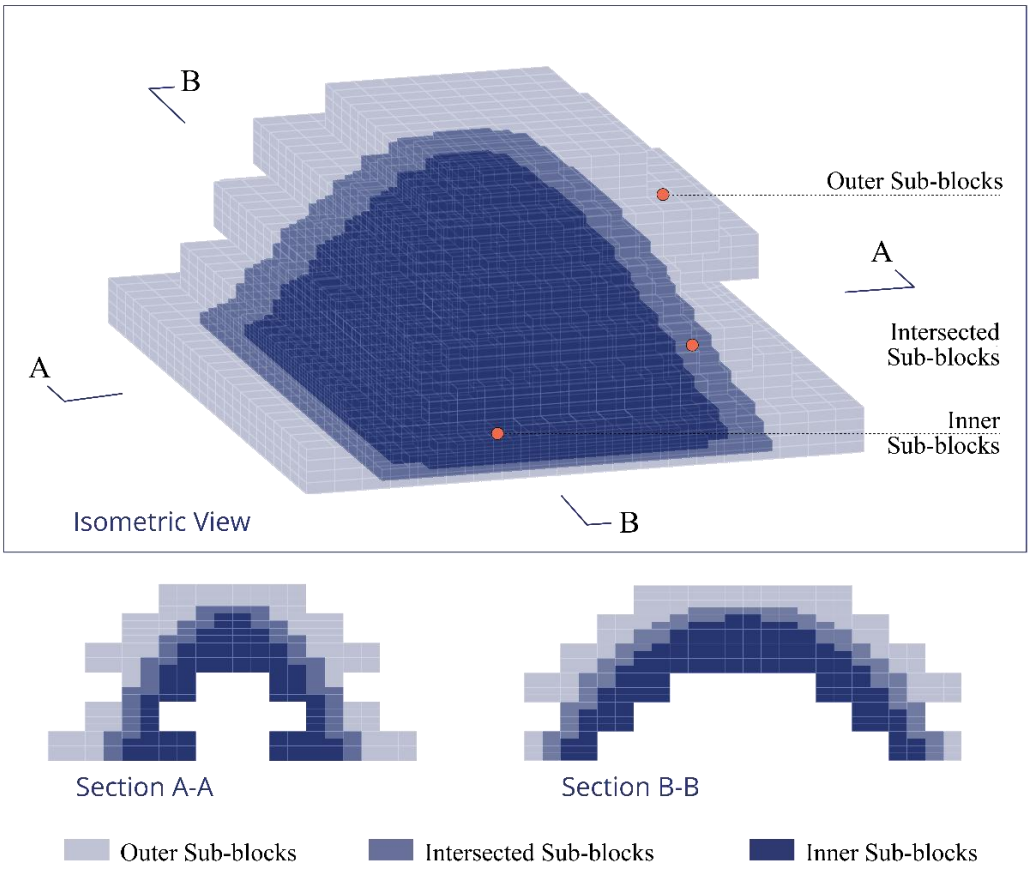


Figure 4.20 *Outer Sub-blocks, Intersected Sub-blocks, Inner Sub-blocks,*

4.6.3 Trimming of Inner Sub-blocks from Its Original Blocks

From the previous step, the intersected sub-blocks already have a higher resolution of approximation (Figure 4.21), so the unnecessary sub-blocks need to be deleted from its original blocks. Therefore, a trimming process (Figure 4.22) is conducted to cut these sub-blocks from their original blocks, thus returning a customized block for the intrados while the outer blocks are kept generic. This process removes the unnecessary sub-blocks, keeping the same topology, but different geometry of the original blocks.

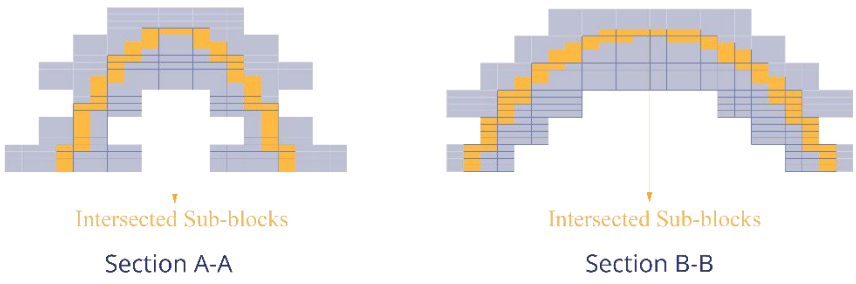


Figure 4.21 *Approximated Sub-blocks in a higher resolution*

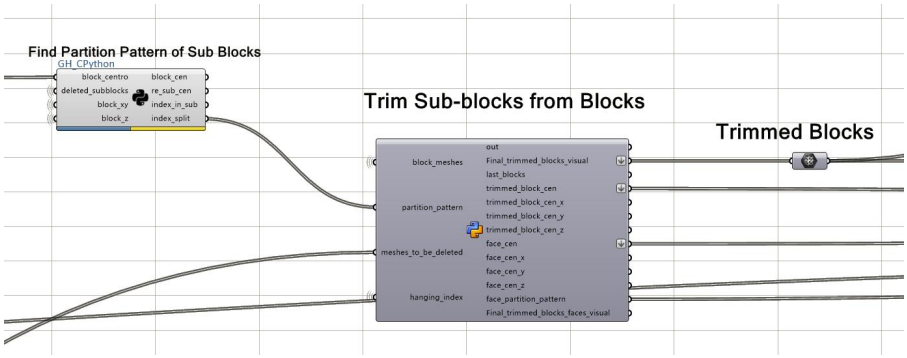


Figure 4.22 *Trimming Process in Grasshopper Python Component*

With the previous step, the inner sub-blocks are obtained. By finding the index, it is easy to know which inner sub blocks belongs to which blocks. Next, by first joining the sub-blocks need to be trimmed within every intersected block, these sub-blocks can be removed from their original blocks easily within the trimming process (Algorithm 5).

Input: sub_blocks_to_trim, intersected_blocks

Output: Trimmed_blocks

```
# Join sub-blocks need to be trimmed within every intersected blocks
def joined_sub(block_id):
  if len (sub_blocks_to_trim[block_id])>1:
        union_sub = ghc.MeshUnion(sub_blocks_to_trim[block_id])
  if len (sub_blocks_to_trim[block_id]) =1:
        union_sub = sub_blocks_to_trim[block_id][0]
  return union_sub

# Trimming sub-blocks from every intersected blocks
def trimmed_block(block_id):
  trimmed_block = ghc.MeshDifference(intersected_blocks[block_id], joined_sub(block_id))
  return trimmed_block

# Trimming Process
trimmed_blocks = []
for i in range (len(intersected_blocks)):
  trimmed = trimmed_block(i)
  trimmed_blocks.append(trimmed)
```

Algorithm 5 *– Pseudo code for trimming process within each intersected block*

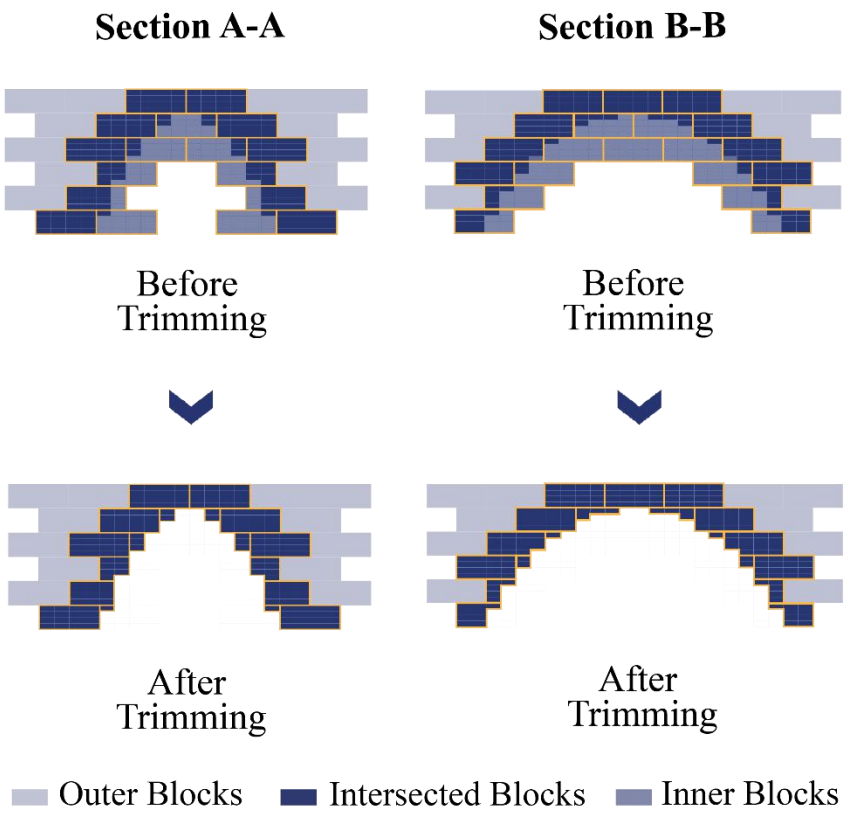


Figure 4.23 *Intersected Blocks Before Trimming and After Trimming*

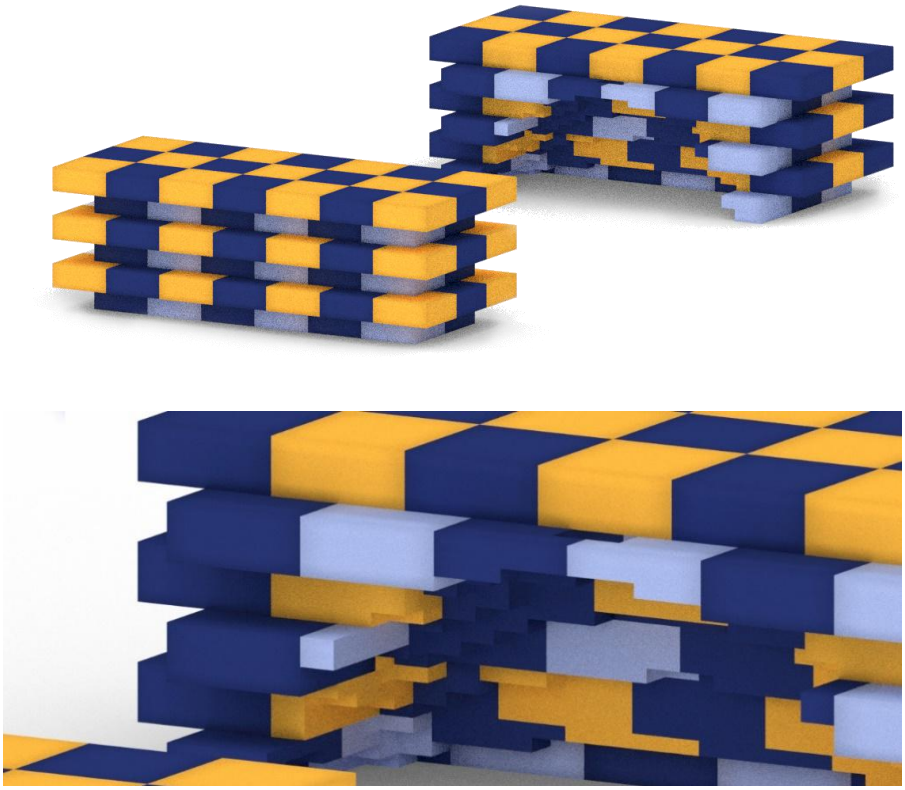


Figure 4.24 *Intersected Blocks After Trimming Process*

4.6.4 Dealing with Hanging Blocks

After the trimming process, a problem still needs to be solved: there are blocks without feet to stand on, which means they are hanging in the air. To deal with this issue, these blocks are firstly detected by checking if they have bottom sub-blocks (if all the bottom sub-blocks within the block are deleted) and if there is at least one-quarter of contact surface underneath (there is at least one foot-block below) (Algorithm 6), then these blocks are going to find their neighbour blocks by choosing their closest block on the same layer. By joining these hanging blocks and their neighbour blocks, the problem can be solved.

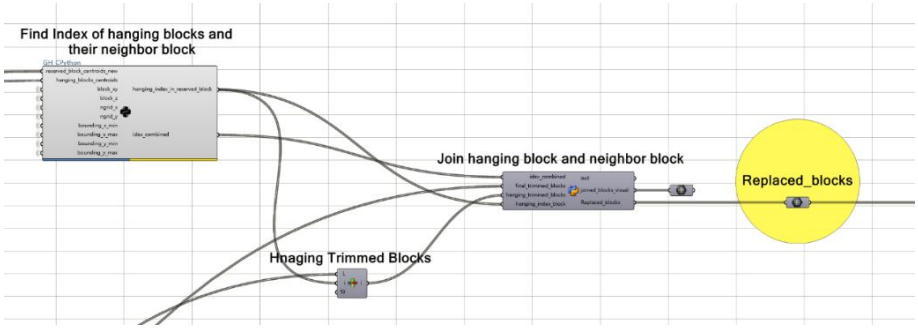


Figure 4.25 *Dealing with Hanging Blocks in Grasshopper Python Component*

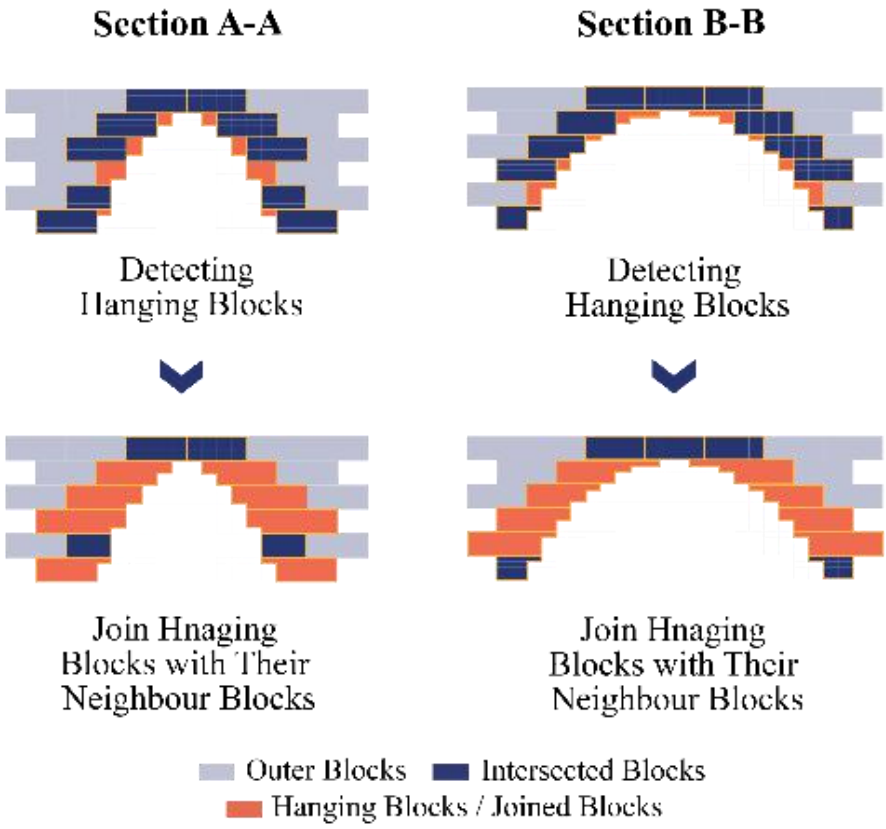


Figure 4.26 *Detecting Hanging Blocks and Join them with their neighbour block*

Input: outer_sub_blocks, intersected_sub_blocks, inner_sub_blocks, intersected_blocks

Output: hanging_blocks

```
reserved_sub = outer_sub_blocks + intersected_sub_blocks

# Detect the Hanging Blocks
for i in range (len(intersected_blocks)):
  overlapping_sub = set(bottom_layer_sub(i)).intersection(reserved_sub)
    # Situation 1: if there are no bottom sub-blocks within every intersected block
  if len(overlapping_sub) = 0:
      hanging_blocks.append(i)
  # Situation 2: if there are at least one-quarter of contact surface underneath
  else:
      range = range0, range1, range2, range3 # 4 square contact area for a block
      # if none of the four square area is full, then this blocks is hanging
      if (range0 is not full)&(range1 is not full)&(range2 is not full)&(range3 is not full):
          hanging_blocks.append(i)
        # if any of the four square area are full, check any of the four range below are full
        else:
          index = range [range is full]
        range_below = range4, range5, range6, range7
        if (range_below is full).any() = False:
            hanging_blocks.append(i)
```

Algorithm 6 *– Pseudo code for detecting hanging blocks*

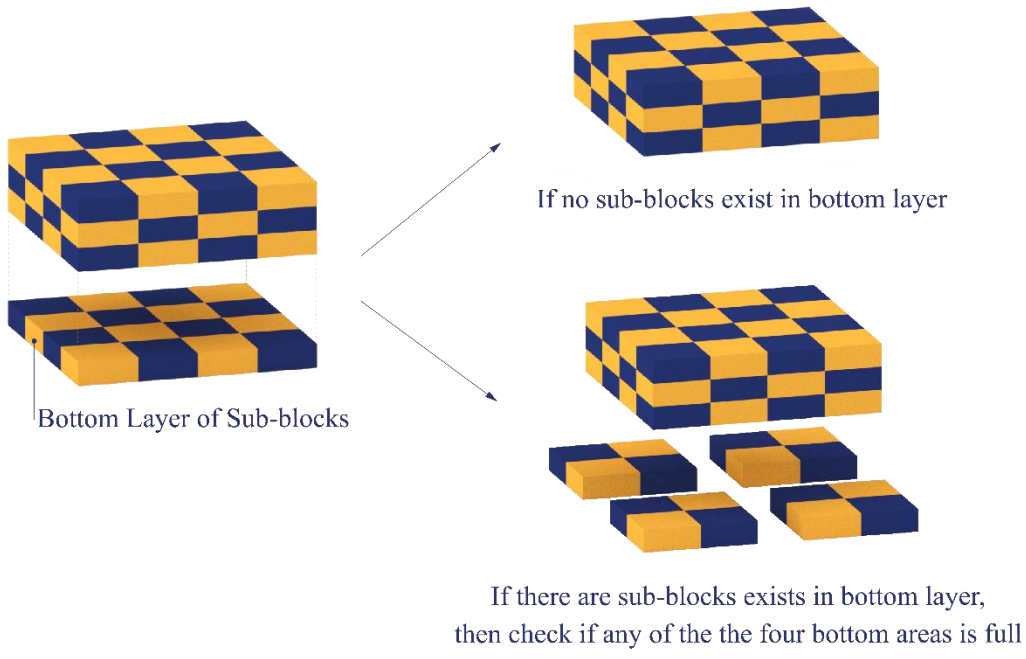
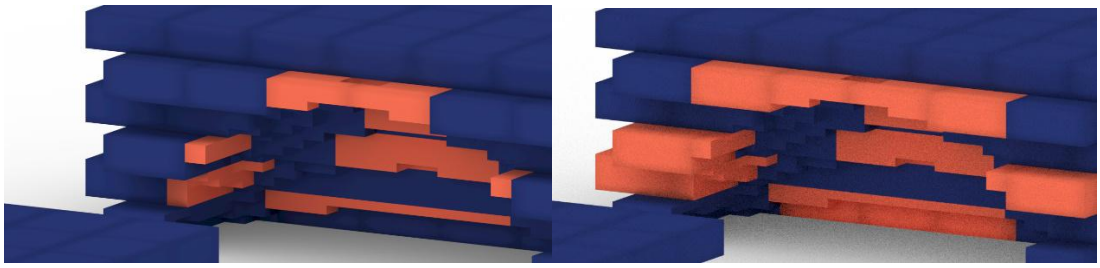


Figure 4.27 *Detecting the Hanging Blocks*



(a) (b)

Figure 4.28 *(a) Detected Hanging Blocks and (b) Joined Hanging Blocks*

4.6.5 Smoothing Intrados for Every Block

Although the process allows obtaining sub-tessellated blocks in a higher resolution of the grid, the trimmed intrados is not ideal yet. To make the intrados as smooth as possible for people to live in as well as to ease the process of manufacturing each block from moulds, a smoothing process is conducted within this step (effectively using the Catmull-Clark subdivision of the outer mesh surface of each block). This process smooths surfaces of intrados within every block facing inwards in order to provide a comfortable inner space. All the inner deleted blocks are gathered together for the selection of the intrados. By detecting if there are overlapping faces between all the deleted inner blocks and reserved outer blocks, the intrados can be finally selected (Algorithm 7). Afterward, this intrados can be returned by every block facing inwards and smoothed (Figure 4.31, 4.32), thus obtaining the smoothed modular blocks (Figure 4.30). Lastly, by adding the interlocking joints, the voussoir geometry for each block can be thus obtained for the construction process.

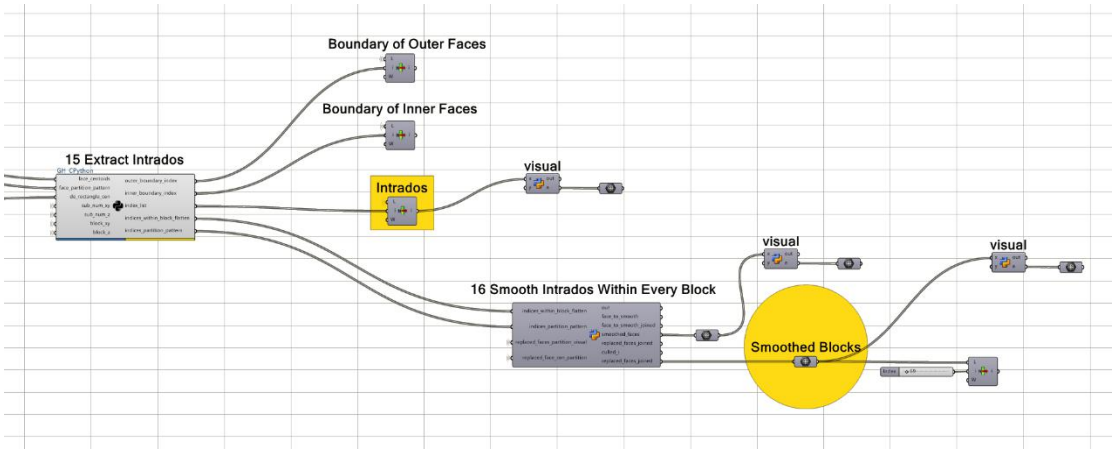


Figure 4.29 *Smoothing Intrados in Grasshopper Python Component*

Input: outer_blocks, trimmed_blocks, inner_blocks, outer_sub_blocks, intersected_sub_blocks, inner_sub_blocks

Output: smoothed_blocks

```
# Gather all the inner deleted blocks
inner_deleted_blocks = inner_blocks + inner_sub_blocks
# Gather all the reserved blocks
final_blocks = outer_blocks + trimmed_blocks

# Extract intrados
face_touched = [x[(x == inner_deleted_blocks_faces).all(2).any(0)] for x in final_blocks _faces]

# Smooth intrados within every final block
smoothed_blocks = []

for i in range (len(final_blocks):
    # if no faces need to be smoothed within the block
  if len(final_blocks[i].faces_to_smooth) == 0:
        smoothed_blocks.append(final_blocks[i])
  # if there are faces need to be smoothed within the block
  else:
        faces_to_smooth_within_block = face_touched [i]
        faces_to_smooth_joined = ghc.MeshJoin(faces_to_smooth_within_block)
        catmull = wb.Common.Subdivision.CatmullClark(faces_to_smooth_joined,3,0)
        face_replaced = ghc.MeshJoin(catmull, rest_faces_within_block)
        smoothed_blocks.append(face_replaced)
```

***Algorithm 7** – Pseudo code for extracting and smoothing intrados*

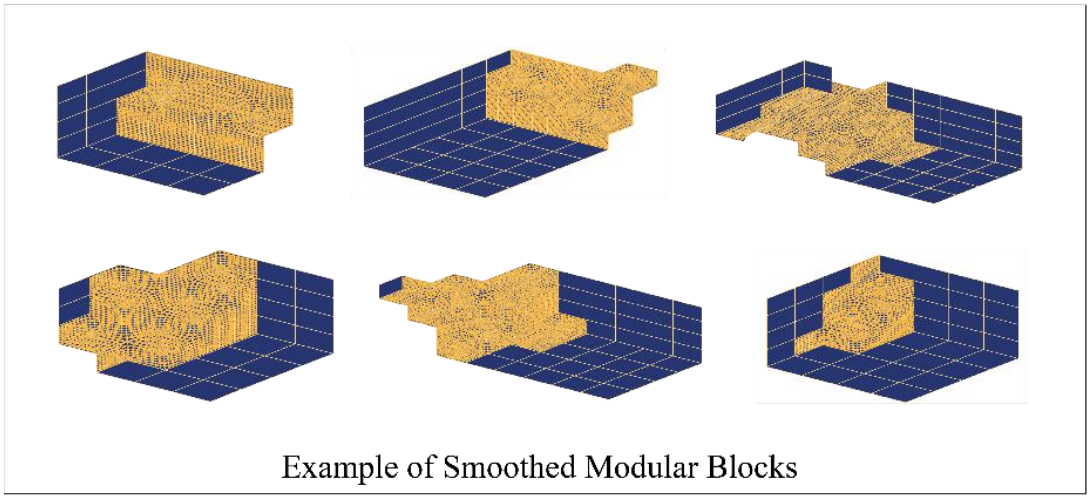


***Figure 4.30** Example of Smoothed Modular Blocks*

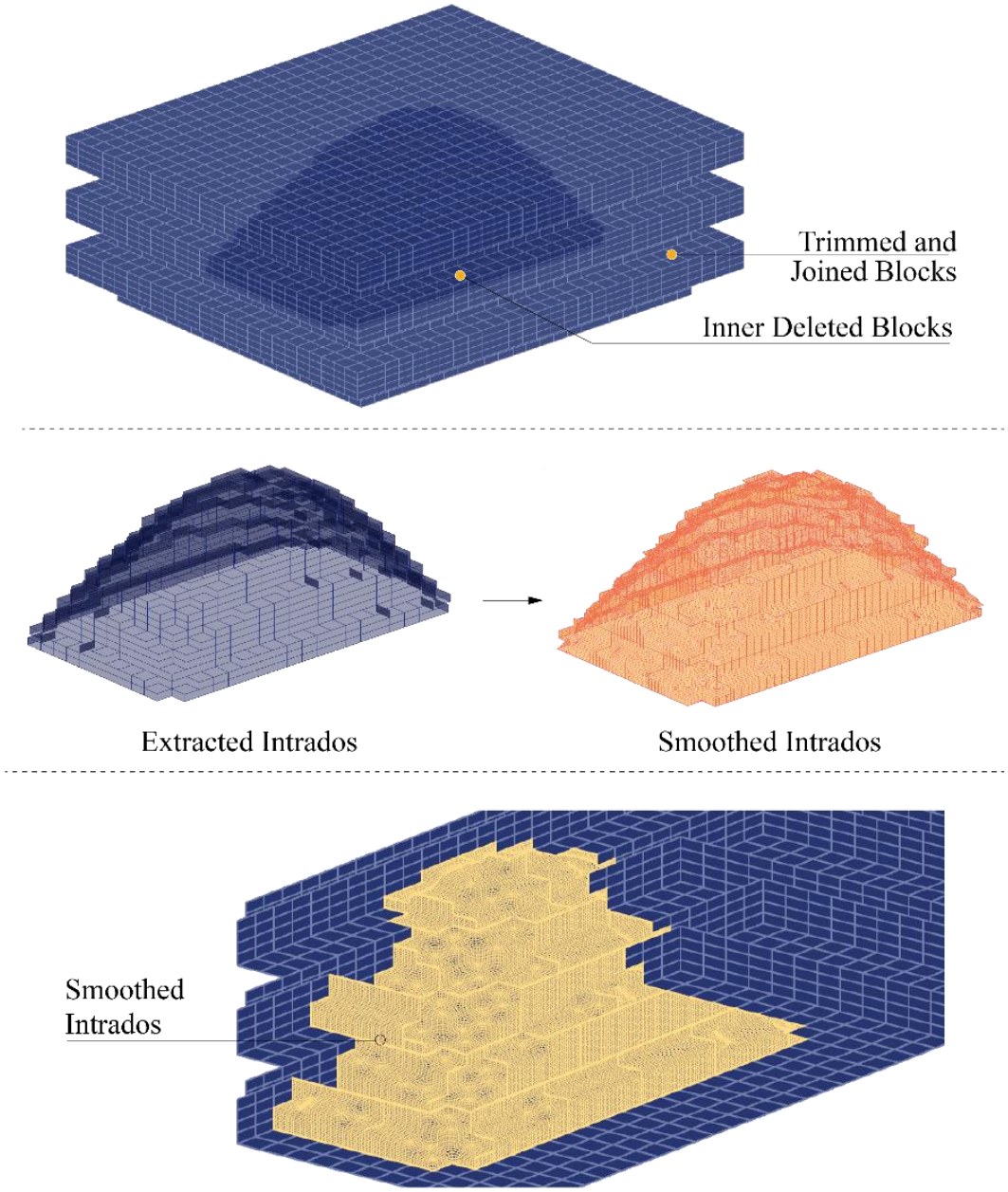


Figure 4.31 *Smoothing Process*

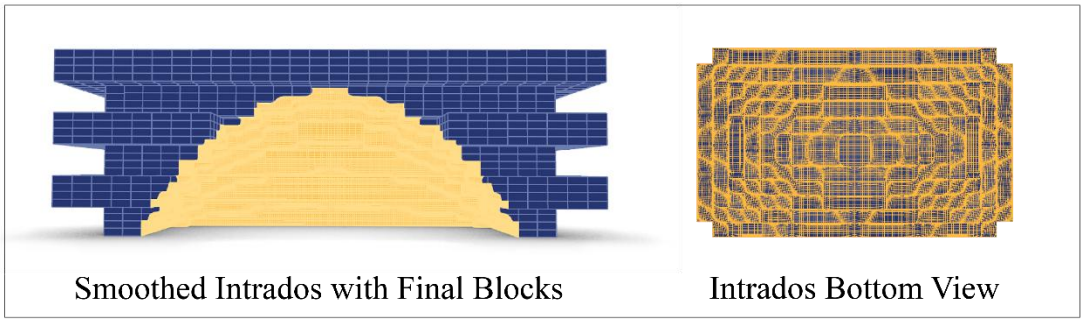


Figure 4.32 *The Smoothed Intrados*

4.6.6 Verification

This sub-section evaluates the results of the algorithm design using physical models by 3D printing. Five layers of a corner of the final structure are printed as shown in Figure 4.33. The pictures show a smooth process of interlocking mechanisms and ensures self-supporting ability, thus verify the results to some extent. However, the 3D printing results doesn't represent the practical model, therefore, a practical verifications process is still needed for the further development of the algorithm.

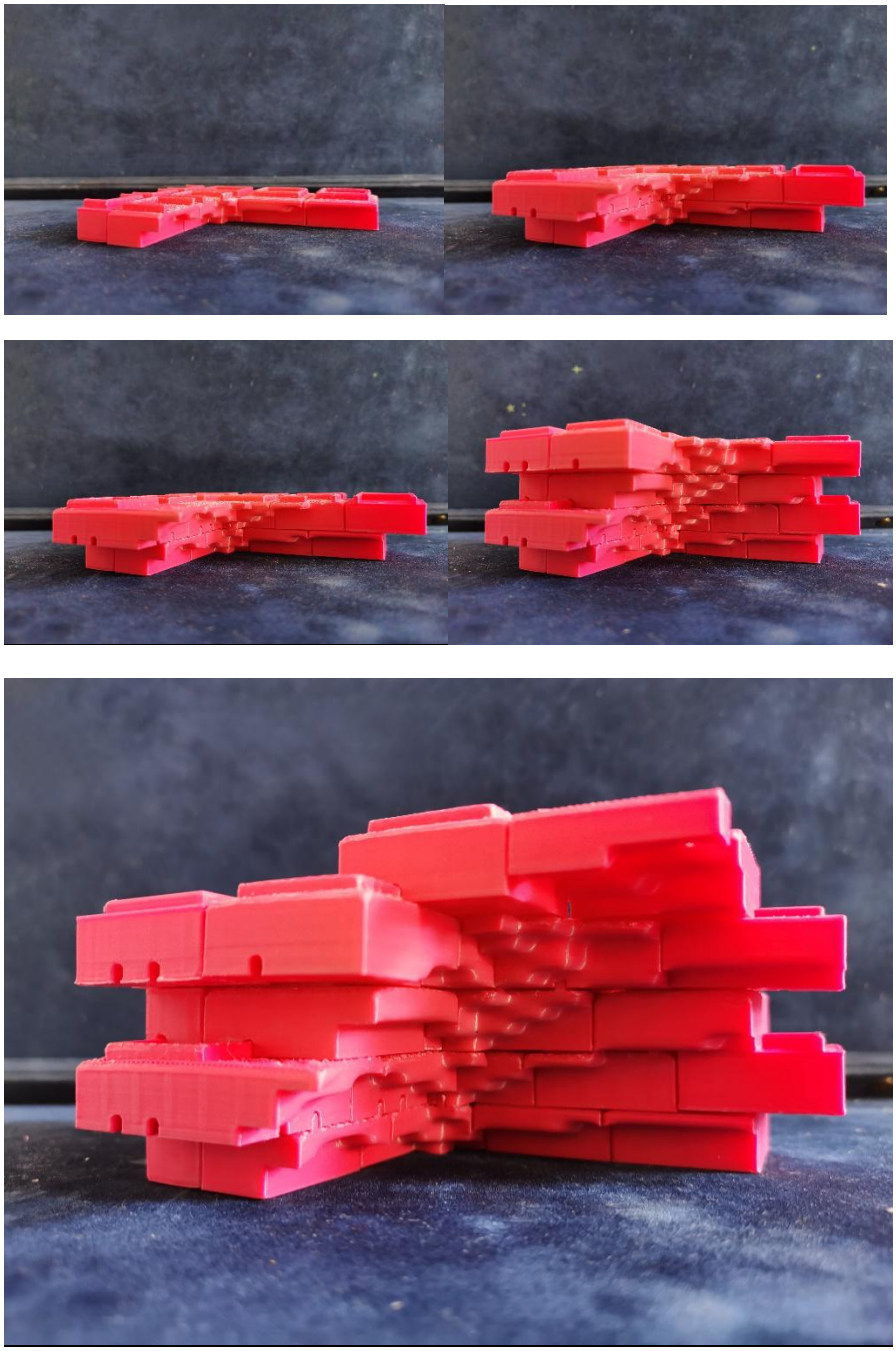


Figure 4.33 *– A corner of 3D printed results*

Chapter 5

Conclusion

5.1 Conclusion

In this paper, a new methodology was presented for automatically generating modular voussoir geometries with interlocking mechanisms for the fabrication of stackable structures, which are also self-supporting during construction, thereby eliminating the need for expensive and temporary formworks and falseworks.

Three types of space-filling grids are firstly designed, then by a general comparison, two grids (Grid 1 and Grid 3) are chosen for structural analyses. Three types of structural analyses are conducted between the chosen two grids, respectively grid comparison, proportion comparison and ability of self-supporting (two types of interlocking joints are tested). In all the analyses, Grid 1 has a better performance regarding the results of principal stresses and displacements. Next, two more analyses are conducted in Grid 1 for exploration of proportions and ability to self-support. In the end, proportion 5:2 is chosen as the most suitable proportion for Grid 1 to continue as a base for the algorithm design.

In the process of the four-step algorithm design, a 3D Grid System is firstly constructed based on the chosen Gird 1 with proportion 5:2, while the proportion and size of block in this step is defined as user input, meaning that users can choose their preferred block size and proportions themselves. In the second step, a shell structure is obtained by a form-finding process (in this case, using Force Density Method). However, the method for form-finding is another user-define choice that users can choose their preferred form-finding methods as well. The first two steps provide as inputs for the following steps. By obtaining the 3D Gird System and the predefined shell structure, an approximated shell structure can be gained in the third step. Last but not least, a refining process is conducted in the fourth step. The last step is aiming at smoothing the Intrados to provide a comfortable spatial space for people to live in. Therefore, a sub-tessellation process is conducted within the approximated blocks, then by removing the unnecessary sub-blocks within every modular block and smooth the intrados for each block, final voussoir geometry are obtained, with the blocks facing inwards customized and other blocks generic.

In the final results, the designed interlocking mechanisms make it possible for the structures to be easily demounted, while reconfigurability is facilitated by keeping the voussoir geometries, for the most part, generic. The main exceptions are the inward-facing blocks, which are customized in order to better approximate the geometry of the shell shape obtained from the form-finding process. Users applying the methodology can also

determine the levels of sub-tessellation and thus determine the degree of smoothness of intrados (and thus the degree of customization of inner blocks). However, it should be pointed out that these customized blocks are expected to comprise only a small part (24% in this case) of the total structure, meaning that most of the structure can still be reused, but this fraction are to be reduced in the future work.

5.2 Limitations (Future Work)

Additional limitations include: 1) the base mesh for the form-finding process is assumed as a rectangle to ensure the successful selection of inner blocks and outer blocks; 2) the degree of sub-tessellation is identified through an even number to ensure full contact when unnecessary sub-blocks are removed. Further developments are therefore needed to improve and validate the method, and develop a practical tool to be used by designers. 3) Although the current customized blocks occupied a small fragment of all the blocks, this number still needs to be reduced in the future work to increase the efficiency of mass production (in the example case, the total number of blocks are 250, and the customised blocks are 54, takes 21.6% of the total blocks).

Reference

[1] Block, P. (Philippe C. V. (2005). *Equilibrium systems: Studies in masonry structure* [Thesis, Massachusetts Institute of Technology]. https://dspace.mit.edu/handle/1721.1/32096

[2] *Shell structure | Definition, Properties, Examples, & Facts | Britannica*. (n.d.). Retrieved January 9, 2022, from https://www.britannica.com/technology/shell-structure-building-construction

[3] Mazin Elbashkatib. (09:07:55 UTC). *The shell structure system*. https://www.slideshare.net/MazinElsayedAElbashk/the-shell-structure-system

[4] Como, M. (2016). *Statics of Historic Masonry Constructions* (Vol. 5). https://doi.org/10.1007/978-3-319-24569-0

[5] Block Research Group. (n.d.). Retrieved December 20, 2021, from https://block.arch.ethz.ch/brg/publications/392

[6] Assembly-aware design of masonry shell structures: A computational approach | Gene Ting-Chun Kao. (n.d.). Retrieved January 10, 2022, from https://www.geneatcg.com/assembly-aware-design-of-masonry-shell-structures-a-computational-approach/

[7] Deuss Mario, Panozzo Daniele, Whiting Emily, Liu Yang, Block Philippe, Sorkine-Hornung Olga, & Pauly Mark. (2014). Assembling self-supporting structures. *ACM Transactions on Graphics (TOG)*. https://doi.org/10.1145/2661229.2661266

[8] Liew, A., Stürz, Y. R., Guillaume, S., Van Mele, T., Smith, R. S., & Block, P. (2018). Active control of a rod-net formwork system prototype. *Automation in Construction*, *96*, 128–140. https://doi.org/10.1016/j.autcon.2018.09.002

[9] Veenendaal, D., & Block, P. (2014). Design process for prototype concrete shells using a hybrid cable-net and fabric formwork. *Engineering Structures*, *75*, 39–50. https://doi.org/10.1016/j.engstruct.2014.05.036

[10] Rippmann, M., T., V., Popescu, M., Augustynowicz, E., Echenagucia, T., Calvo Barentin, C., Frick, U., & P., B. (2016). *The Armadillo Vault: Computational design and digital fabrication of a freeform stone shell*. https://doi.org/10.3218/3778-4_23

[11] *The-Colosseum.net: Figures*. (n.d.). Retrieved 24 June 2022, from https://www.the-colosseum.net/architecture/numeri_en.htm

[12] Iannuzzo, A., Dell'Endice, A., Avelino, R. M., Kao, G. T. C., Mele, T. V., & Block, P. (n.d.). COMPAS MASONRY: A COMPUTATIONAL FRAMEWORK FOR PRACTICAL ASSESSMENT OF UNREINFORCED MASONRY STRUCTURES. 11.

[13] Cowan, H. J. (1977). A history of masonry and concrete domes in building construction. *Building and Environment*, *12*(1), 1–24. https://doi.org/10.1016/0360-1323(77)90002-6

[14] Como, M. (2016). *Statics of Historic Masonry Constructions* (Vol. 5). https://doi.org/10.1007/978-3-319-24569-0

[15] Snijders, M. (n.d.). *Corbeled dome design using HCEB*. 43.

[16] Fraddosio, A., Lepore, N., & Piccioni, M. D. (2019). Further refinement of the Corbelling Theory for the equilibrium analysis of corbelled domes. *Curved and Layered Structures*, *6*(1), 30–40. https://doi.org/10.1515/cls-2019-0003

[17] Estrin, Y., Dyskin, A., & Pasternak, E. (2011). Topological Interlocking as a Material Design Concept. *Materials Science and Engineering: C*, *31*, 1189–1194. https://doi.org/10.1016/j.msec.2010.11.011

[18] Sheth, U., & Fida, A. (n.d.). *FUNICULAR STRUCTURES USING TOPOLOGICAL ASSEMBLIES*. 10.

[19] Tessmann, O. (2012). Topological interlocking assemblies. *Digital Physicality: Proceedings of ECAADe 2012*, 211–219.

[20] Belov-Kanel, A., Dyskin, A., Estrin, Y., Pasternak, E., & Ivanov-Pogodaev, I. (2009). Interlocking of Convex Polyhedra: Towards a Geometric Theory of Fragmented Solids. *Moscow Mathematical Journal*, *10*. https://doi.org/10.17323/1609-4514-2010-10-2-337-342

[21] Shell Structures for Architecture: Form Finding and Optimization. (n.d.). Routledge & CRC Press. Retrieved 19 January 2022, from https://www.routledge.com/Shell-Structures-for-Architecture-Form-Finding-and-Optimization/Adriaenssens-Block-Veenendaal-Williams/p/book/9780415840606

[22] Veenendaal, D., & Block, P. (2012). An overview and comparison of structural form finding methods for general networks. *International Journal of Solids and Structures*, *49*(26), 3741–3753. https://doi.org/10.1016/j.ijsolstr.2012.08.008

[23] Gueto, M. C. (n.d.). *THE FORCE DENSITY METHOD FOR FCRM FINDING AND COMPUTATION OF GENERAL NETWORKS*. Retrieved January 19, 2022, from https://www.academia.edu/9600283/THE_FORCE_DENSITY_METHOD_FOR_FCRM_FINDING_AND_COMPUTATION_OF_GENERAL_NETWORKS

[24] Rippmann, M. (2016). *Funicular Shell Design: Geometric approaches to form finding and fabrication of discrete funicular structures*. https://doi.org/10.3929/ethz-a-010656780

[25] Block, P. (Philippe C. V. (2009). *Thrust Network Analysis: Exploring three-dimensional equilibrium* [Thesis, Massachusetts Institute of Technology]. https://dspace.mit.edu/handle/1721.1/49539

[26] C. A. Felippa, "Introduction to Finite Element Methods," p. 791.

[27] J. V. Lemos, 'Discrete Element Modeling of Masonry Structures', International Journal of Architectural Heritage, vol. 1, no. 2, pp. 190–213, May 2007, doi: 10.1080/15583050601176868.

[28] Sarhosis, V., Lemos, J. V., & Bagi, K. (2019). *Chapter 13—Discrete element modeling* (B. Ghiassi & G. Milani, Eds.; pp. 469–501). Elsevier. https://www.sciencedirect.com/book/9780081024393/numerical-modeling-of-masonry-and-historical-structures

[29] Vasilis, S., Katalin, B., V, L., José, & Gabriele, M. (2016). *Computational Modeling of Masonry Structures Using the Discrete Element Method*. IGI Global.

[30] Architectural Heritage, vol. 1, no. 2, pp. 190–213, May 2007, doi: 10.1080/15583050601176868.Smith, Michael. / ABAQUS/Standard User's Manual, Version 6.9. Providence, RI: Dassault Systèmes Simulia Corp, 2009.

[31] Itasca Consulting Group, Inc. (2020) 3DEC — Three-Dimensional Distinct Element Code, Ver. 7.0. Minneapolis: Itasca.

[32] https://www.phd.eng.br/wp-content/uploads/2015/12/en.1991.1.4.2005.pdf

[33] Afif, N., Zagi, N. Z., Hariyadi, A., & Cinderakasih, A. P. (2022). A Modular Interlocking Element for Material-Efficient Stereotomy Construction. Nexus Network Journal, 24(1), 103–145. https://doi.org/10.1007/s00004-021-00577-6

[34] S. Bitting, S. Azadi, and P. Nourian, Reconfigurable Domes: Computational design of dry-fit blocks for modular vaulting. 2021. doi: 10.52842/conf.ecaade.2021.1.263.

[35] K. Daw, S. Azadi, P. Nourian, and H. Hoogenboom, *Earthy Honeycombs: Construction Design of Adobe Shell Structures by Topological Polyhedralization*. 2019. doi: 10.13140/RG.2.2.19015.75684.

[36] P. Nourian, S. Azadi, H. Hoogenboom, and S. Sariyildiz, *EARTHY: Computational Generative* Design for Earth and Masonry Architecture. 2020. doi: 10.13140/RG.2.2.28390.65607.

[37] H.-J. Schek, “The force density method for form finding and computation of general networks,” *Computer Methods in Applied Mechanics and Engineering*, vol. 3, no. 1, pp. 115–134, Jan. 1974, doi: 10.1016/0045-7825(74)90045-0.

[38] P. Nourian, R. Goncalves, S. Zlatanova, K. Ohori, and A.-V. Vo, ‘Voxelization Algorithms for Geospatial Applications’, *MethodsX*, vol. 3, Jan. 2016, doi: 10.1016/j.mex.2016.01.001.

Appendix 1: Input Data for Structural Analyses

1. Material Property

	Material Property		
	Density [kg/m^3]	1800	
	Young's Modulus [kN/m^2]	1.55E+07	
	Passion Ratio [-]	0.2	
Drucker Prager	Angle of Friction [°]	31.79	
	Flowstress Ratio [-]	0.8	
	Dilation Angle [°]	2.85	
	Drucker Prager Hardening		
	Hardening behavior type	Compression	
	Data	Yield Stress [MPa]	Abs Plastic Strain [-]
		7.26	0
		7.03	0.00045
		6.58	0.0029
		5.9	0.0044
		4.83	0.005
Contact Property	Tangential Behavior	Friction formulation	Penalty
		Friction Directionality	Isotropic
		Friction Coefficient	Slip Rate / Contact Pressure [kPa]
		1.504	0 / 0.5
		0.91	0 / 1
		6116	0 / 2
	Normal Behavior	Pressure Overclosure	"Hard" Contact
	Abdulla, K. F., Cunningham, L. S., & Gillie, M. (2017). Simulating masonry wall behaviour using a simplified micro-model approach. Engineering Structures, 151, 349–365. https://doi.org/10.1016/j.engstruct.2017.08.021		

2. Basic Units in Abaqus and 3DEC

1) Units in Abaqus

Table 1. Consistent units.

Quantity	SI	SI (mm)	US Unit (ft)	US Unit (inch)
Length	m	mm	ft	in
Force	N	N	lbf	lbf
Mass	kg	tonne (10^3 kg)	slug	lbf s^2/in
Time	s	s	s	s
Stress	Pa (N/m^2)	MPa (N/mm^2)	lbf/ft^2	psi (lbf/in^2)
Energy	J	mJ (10^{-3} J)	ft lbf	in lbf
Density	kg/m^3	tonne/mm^3	slug/ft^3	lbf s^2/in^4

2) Units in 3DEC

Property	SI				Imperial	
Length	m	m	m	cm	ft	in
Density	kg/m^3	10^3 kg/m^3	10^6 kg/m^3	10^6 g/cm^3	slugs/ft^3	snails/in^3
Force	N	kN	MN	Mdynes	lb$_f$	lb$_f$
Stress	Pa	kPa	MPa	bar	lb$_f$/ft^2	psi
Gravity	m/sec^2	m/sec^2	m/sec^2	cm/sec^2	ft/sec^2	in/sec^2
Ball stiffness	N/m	kN/m	MN/m	Mdynes/cm	lb$_f$/ft	lb$_f$/in

Appendix 2: Applied Load Cases for Structural Analyses

1. Applied Load Cases

	Block & applied loads				
	Voxel unit dimension HeightxWeightxLength [mxmxm]	0.45x0.45x0.45			
	Number of voxels for single brick [-]	8			
	Self-weight per brick [kg]	m=ρV	7.3728		
Vertical	Snow load [kN/m²]	Characteristic value of snow load (s_k)	0.45 EN 1991-1-3 (zone A1)	0.675KN/m²	5.175KN/m²
		safety factor	1.5		
	Live load [kN/m²]		3 kN/m²	4.5KN/m²	
		safety factor	1.5		
Horizontal	Wind load [kN/m²]	Basic velocity pressure (q_b)	0.35 EN 1991-1-4 (zone 2)		0.525kN/m²
		safety factor	1.5		

2. Applied Load Cases in 3DEC and Abaqus respectively

	3DEC	Abaqus
5.175KN/m²	5175N/m²	G1:5.175*0.45*0.45*1000 =1048N G2: 1048/2=524N
0.525kN/m²	525N/m²	0.000525kN/mm²