Approximating & Optimzing Designs with Modular Structures

John Rao
4.451 Computational Structural Design and Optimization Final Report
Professor Caitlin Mueller
January 21, 2021

ABSTRACT

This paper describes an ongoing research on generating and evaluating structure designs constructed as aggregations of modular parts, with focus on using recycled shipping containers as the basic units. A new workflow is proposed to allow designers to input more creative design preferences and select from a catalog of potential structures for further studies. The computational pipeline introduced in this paper is developed from existing discrete assembly design, structural analysis, and optimization tools in Rhino Grasshopper environment. This current implementation takes a simple geometry as user input then parameterizes the input to generate different design choices of shipping container assemblies that approximates the input geometry and their respective structural performance. The eventual goal of this project is to develop a tool that helps architects design and evaluate, quantitatively and qualitatively, complex modular structures following specific design intentions. This paper discusses the current implementation of such tool in Rhino Grasshopper through a case study, presents the experimental results, and introduces potential next steps for this project.

1. Introduction

Using modular units as a building system has become increasingly popular and has been proven to be efficient. As seen today, modular constructions are often optimized to most efficiently use the existing structures of the modules, such as the structural efficiency of the system or the occupiable volume the system yields thus resulting in rectangular buildings of boxes that are repetitively stacked on the four corners. It can be challenging to understand the structural performance of modular systems once the design of such system becomes more geometrically complex. This seems to be a major factor that restricts the creativity of designers. When designing with modular units, designers tend to stick to forms that are intuitively easy to evaluate and to be constructed. Without a thorough understanding of the structural behaviors of modular systems, it is difficult for architects to balance the quantitative goals and qualitative goals of a design. Although many structural analysis software allow designers to access the quantitative performance of a particular design, there is no existing tool that is specifically geared towards designing with modular units. Since there are a lot variables, like the rotation or the offset distance of certain modules, can have unexpectedly large impact on the structural efficiency of a modular system, it would be impractical to manually model and test all design alternatives of an early concept.

If the assembly logics and structural feedbacks can be generated automatically, there is the potential to generate a greater variety of possible designs following a specific concept, allowing designers to spend more time on iterating through different concepts. The conceptual framework of this project is illustrated in Figure 1. The goal for this project is to allow designers to input their early concepts regarding the overall form of the building and then to access 3D massing models and the structural performances of different modular buildings following that input concept.

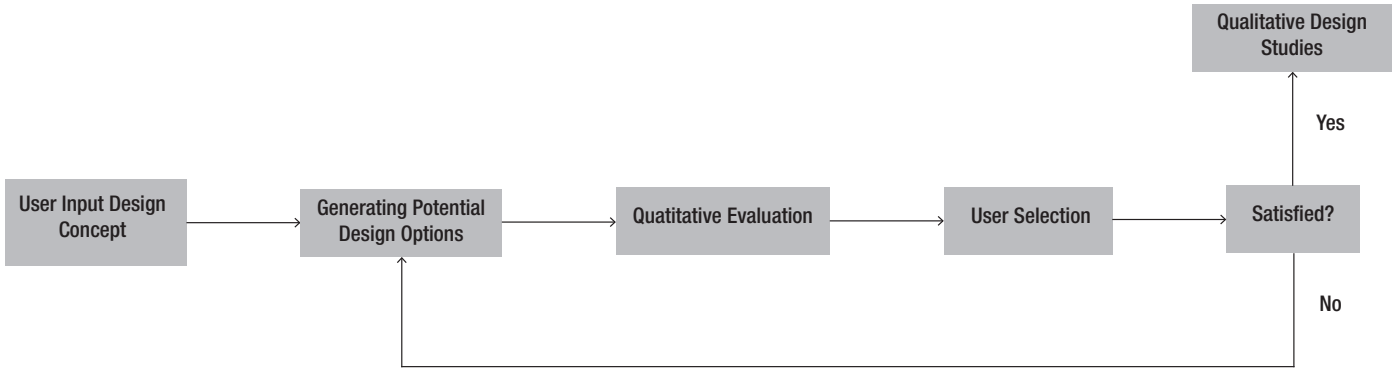


Figure 1. Conceptual Framework

Following the conceptual framework outlined above, the current approach to this problem is to implement a computational pipeline in Rhino Grasshopper environment. The computational pipeline can be broken down into seven steps, as shown in Figure 2. The first step is to define the assembly parts (or modules) to be used to generate design options. The script then creates design constraints and variables based on the user input and common construction logics. The third step is to generate different assemblies using the modules given in the first step. Different quantitative analysis of the generated assemblies can then be calculated. For this project specifically, structural analysis is performed on all these different assemblies and the total strain energy of each assembly is used as a measurement of structural efficiency. The next step is to define the objective functions that designers would use to evaluate the performance of each option. An objective function can be defined using only the strain energy of each assembly, but designers can also include factors like the number of modules used or the angle of rotation allowed. In this step, the designers have the opportunity to introduce their subjective opinions regarding which aspects of the design they would want to optimize. For instance, if designers choose to include rotation angle and strain energy in the objective function, the best performing designs according to this function would be the assemblies with the most (or least) rotations and the lowest strain energy. Once an objective function is set, the script would then adjust the design parameters to produce a specified number of designs and their respective performance score. The current implementation of this project uses a genetic algorithm solver and a sampling tool to capture both the most quantitatively optimal design and a greater variety of other options. The final part of the pipeline visualizes the different options along with their variable values so that designers would be able to compare different designs in parallel and have a better understanding of the relationship between the geometric form and the performance of a modular structural system, and again, this would allow designers to explore and test different ideas almost instantly.

This project aims to demonstrate the potential of such a computational tool and lay the foundation for further research in developing a more robust, user-friendly, and interactive optimization tool that all designers can take advantage of while designing for modular constructions. Figure 3 shows a set of designs generated by sampling 30 points in the design space using this current implementation of the pipeline. Section 4 of this paper will further discuss the variables and user input that produced these results. These results show that many designs that have relatively similar quantitative performance can be significantly different in terms of their form and other qualitative values.

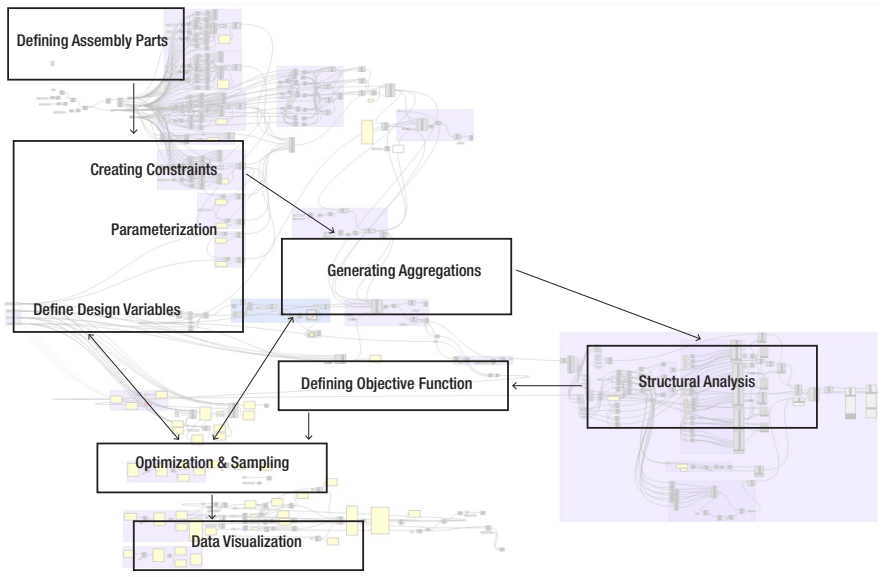


Figure 2. Implementation Framework

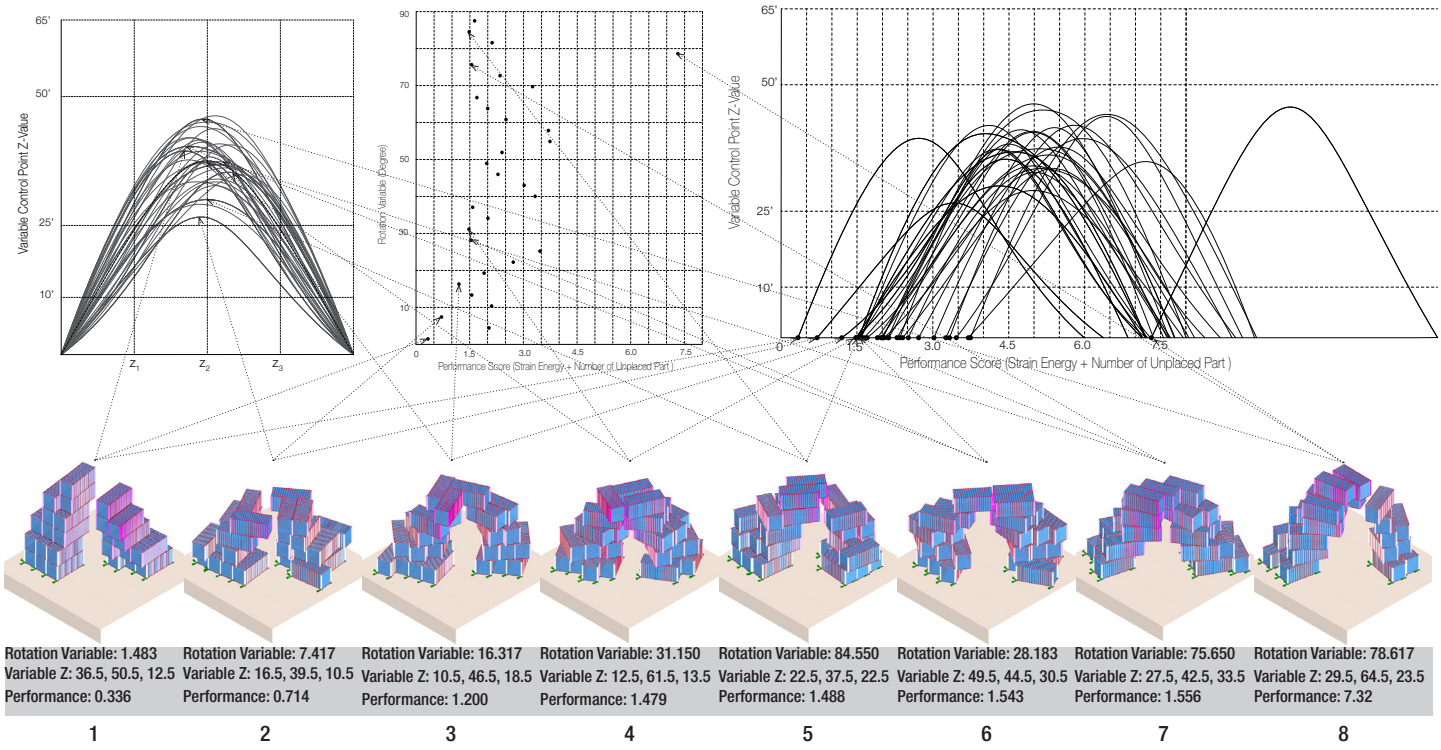


Figure 3. Experiement results highlighting 8 designs of 30 sampled using input and design variables described in Section 4.

It is also noteworthy that some designs that seems structurally inefficient actually perform better structurally that others that might seem to be a higher performing structural design. For example, the 3rd design, which has more cantilevering modules than the 4th design, has a 18% higher performance score. Some of these preliminary results also show that symmetry might not always have a large impact on the total performance. This demonstrates that being able to get instant performance feedbacks would open more possibilities for design.

2. Literature Review

Currently, there is a wide range of research focus on modular constructions. Some focus on the potential and market demand of modular construction in hotel and residential applications [5, 6]. Some present recent development in architectural computing for designing and manufacturing modular assemblies [2, 8, 9, 10, 11]. Some present more detailed structural elements and behavior modular units in different loading conditions [1, 5, 12]. Despite modular construction's growing popularity, very few research studies focus on methods of balancing the qualitative design needs and quantitative performance of modular structures.

In reference[8, 9, 10], Andrea Rossi and Oliver Tessmann primarily discuss their research in bridging the gap between design and robotic fabrication using discrete element assembly. The research presented in reference[8] is similar to what this project is attempting to achieve in some ways. Their proposed method takes a refined design geometry and custom load conditions as inputs which are then converted to a material density field by a topology optimization algorithm. In their case, the material density field is used as the aggregation driver to modularize the input geometry. Even though it can optimally translate a specific design to robotically manufacturable assembly of discrete elements, it might not be the most efficient tool for generating diverse design options for users to explore at early design stage. Additionally, the results from Rossi and Tessmann's research have only demonstrated its performance with smaller building block modules but not with larger and inhabitable modules like shipping containers which would certainly further increase the computational intensity and might present other complications.

Reference[10] focuses more specifically on the different methods of generating architectural objects as aggregations of modular parts by combining discrete element design with voxel-based design. These methods are proven to be able to generate aggregations that respond to different performance-based requirements [9]. The

current implementation of the project uses the software tool developed as a part of Rossi and Tessmann's research for aggregating modules.

An article published in *Journal of Civil Engineering and Management* [1] discusses on the use of shipping containers as structure modules and performs finite elements analysis on 40' and 20' High Cube containers. Luis Bernardo also presents a case study design of a single-family house using 8 shipping containers. The structural elements detailed in this article will be used for the modules in this project. Beyond this project, the calculations presented in this paper could be used as reference for further research focusing on more detailed structural representation (Section 3.2 describes the structural model used in this project) of shipping containers. There is also the potential to introduce other types of modules, as described in [5]. Andrew William Lacey's paper also discusses the structural responses of different modular systems to natural hazards like wind and earthquake, which could also be incorporated into the performance measurements of this project in the future.

3. Methodology

3.1 Aggregation Generation

Wasp, a Grasshopper plug-in for discrete modeling, is the primary tool used to generate aggregations of the modules in the current implementation of this project. Wasp is able to produce designs that are discrete in their nature and generate assembly logic directly from the geometry of the parts to be assembled without relying on any pre-determined grid. [10] With Wasp, users are provided with the opportunity to control aggregation procedures by specifying discrete connections, assembly rules, local and global constraints, as well as drivers for aggregation. More research on the development and algorithms of Wasp is discussed in Andrea Rossi's papers [8, 9, 10].

3.1 (a) Aggregation Methods

The current version of Wasp has 2 built-in aggregation driver components, stochastic aggregation and field-driven aggregation. These two drivers follow different aggregation procedures for generating assemblies of given parts. This part of the paper will briefly discuss the advantages and disadvantages of each aggregation procedures as well as three different approaches of using these aggregation drivers that were experimented in this project.

Stochastic aggregation, as the name indicates, aggregates given geometry parts in a stochastic process during which parts and aggregation rules are selected randomly at every step. Stochastic aggregation produces a larger variety of assemblies because there are fewer constraints. Since parts are placed at random locations using a random rule from given rule sets, stochastic aggregation is a very efficient algorithm and it can generate different assemblies with a very large number of parts much faster than field-driven aggregation. At first, these two main advantages of stochastic aggregation seem to be beneficial for the purpose of this project because, once again, the goal is to quickly produce a large variety of options for designers. However, the fact that stochastic aggregation employs an entirely random procedure without giving users access to the seed number makes it nearly impossible to replicate a specific result and to study the set of rules that led to that result. It is also not possible to generate assemblies that follow specific design intent with stochastic aggregation without creating additional constraints to direct the growth of the aggregation.

Field-driven aggregation is a voxel-based volumetric modelling method designed for discrete assemblies. It takes a scalar field as an input which then drives the growth of the aggregation. At each step, a part is placed at the voxel with the highest value in the scalar field that is reachable according to the given assembly rules. In comparison to stochastic aggregation, field-driven aggregation procedure is much more constrained and thus often only generates one solution for one specific input scalar field. This limits the variety of options produced but does allow users to replicate designs and parameterize the scalar field to have more control over the generation process. In Grasshopper environment, scalar fields can be generated using a volume, a surface, a curve, or a combination of the three. As discussed below, field-driven aggregation is proved to produce assemblies that better approximate the design input.

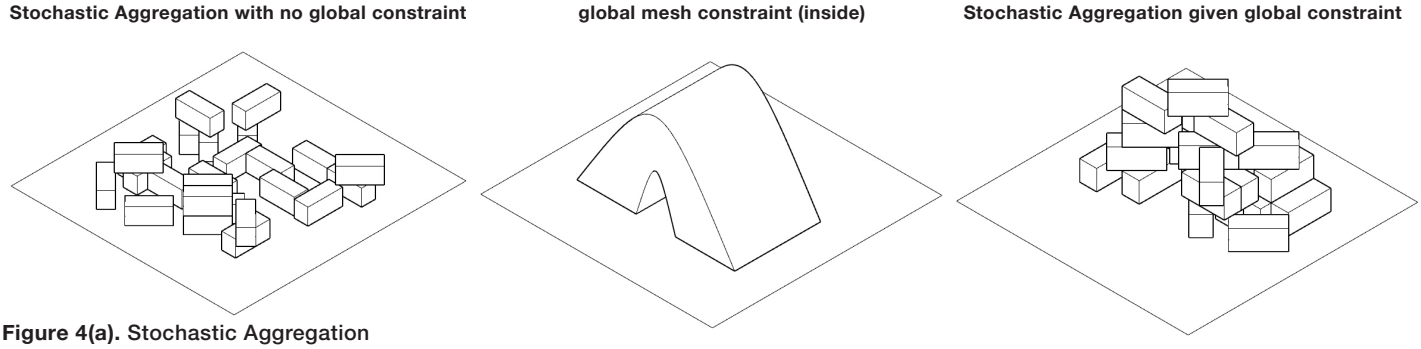


Figure 4(a). Stochastic Aggregation

The three approaches that were tested while creating the current implementation of this project are illustrated in Figure 4. The first approach tested in the implementation is using stochastic aggregation with global constraints. As shown in Figure 4(a), global constraints can be set to force the growth of aggregation to stay inside or outside specified volumes, or even between two volumes. Even though stochastic aggregation itself has a very low computational cost but checking whether global constraints are satisfied at each step increases the time complexity of the algorithm exponentially. Furthermore, controlling stochastic aggregation with global constraints significantly reduces the number of solutions found, and sometimes does not guarantee convergence.

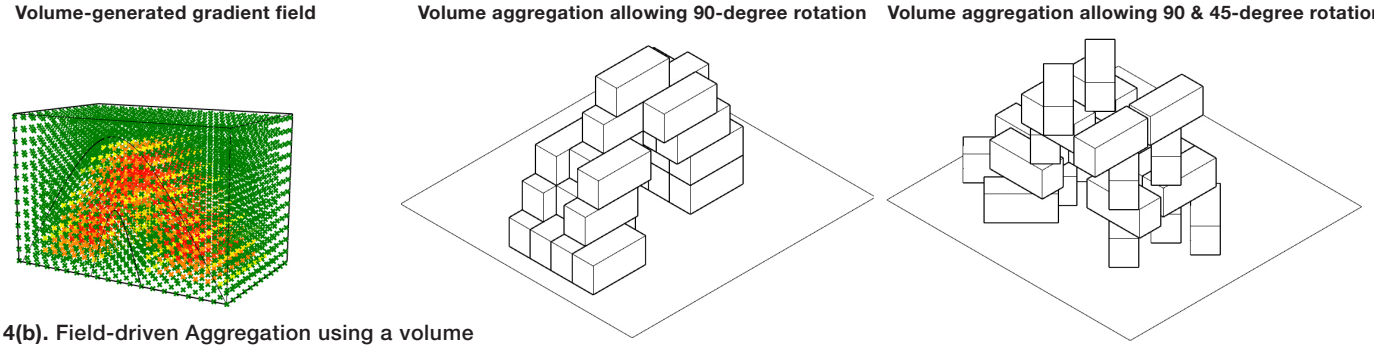


Figure 4(b). Field-driven Aggregation using a volume

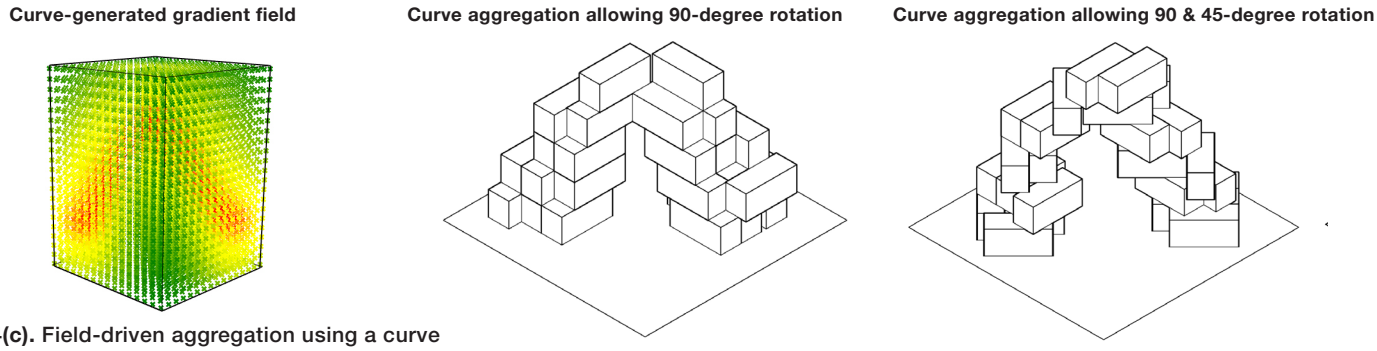


Figure 4(c). Field-driven aggregation using a curve

The second and third approach, shown in Figure 4(b) and Figure 4(c), both use field-driven aggregation but the gradient field inputs are created with a volume and a curve respectively. The color of point cloud in the diagrams denote the value associated with that particular voxel. In a way, the scalar value determines the probability of a part being placed at a specific voxel. The green points represent voxels with probability of 0 while the red points represent voxels with probability of 1. The yellow and orange gradient represent voxels with probability values between 0 and 1. When generating a gradient field with a volume, all voxels outside of the given volume are assigned a value of 0, strictly restrains that growth of the aggregation to inside of the volume. On the other hand, this curve-generated gradient field assigns scalar values according to the distance between the voxel and the given curve, thus creating a smoother gradient that is generous enough to allow the aggregation to grow around it. After testing a few different gradient fields, it was clear that curve-generated gradient field generally allowed better approximation of the given geometry because of the scale of the container module and the relatively small number of points used in this project. Using a curve-generated gradient field also allows faster computing and easier parametrization of the input. Therefore, the current implementation uses this approach as the chosen method of driving aggregation, which means that users would use curves as design intent inputs.

3.1 (b) Other Constraints

The first step in generating aggregations of modules with Wasp is to define the parts to be aggregated. This process involves designating the geometry of the part, local constraints, and the points on the geometry face to be the connected to other parts. For the purpose of this project, the part geometry is defined as a rectangular box with width of 8 feet, height of 9 feet and length of 20 feet, representing shipping containers that are known as 20'HC [1] on the market. The connections and local support conditions would be used as the main constraints for the aggregation procedures in this project.

The set of connections given in the part definition determines where and how a part can be connected to a previous part in the assembly process. Since this project is geared specifically towards designing architecture, it is crucial to define connections that would restrain the aggregated modules to always form continuous volumes and create inhabitable spaces. Multiple connection points are needed on some faces to allow the containers to offset and cantilever. Different sets of connection points were tested in a series of mini-optimization problems. Between 2 and 10 connection points were assigned to the long faces of the container part, and an optimization algorithm checked if the given set of connections allowed Wasp to place a specified number of parts while following an input curve. The results showed that having too many connections actually allowed the aggregation growth to be too constrained to the input curve thus unable to always place the desired number of modules. The set of connections shown in Figure 5 always placed desired number of parts while producing some variety of designs. The chosen set include 1 point on each ends of the container, 3 points on each top and bottom faces, and 3 points on each of the long side faces.

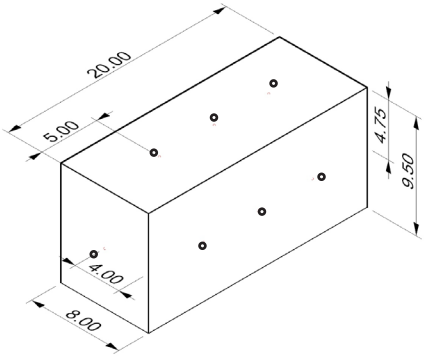


Figure 5. Conection Constraints

In order to avoid generating geometries that cannot be analyzed structurally, it is also important to specify support conditions as local constraints. In Wasp, the support conditions are defined by a set of lines, extending from each part, that must by intersected by another part. A part would only be placed if one of the support conditions were satisfied. 2 Sets of support conditions that were tested are illustrated in Figure 6. Once again, having too many support conditions limits the variety of design options, so the set of support conditions in Figure 6(a) was chosen for this implementation.

Additional constraints are embedded in a set of written assembly rules that dictates which connections are allowed to connect. Since this project is aimed to have assemblies computed by Wasp to mimic real-world constructions, it is optimal to have the aggregation grow from the bottom up. Therefore, the assembly rules are written so that the top of the modules would only connect to the bottom connections of other modules.

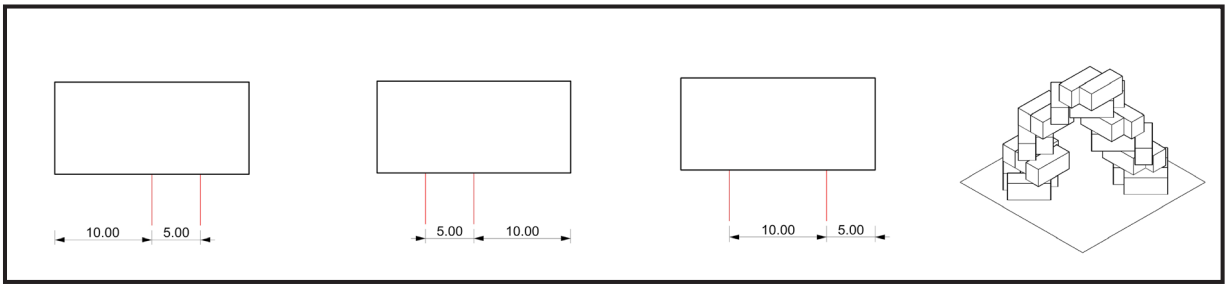


Figure 6(a). Suppport constraint used in this implementation

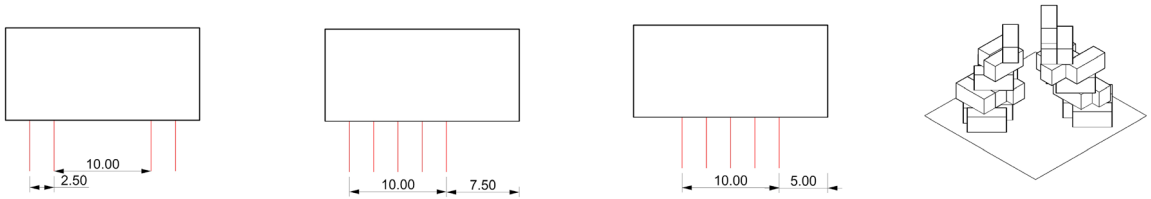


Figure 6(b). Other support constraint tested with an ached profile curve shown in Fig. 10

3.2 Structural Analysis

The structural analysis part of this project is implemented using Karamba 3D in Grasshopper. Once an assembly is generated by Wasp, Karamba then analyzes the assembly as an system of steel frames. For the purpose of this project, each container is abstracted to a steel frame shown in Figure 7. Each ground level container has 4 fixed supports assigned to its bottom corners. This representation is based on the shipping container structures drawn in reference [1]. The top and bottom face structures of a container are abstracted to steel beams with the cross-section illustrated in Figure 8(a). Figure 8(b) illustrates the cross-section of the vertical members that represent the side faces of a container, and the vertical members at the four corners have the cross-section shown in Figure 8 (c).

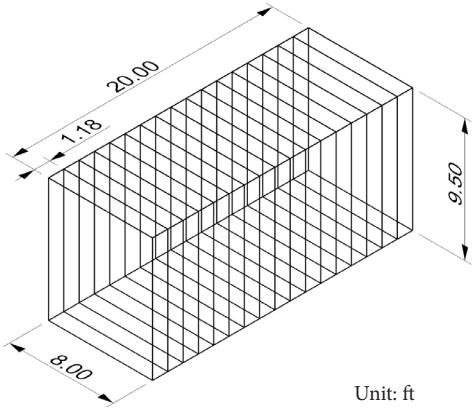


Figure 7. Steel frame abstraction of 20' HC shipping container

In this current implementation, self-weight of 2,800 kilograms is split between 36 point-loads applied to the long horizontal members at the top of the container. However, there is the opportunity for users to adjust the loading conditions to test how different design options would behave in a variety of scenarios. The next section of this paper will discuss other parts of the pipeline that are also specific to user input, such as design variables and objective functions. The Karamba model shown in Figure 9 illustrates the structural behavior of stacking three steel frame systems.

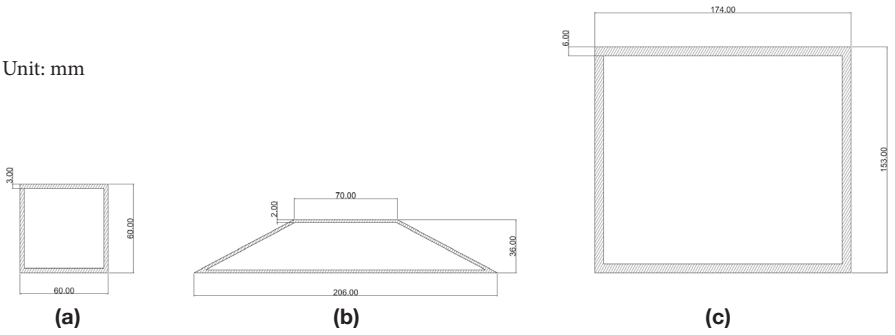


Figure 8. Beam cross-sections

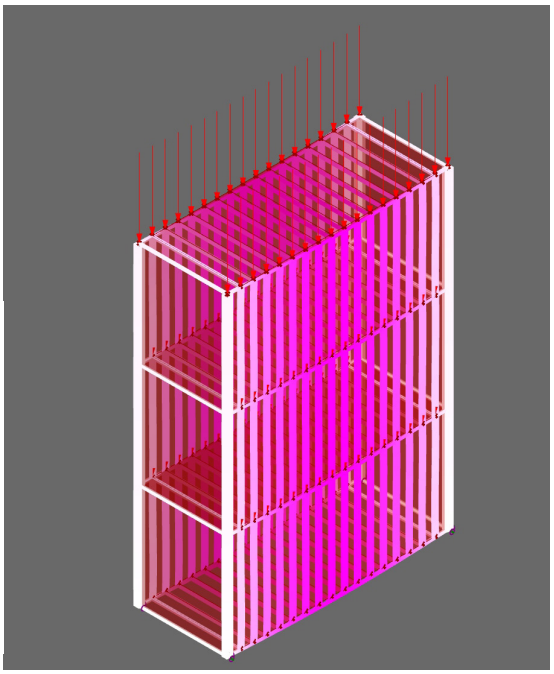


Figure 9. Karamba model of stacking 3 modules

4. Case Study

To demonstrate the capabilities of this tool and how it can be incorporated in the design process, this section of the paper will discuss a case study using the input curve shown in Figure 10. Since the premise of the project is to generate a large variety of interesting geometric forms consisted of shipping containers, using this an arched profile curve, which is not commonly seen in modular construction buildings, demonstrates how this project can be used to discover new design opportunities that might be hard evaluate structurally. The number of variables and their domains are specific to this input curve, and users have the option to specify the domains and number of variables that are appropriate for their design intent. Having users define the variables allows designers to have more control over the aggregation generation process and use this tool in a more iterative manner.

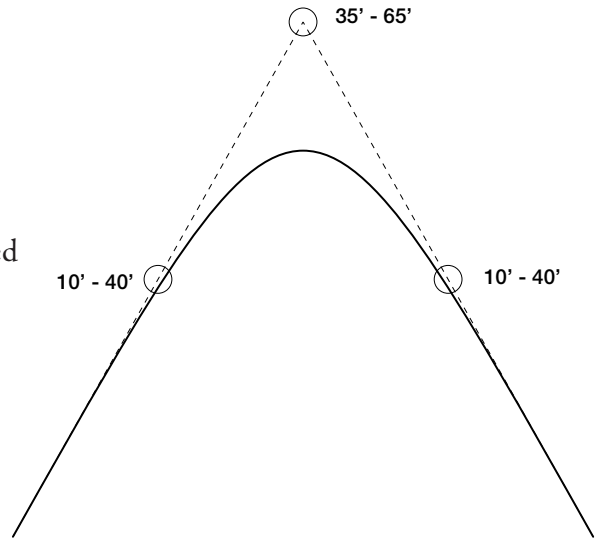
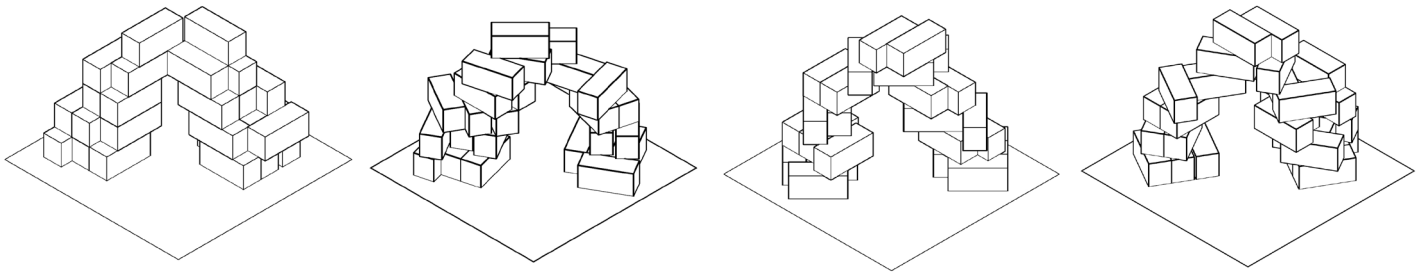


Figure 10. Case study input NURBS curve

4.1 Design Variables

In this case study, four variables are given to the optimization and sampling tools to explore the design space. The first three variables are the Z-values of the three NURBS curve control points illustrated in Figure 10. By parameterizing the input curve, the pipeline is essentially creating a new scalar field to compute a new assembly at each iteration. Based on preliminary testing results, the initial domains of the control points' Z – positions limit the aggregations to between 4 and 6 layers (for faster computation). Increasing the domain would increase the variety of designs while decreasing the domain would thus produce more similar options closer to the initial input curve. The fourth variable is a rotation variable which determines the angle of which the top and bottom connections can rotate to during assembly. Since 90-degree rotations are already enabled in the definition of the parts, the initial range for rotation variable is set to between 0 and 89 degrees in this case. Figure 11 shows how adjusting this variable alone could produce very different design options.



Rotation Variable: 0 degrees Variable Z: 25, 50, 25	Rotation Variable: 15 degrees Variable Z: 25, 50, 25	Rotation Variable: 45 degrees Variable Z: 25, 50, 25	Rotation Variable: 60 degrees Variable Z: 25, 50, 25

Figure 11. Preliminary test results demonstrating rotation variables

4.2 Objective Function

Section 1 of this paper mentions how users could assign the components and their weights in objective functions to meet specific design intent or requirements. Because of the constraints that were defined as a part of the aggregation procedure, certain sampled points in the design space might not place the desired number of parts. In this case study, one of the design requirements is to place as close to 25 containers as possible. Since the total strain energy of the system is used as a measurement of structural performance, the number of modules in the system is also proportional to the structural performance. Therefore, to take into account how many parts were not placed, the objective function in this case study is defined as the sum of the total strain energy and the number of parts that were unplaced. One can observe that the strain energy (in kip*feet) of a 25-containter system has a relatively small value, mostly between 0.28 and 4.50. This means that adding the number of unplaced parts and total strain energy would significantly increase the overall performance score (the best performing design would have the lowest score in this case) thus filtering out design options that do not have the desired number of modules.

Performance Objective Function: Total Strain Energy (kip*ft) + Number of Unplaced Parts

4.3 Results

In order to find the most quantitatively optimal design and establish a point of reference, the first set of experimental results is produced using optimization. Specifically, the script attempts to minimize the overall performance score using evolutionary algorithm solver. Because of the nature of evolutionary algorithm, there is no guaranteed convergence and running the solver for different amount of time can produce very different results. Figure 12 shows the outputs of running the evolutionary algorithm solver for 30 seconds, 5 minutes, 15 minutes, and 30 minutes respectively.

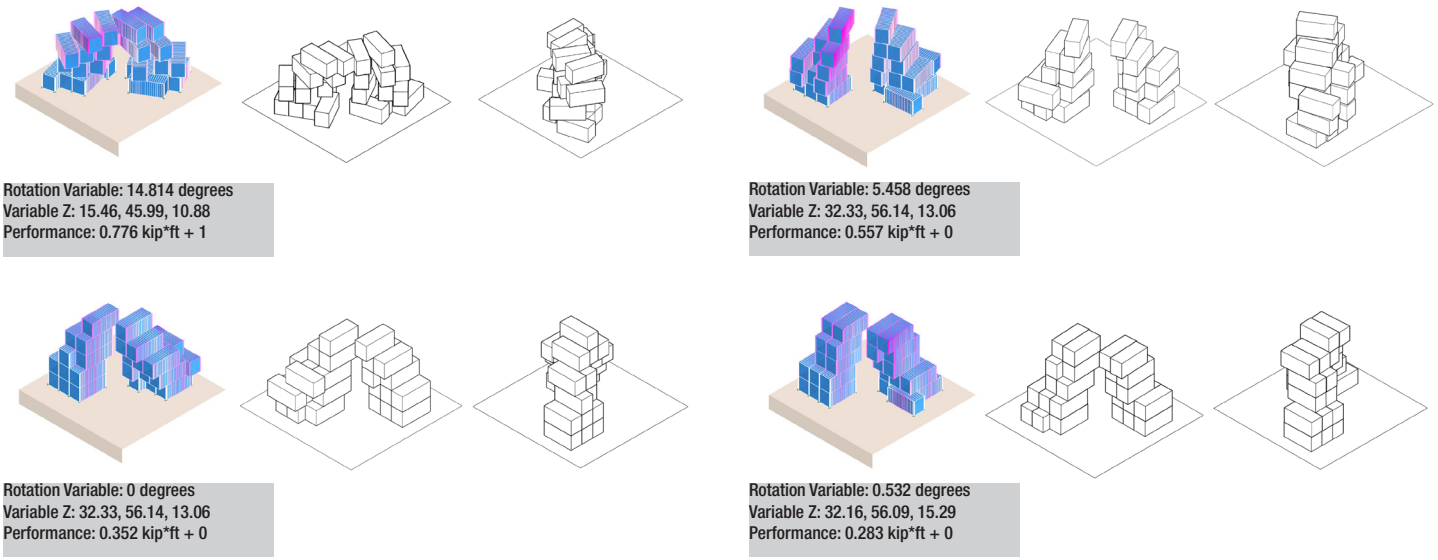


Figure 12. Case study results using optmization

However, using existing optimization tools limits the variety of designs that can be generated and does not allow user selection in the process. This current implementation also uses a sampling tool to explore the design space more comprehensively. Figure 3 and Figure 13 highlight 8 of 30 designs sampled using Latin Hypercube. Both sampled and optimization results show that the best performing structures have rotation variable with a value close to 0, which is not surprising. This proves that the conventional way of stacking modules repetitively is indeed an efficient modular construction method, but the resulting designs might not be as interesting. The rotation variable vs. performance plot also shows that rotation variable is not necessarily the dominating factor. Many options with relatively high rotation variable also have performance score within 10% of that of other better performing structures. At this stage, after users have a better understanding of what designs can be generated from the input curve and how they perform relative to their design variables, designers can manipulate the domain of the design variables to use this tool more iteratively, as described in Section 4.1.

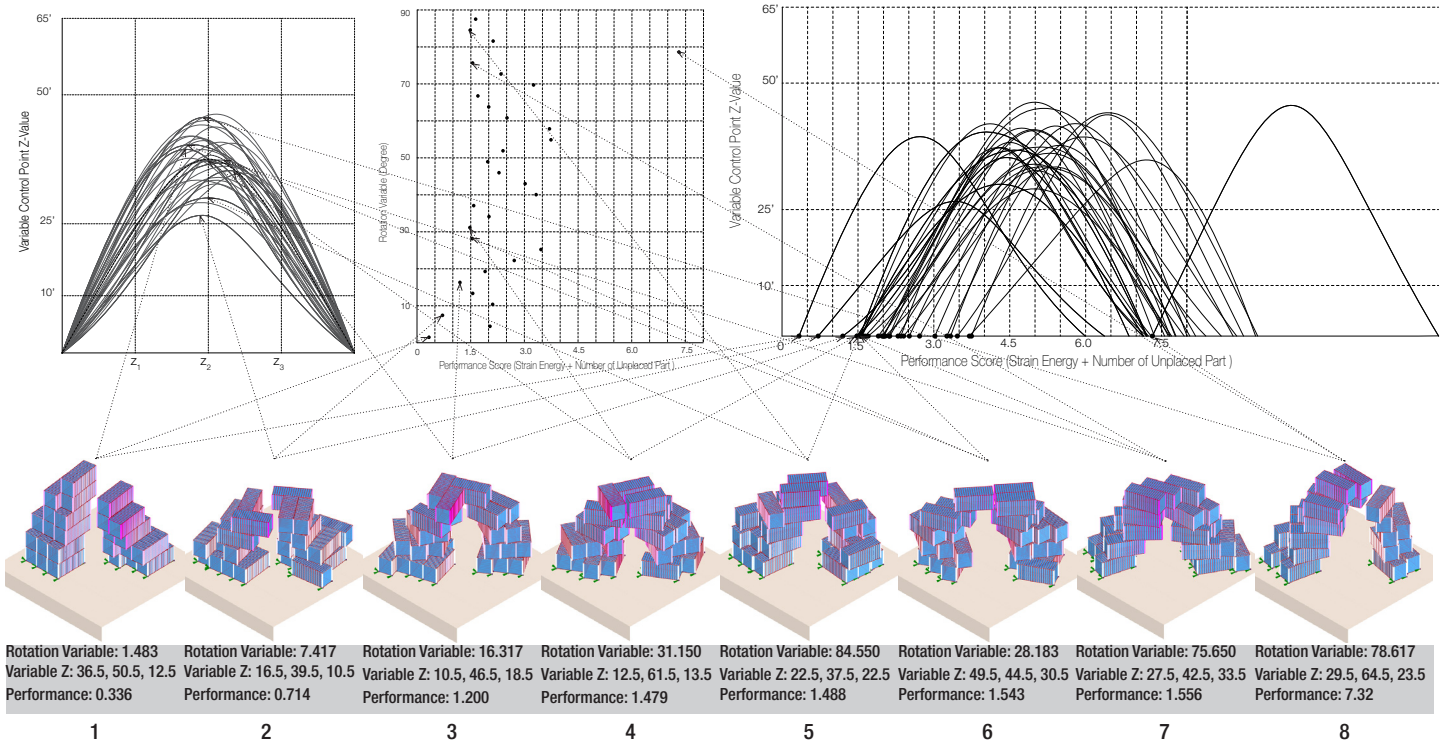


Figure 3. (duplicated) Case study results highlighting 7 quatitatively best performing options and 1 extraneous option of 30 samples

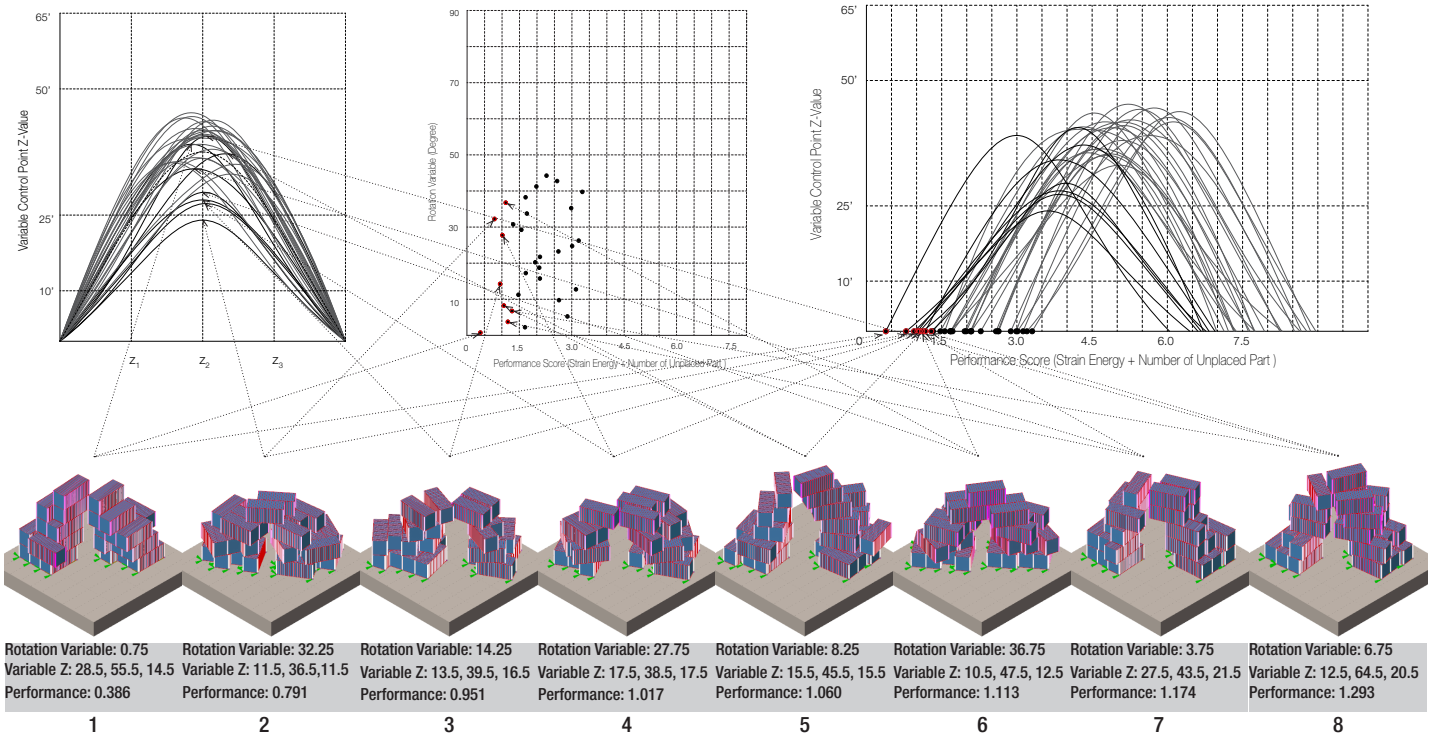


Figure 13. Case study results highlighting 8 quatitatively best performing options of 30 samples

In this case study, the 2nd, 4th and 6th option in Figure 13 are chosen for further exploration because of aesthetics and other qualitative values. To generate more options that are similar the selected designs, the next step is to sample a smaller design space with reduced variable domains. For this specific example, the domain of the rotation variable in reduced to between 25 and 40 degrees. to focus only on designs that rotate similarly as the chosen designs. Figure 14 shows the 8 best performing designs amongst 90 points sampled in this new space. The performance score of all 90 points range between 0.681 and 4.429. 70% of the sampled points scored between 1.000 and 2.250. This set of results seems promising for iterative design process because many of these designs have very similar appearance compared to previous results, which means that users can control the diversity of design options simply by manipulating the domain of the design variables. This process of decreasing the domain of rotation variables can also help designers understand the relationship between the input curve and the overall performance of the structures. In comparison to the previous results, certain common patterns can be better observed between these NURBS curves.

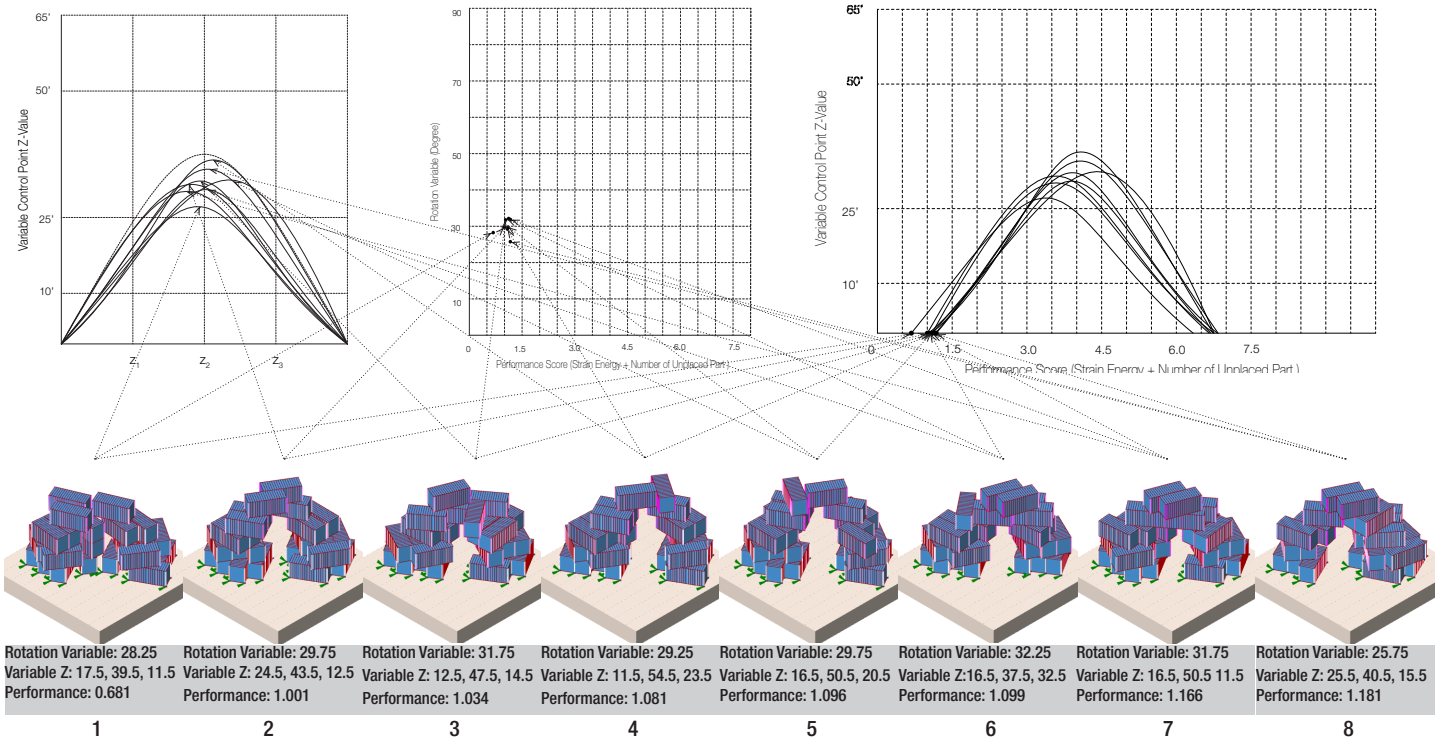


Figure 14. Case study results highlighting 8 quatitatively best performing options of 90 samples

As seen in the results, optimization and sampling method each has its own limitations. Using a sampling tool gives users the option to control the number and diversity of outputs but does not always produce the most quantitatively optimal design. On the other hand, optimization algorithms used in this project does not give users the variety of options desired, which makes it difficult for designers to balance the quantitative and qualitative goals. Section 5 of this paper will discuss how an interactive optimization process can be incorporated into this project in the future to address this problem.

5. Conclusion

5.1 Summary

This paper proposes a new computational tool that aims to help architects explore more geometrically interesting options when designing with modular units. Designers can access a variety of potential designs and their respective structural performance by simply inputting profile curves of early design concepts thus allowing designers to quickly test and iterate through different concepts. Section 3 of the paper describes the three aggregation methods and multiple sets of constraints that were tested in this project. The project currently uses a steel frame abstraction of the 20' HC shipping container for structural analysis and computes the total strain energy as a measurement of structural performance. This current pipeline allows users to define their own objective function to evaluate each option according to their own design priorities. The setup of the design problem is further discussed through a case study in Section 4. Experimental results of the case study using both optimization and sampling are shown. Three different sets of sampling results demonstrate how users can control the diversity of the outputs to balance the quantitative and qualitative performance. Beyond generating modular assemblies automatically, this tool can also help designers gain a better understanding of how certain design changes could affect the structural performance of a modular system.

5.2 Future Work

The current implementation of this tool has some limitations that can be addressed in the future. Currently, the tool works only with curves that can be parametrized with a few variables. To allow more complicated and different types of user input, this tool should be improved to generate gradient-fields as aggregation drivers using a combination of curves, volumes, and surfaces whenever applicable. As discussed in Section 3.2, the module used in this project for structural analysis is a rough approximation of a shipping container. Future development of this tool should attempt to incorporate a more detailed structural model without significantly increasing the computational costs. Furthermore, this tool could potentially extend its capabilities to allow users use custom modules by parametrically adjusting individual members, connection types, and loading conditions of the structural model. The structural performance of a modular system can be quite different depending on the connection types and the loading conditions [5]. Perhaps this would allow designers to consider more site and project specific metrics, such as wind load or seismic hazard.

In future implementations, this tool could also extend its capabilties to optimize for other formulated quantitative goals beyond structural analysis, incorporating other multi-objective optimizations algorithms. As mentioned in Section 4.3, this current version displays results from both optimization and sampling. Neither approach is ideal. Further improvement should also include developing an interactive evolutionary algorithm optimization tool, like StructureFIT [7], that allows users to have more direct control, access more than a single solution, and adjust the diversity of generated designs as desired. Hopefully, future implementations of this tool would enable designers to explore the potential of modular construction without compromising architectural desires and creativity.

References

[1] Bernardo, L. F., Oliveira, L. A., Nepomuceno, M. C., & Andrade, J. M. (2013). Use Of Refurbished Shipping Containers For The Construction Of Housing Buildings: Details For The Structural Project. Journal of Civil Engineering and Management, 19(5), 628-646. doi:10.3846/13923730.2013.795185

[2] Drude, J. P., Rossi, A., & Becker, M. (2019). Project DisCo: Choreographing Discrete Building Blocks in Virtual Reality. Impact: Design With All Senses, 288-299. doi:10.1007/978-3-030-29829-6_23

[3] Ergan, S., Shi, Z., & Yu, X. (2018). Towards quantifying human experience in the built environment: A crowdsourcing based experiment to identify influential architectural design features. Journal of Building Engineering, 20, 51-59. doi:10.1016/j.jobe.2018.07.004

[4] Hyun, H., Kim, H., Lee, H., Park, M., & Lee, J. (2020). Integrated Design Process for Modular Construction Projects to Reduce Rework. Sustainability, 12(2), 530. doi:10.3390/su12020530

[5] Lacey, A. W., Chen, W., Hao, H., & Bi, K. (2018). Structural response of modular buildings – An overview. Journal of Building Engineering, 16, 45-56. doi:10.1016/j.jobe.2017.12.008

[6] Lawson, R. M., Ogden, R. G., & Bergin, R. (2012). Application of Modular Construction in High-Rise Buildings. Journal of Architectural Engineering, 18(2), 148-154. doi:10.1061/(asce)ae.1943-5568.0000057

[7] Mueller, C. T., & Ochsendorf, J. A. (2015). Combining structural performance and designer preferences in evolutionary design space exploration. Automation in Construction, 52, 70-82. doi:10.1016/j.autcon.2015.02.011

[8] Rossi, A., & Tessmann, O. (2017). Aggregated Structures: Approximating Topology Optimized Material Distribution with Discrete Building Blocks. Proceedings of IAASS Annual Symposia, (2017), 1-10.

[9] Rossi, A., & Tessmann, O. (2017). Geometry as Assembly: Integrating design and fabrication with discrete modular units. ECAADe, 2, 201-210.

[10] Rossi, A., & Tessmann, O. (2018). From Voxels to Parts: Hierarchical Discrete Modeling for Design and Assembly. Advances in Intelligent Systems and Computing, 1001-1012. doi:10.1007/978-3-319-95588-9_86

[11] Tessmann, O., & Rossi, A. (2019). Geometry as Interface: Parametric and Combinatorial Topological Interlocking Assemblies. Journal of Applied Mechanics, 86(11). doi:10.1115/1.4044606

[12] Sultana, P., & Youssef, M. A. (2018). Seismic Performance of Modular Steel-Braced Frames Utilizing Superelastic Shape Memory Alloy Bolts in the Vertical Module Connections. Journal of Earthquake Engineering, 24(4), 628-652. doi:10.1080/13632469.2018.1453394