COMPUTER-AIDED DESIGN & APPLICATIONS, 2018
VOL. 15, NO. 5, 653–660
https://doi.org/10.1080/16864360.2018.1441231

The STG-framework: a pattern-based algorithmic framework for developing generative models of parametric architectural design at the conceptual design stage

Chieh-Jen Lin

Tainan University of Technology, Taiwan

ABSTRACT
Although algorithmic modeling tools have become a popular means of generating complex geometric forms, the potential of generative algorithms should not be limited to geometric intentions. However, the need to possess programming and data manipulation skills is often a major obstacle when architects wish to implement algorithms for representing their non-geometric intentions. This paper therefore proposes an algorithmic framework entitled STG*f*, which is based on the "Semantic-Topological-Geometric (STG)" information conversion pattern, and can help architects to convert their abstract design intentions into computational procedures. By providing rewritable sample GhPython scripts and adjustable components' clusters of Grasshopper, the STG*f* framework aims to help architects for representing then to explore their abstract intentions beyond geometric features at an early design stage.

KEYWORDS
Design intention; parametric design; generative modeling; design pattern; algorithmic framework

1. Introduction

Confusion concerning methods, thinking, and techniques among parametric, generative, and algorithmic approaches has emerged with the appearance of such new digital design tools as Grasshopper and Dynamo. Leach claims that one reason for this confusion is that persons in the architectural domain are unfamiliar with the computer science [9], and refers to the differences between parametric and algorithmic design, where parametric techniques are based on the manipulation of geometric forms, while algorithmic design is based on the use of programming codes. But regardless of whether through the manipulation of forms or the use of code, architects should be able to use those digital tools to solve architectural design problems. Kotnik suggested that computational architectural design should involve "exploring computable functions," which should take design information as input parameters and buildings' properties as output variables [8]. Parametric design implies that an algorithm is fixed, and that output variables are consequently predictable from parameters. Generative modeling implies that output variables are not only controlled by input parameters, but also by flexible and adjustable functions. As a consequence, the computable functions used in various architectural disciplines, i.e. the algorithms for solving various architectural design problems, should be the key to resolution of many digital architectural design issues.

In the domain of computer science, an algorithm is a process for solving a problem in a finite number of steps. Algorithmic modeling tools such as Grasshopper were developed to automate and accelerate 3D modeling tasks by applying generative algorithms. However, cognitive research has revealed that designers typically apply algorithms only as means of exploring geometric intentions, and prefer to apply known solutions and design patterns for other, non-geometric issues [16]. When designers' intentions go beyond geometry, regardless of the type of design objective [4], designers need to find or develop appropriate algorithms before they can implement generative or evaluative scripts. Algorithmic modeling is gradually being applied to the generation of complex forms, multiple objective optimizations, and the control and evaluation of building performance. One of the reasons for this is that the relevant algorithms, including mathematical formulas for complex geometries, metaheuristic algorithms for artificial intelligence [15], structural analysis, and energy consumption formulas, have been validated in relevant domains. The task of algorithmic design has thus become the implementation of algorithms in modeling tools, rather than the interpretation of architectural design problems and derivation of solving algorithms. Since there is insufficient guidance

CONTACT Chieh-Jen Lin t60011@mail.tut.edu.tw

and assistance for converting architectural knowledge into algorithmic scripts, it is not surprising that designers prefer to apply known solutions, rather than develop or implement algorithmic scripts on their own.

While fewer architects are directly employing Alexander's pattern language [1], more software engineers are applying "design patterns" in identifying and reusing the best practices in known situations. Based on design patterns, such as the model-view-controller (MVC) pattern, application frameworks have therefore been developed to facilitate and accelerate the development of applications. For example, Ruby on Rails, Symfony, and Django have been developed for web-based applications in different programming languages. While programming/scripting skills have become more critical when applying algorithmic tools in parametric architectural design, developing an algorithmic framework for the exploration and development of algorithms can help architects to focus on solving design problems, rather than on programming/scripting tasks. Based on previous studies proposing information conversion patterns employing Building Information Model (BIM) schema, which involve semantic, topological, and geometric information [11], this paper proposes an algorithmic framework to help architects explore and develop algorithms going beyond geometric intentions.

2. An algorithmic framework for parametric architectural design

Unlike other studies focusing on algorithmic programming patterns [14], or architects' geometric intentions [12], this paper proposes an algorithmic framework that aims to help architects to connect abstract design intentions by integrating semantic ontologies and topological algorithms. Based on the STG pattern proposed in previous studies [11], this framework is divided into three parts: (1) a semantic module that can help architects to indicate design objects and their semantic relations as the "Model" module in a MVC pattern, (2) a topological module that presents the topological algorithms of given semantic relations as the "Controller" module in a MVC pattern, and (3) a geometric module that presents the visual validation of topological algorithms as the "View" module in a MVC pattern. This "Semantic-Topological-Geometric framework" (STG*f*) realizes an algorithmic framework by applying Grasshopper and the GhPython plugin as an algorithm-aided design tool [13].

2.1. Semantic components as representing models of design intentions

Semantic ontology is a knowledge representation technique in the artificial intelligence (AI) domain. One of the most popular tools for authoring semantic ontology is Protégé, which is based on the Ontology Web Language (OWL) originally used to develop semantic networks. By applying OWL reasoner plugins, such as FaCT++ and HermiT, Protégé can validate an OWL-based ontology in order to ensure that it is correct and consistent. Furthermore, the semantic web rule language (SWRL) plugin can be used to express logic rules in order to infer implicit knowledge within the ontology. To convert the semantic ontology of architects' design intentions into generative algorithms, an ontological technique based on Protégé was incorporated into the semantic module of STG*f*.

The first module of STG*f* is the "Semantic" component, which consists of semantic information concerning design intentions. In an MVC-based application, the Model module is used to capture behavior and logical rules in the problem domain. In order to associate generative algorithms with architectural design intentions, this module must first represent design intentions in a computable format. At an early design stage, design intentions usually consist of abstract, textual descriptions concerning various design objects and their relationships, but only the essential semantic information regarding building components can be predefined in BIM applications and Industry Foundation Classes (IFC) schema. However, although Rhino has no predefined semantic schema for building components in architectural design, an architect may define or interpret his/her unique design objects and relationships, which cannot be predicted by BIM or IFC during an early design stage, when design situations have not yet emerged. Rhino therefore needs a contextual semantic ontology [6], which is a computational format for representing, storing, and validating a semantic ontology of domain knowledge in order to convert architects' design intentions into generative parametric design algorithms.

2.2. Topological modules as generative controllers of design intentions

Although ontological techniques can validate the conceptual consistency of design intentions, they cannot guarantee that relevant instances of semantic concepts will also comply with the necessary properties. For example, a partial ontology may indicate that an "OpenSpace" class has an "ExistingTree" property allowing the inference of the "GoodQuality" property of the "StaticLeisureActivity" class for seniors. However, it is also necessary to calculate the correlations between the existing trees and the instance of the park in order to determine whether those existing trees are located within the scope of the park. As a consequence, in addition to their definitions of conceptual classes within the ontology, topological relationships must also address more feature properties in specific instances.

The second module of STG*f* is a "Topological" component, which consists of a controlling algorithm for validating design intentions. In an MVC-based application, the "Controller" module is used to accept operations from users to modify the data within models, and therefore controls interactive behaviors among different models in a system. Eastman suggested that topologies are the mathematical relationships and fundamental definitions of parametric models in BIM [5]. At an early design stage, topological relations expressing design intentions are usually abstract, and may consist of enclosure, extension, and concentration of indoor/outdoor spaces and building masses [7]. Topological relations among design objects defined within the "Semantic" module can thus be regarded as "controllers" of design intentions.

2.3. Use of the geometric modules to validate design intentions

Since Grasshopper aims to generate 3D models through given algorithms, it seems unnecessary to add other geometric functions. To provide visual validation of whether a design intention has been accomplished, however, Grasshopper should provide more visual clues for users, such as textual and numerical tags and colored previews. The final module of STG*f* is the "Geometric" component, which can validate views of design intentions. In an MVC-based application, a "View" module is used to display information concerning a retrieved "Model" and the results of "Controller." In architectural design, architects always need visual feedback to validate the content of semantic ontologies or the topological behaviors of generative algorithms. Immediate visual feedback concerning generative algorithms, which helps users to validate algorithms, is one of the most attractive features of Grasshopper. For design intentions other than geometric features, especially in the case of invisible or non-obvious intentions such as outdoor spaces and mathematical ratios, geometric features can not only be input as parameters in generative algorithms, but also be generated for the visual validation of design intentions. The "Geometric" module of STG*f* therefore aims to demonstrate how to input geometric objects from Rhino into the STG*f* "Semantic" and "Topological" modules, and how to provide visual clues for the validation of users' design intentions.

2.4. Summary

By providing rewritable example scripts and adjustable algorithmic modules, which are editable clusters of algorithmic components in Grasshopper, the STG*f* algorithmic framework aims to help architects to connect abstract design intentions by integrating semantic ontologies and topological algorithms. Based on the STG pattern proposed in previous studies, the STG*f* framework is divided into three parts: (1) a semantic module that can help architects to indicate design objects and their semantic relations as the "Model" module in a MVC pattern, (2) a topological module that presents the topological algorithms of given semantic relations as the "Controller" module in a MVC pattern, and (3) a geometric module that presents the visual validation of topological algorithms as the "View" module in an MVC pattern. This algorithmic framework is realized by applying the GhPython plugin of Grasshopper as an algorithm-aided design tool [13].

3. Initial test of the STG framework

Our prototype STG*f* framework was based on the results of previous studies, which included some GhPython components used to hook the Resource Description Framework (RDF) files of semantic ontologies from Protégé [10], example topological algorithmic clusters, such as "Adjacent," "Enclosed," and "OpenTo" topologies, and scripts for filtering and automatically collecting geometric features from the models in Rhino. We provided this STG*f* prototype to students for modeling their design intentions concerning the "Community Library and Public Spaces" design proposal, which was used as the rapid-design topic on Taiwan's 2016 architect qualification exam.

3.1. Example of design contexts

"Community-friendliness" issues have been design subjects on Taiwan's architect qualification exam for three consecutive years (Fig. 1). Unlike the contexts of the previous two years (Fig. 1.a, b), however, the site on the 2016 exam consisted of an irregular shape, which was four times larger than the required building area, had no existed buildings, and had a complex traffic context (Fig. 1.c). The building proposal on the 2016 exam consisted of a library, some rental stores, and an underground parking lot for the community. The exam's "community-friendliness" issues concerned how to arrange the relationships between the building and the community in order to create high-quality spaces facilitating community activities. However, the outdoor spaces shaped by the building and its surroundings are ignored in BIM and IFC semantic ontologies, which mainly focus on the physical components of a building itself. In addition, the question of what relationships between a building

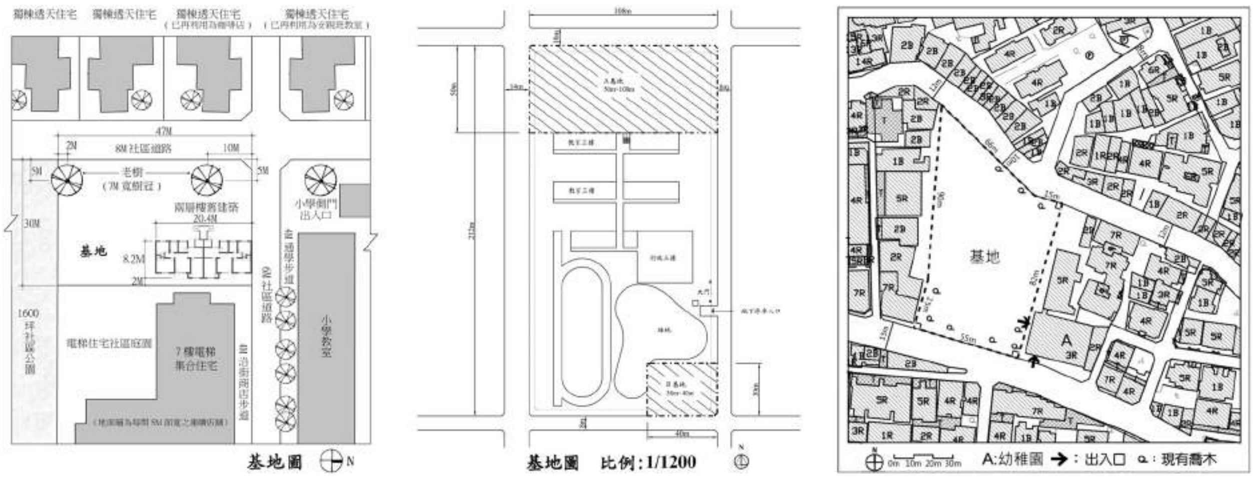

Figure 1. Three different site contexts relating "community-friendliness" issues in Taiwan's architect qualification examinations: (a) An architect firm as the cornerstone of a good neighborhood in 2014, (b) a community-friendly elementary school in 2015, and (c) a community library and public spaces in 2016.

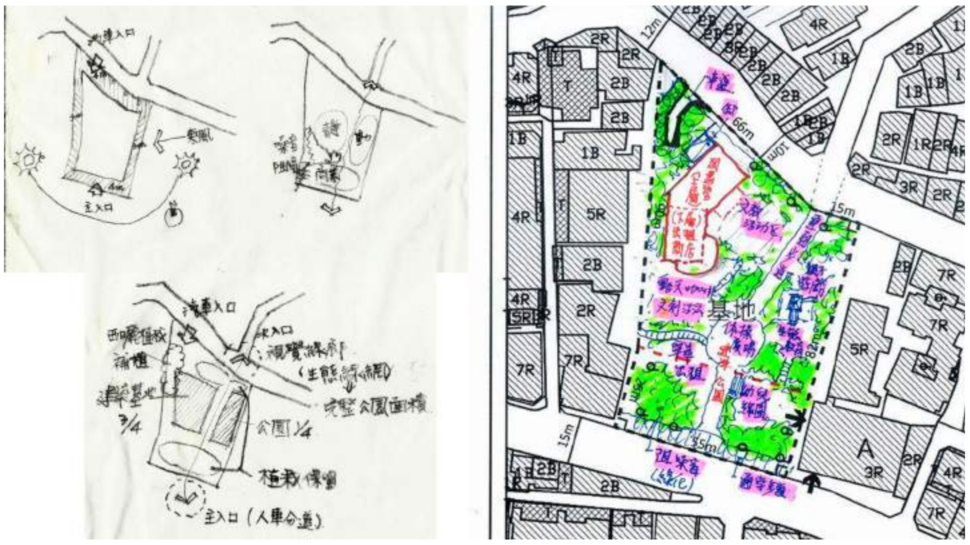

Figure 2. Similar design intentions proposed by two experienced students who had taken Taiwan's 2016 architect qualification exam.

and its surroundings will effectively facilitate community activities still leaves much room for discussion among architects.

3.2. Analysis of design intentions

Since two students had taken this rapid-design exam, but were not familiar with the use of Grasshopper, these two individuals were asked to provide their sketches in order to recall their design intentions on the exam. Other students, who were more familiar with Grasshopper, were asked to compose algorithms concerning common design intentions by applying STG*f*.

As the answer sheets of Taiwan's architect qualification exam cannot be obtained, both of the experienced students who had taken the exam are asked to redraw their proposals in the exam, but also write down their ideas about the proposals (Fig. 2). Other students are first asked to analyze those textual and graphic annotations, and then to build a semantic ontology of design intentions about "community-friendliness" in STG*f*. Based on the view of parametric architectural design, this paper focuses on those intentions in response to the given sites' contexts. It is found some similar intentions concerning "community-friendliness" issues in response to site contexts on the exam. These common intentions included (Fig. 2): (1) the ramp down to the underground parking area should be located at the northwest corner; (2) a passage should be left through the site to connect the northern and southern alleys; (3) two open spaces should be left around two groups of existing trees for different levels of activity intensity in the case of one static leisure park located in the west, and another dynamic recreation park in the east; (4) the building mass should be located in the north-west corner over the possible range of underground parking, and (5) in this building mass, rental stores should be located on the ground floor, and the library should be located on higher floors. Since this paper focuses on design intentions other than geometric building forms, the framework's testers were asked to compose algorithms using the site's contextual factors as input parameters.

3.3. Implementation of generative algorithms

One of the benefits of STG*f* is that it can achieve a hierarchical structure of design intentions by establishing a semantic ontology. For example, the underground parking should connect to the road by a ramp, which therefore restricts by the location of underground parking. The library building should be above the underground parking, which therefore restricts the location of the building. The static leisure park should be close the library building, which therefore restricts the location of the static leisure park. After semantic analysis of the five design intentions mentioned above, the first, which concerned the location of the ramp entrance, therefore exerted a critical influence on the other three. Consequently, the first intention for the selection of the ramp entrance is firstly chosen to implement its algorithm.

In order to complete an algorithm, which should take the site's contexts as input parameters, students are asked to proposed topological relations between the location of the ramp entrance and the features of design contexts, such as traffic conflicts. Accordingly, students are further asked to analyze the computable relationship between the ramp location and the features of site's contexts. It is found that the first intention was based on two sub design intentions: (1) traffic on the narrower street was relatively light, and (2) a ramp entrance close to the starting point of the construction line could reduce conflicts with pedestrians. Although there are other sub-intentions proposed by students, such as to keep away from kindergarten to protect children. However, these two semantic intentions can be more easily understood and modeled as topological algorithms: (1) selecting the "Narrowest" but more than 6 meter-wide-road around the site, and (2) calculating the "Start Point" of the construction line on the selected street in a "Clockwise" direction. Since vehicles are driven on the right side of the road in Taiwan, the "Start Point" of the construction line should be in a "Clockwise" direction (Fig. 3). By applying these algorithms, users can input the contour of the site and the borderlines of the streets around the site in order to generate a suggested location for the ramp entrance. And while since there is no default semantic ontology for architectural design in Rhino, STG*f* algorithms can indicate the semantics of input geometric features by manual or semi-automatic filter in the case of such other features as specified layer name, object's name, or geometric type. These assigned semantics can remind users of the original design intentions.

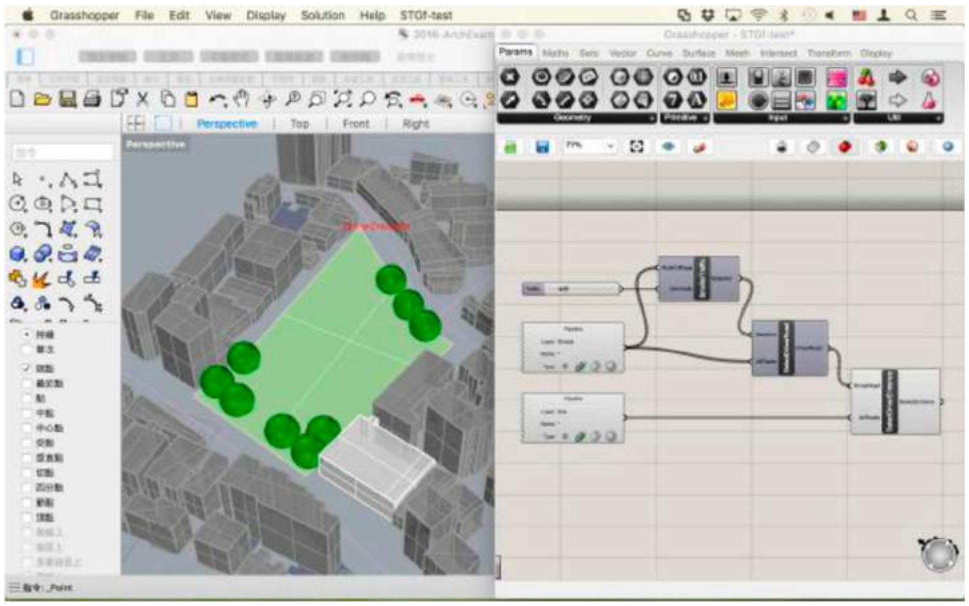

Figure 3. An example algorithm is used to select a parking facility entrance into the site able to reduce traffic conflicts.

4. Discussion

Computational architectural design should apply algorithms to solve specified architectural design problems, which should not be limited to merely the generation of complex geometric forms. Even though most solutions to architectural design problems must be represented using the geometric features of building components, at least the input parameters of algorithms should not be limited to just numbers and geometric features, which are too abstract to be associated with architectural knowledge. However, developing an algorithm usually requires more prior knowledge of programming skills than architectural design knowledge. As an application framework can facilitate and accelerate the development of applications, STG*f* also contained collected sample Grasshopper scripts to help users develop their own algorithms for representing their design intentions and solving design problem. STG*f* is discussed as follows based on this view of computational architectural design.

4.1. Semantic models for architectural design concepts

In the domain of architectural, engineering, and construction (AEC), IFC schema occupy the highest level of semantic ontology [6] for facilitating information exchange and knowledge sharing among different disciplines in AEC. However, the ontological knowledge used in BIM is based on IFC and focuses on the physical components of a building, and the semantics of both BIM and IFC usually lack the flexibility to handle contextual situations at different design stages. At an early stage, design intentions are usually textual descriptions, and are too abstract to be input as parameters, let alone to find an algorithm for generating or validating them. By employing semantic ontology techniques, STG*f* provides computational components to hook Protégé via RDF files in order to represent, store, and validate abstract intentions and their related design knowledge. Since a user-defined semantic ontology in STG*f* is contextual and situational, semantic models of design intentions therefore can be converted into possible parameters that can be input into algorithms.

As for the aforementioned issue of "community-friendliness," candidates taking the architect qualification exam were asked to propose a contextual ontology concerning their intentions for achieving "community-friendly" spaces. For example, a candidate taking the 2016 exam could propose the idea of preserving existing trees in order to rapidly form a static leisure park for seniors, while another candidate might suggest installing more game and sports equipment in order to improve the health of both seniors and children. If those design intentions could be represented in OWL, then the logic reasoners in Protégé could help architects to validate, keep consistency, and to infer implicit intentions in the ontology. More importantly, abstract design intentions can be converted into a computational format allowing them to be input as parameters in generative algorithms.

4.2. Topological controllers of architectural design intentions

Algorithmic thinking means to understand the interpretive correlations between algorithms and design intentions [2]. The critical relationships among design objects can be obtained from the corresponding semantic triple set, which consists of a subject, predicate, and object within a contextual ontology of design intentions. The "predicate" of an ontological triple set is actually the interpretive correlation between the subject and the object. STG*f* retrieves predicates from semantic ontologies of design intentions as the names of topological functions. Even though STG*f* cannot directly implement a computational function, however, these names cannot only help users to associate relevant design intentions, but also indicate the due behavior of this function. Cognitive studies have found that designers tend to spend more effort to find appropriate algorithms fitting their intentions [16], rather than testing possible outputs by modifying parameters. To indicate the due behavior of a function via its name is found to facilitate finding appropriate algorithms and to validate the functional behavior of the algorithms.

However, topological relationships may only be mathematical, but not directly involve geometric features. As an example, the Taiwan's 2016 architect qualification exam asked candidates to use three quarters of the given site as a community park. The ratio of land usage was only mathematical, and was independent of the shape, position, and other geometric features. In addition, land usage ratios required the calculation of correlations among different usages of the site, which were situated properties and could not be asserted within an ontology in advance. The "Topology" module in STG*f* therefore serves as a key controller matching the semantic ontologies of design intentions with the geometric features of design objects. Most abstract design intentions still need to be validated visually through the geometric features of design proposals. Even in the case of mathematical topologies such as land usage ratios, it is better for architects to visually inspect this kind of abstract concept than to rely on exclusively on text or numbers.

4.3. Geometric views of architectural design algorithms

Regardless of design intentions, the geometric features of design objects, including physical building components and void indoors/outdoors spaces shaped by the building and its surrounding, should be the ultimate goal of algorithmic modeling. Thanks to the strong ability of Grasshopper to generate 3D models using algorithms, the problem is therefore not how to generate a model, but why the geometric features of a model should be the way they are. This is not just a matter of aesthetics; the best way to validate whether an algorithm reflects a design intention should be through the generated model. However, design intentions sometimes cannot be directly recognize by generated models. For example, the public/private domain and static/dynamic recreation levels of spaces must be interpreted and judged on the basis of the site situation and design context, and sometimes cannot be directly recognized in the geometric features of 3D models. In these circumstances, the use of different colors to visualize the levels of public/private domain or static/dynamic recreation should be able to help designers to determine whether an algorithm has responded to his/her intentions or not. In order to perform the effective visual validation of topological algorithms, there must be more visual clues other than vectors, curves, surfaces, and other geometric features.

For example, the site context employed in the 2016 architect qualification exam included two adjacent parallel streets on the north and south sides, and the presence of a kindergarten adjacent to the southeast corner and the south street. Based on the "community-friendly" concept, the site needed passages allowing children to cross the site to return to their homes on the north side. In this situation, the critical feature of the proposed passages did not consist of a certain geometric shape, but that the passages' nodes had to connect the entrances of the kindergarten with the street on the north. More algorithmic explorations were therefore needed to determine how to check whether the connecting topology was accomplished through the input geometric features of the proposed passages. However, developing algorithms inevitably involves programming skills, understanding of data structures, and other computer science knowledge, which sometimes go beyond the domain knowledge of architectural design. Providing demonstrations and guidance through examples is therefore more helpful to architects than to teaching them programming skills and computer science knowledge from scratch. The "Geometric" module of STG*f* consequently provides rewritable sample scripts and adjustable algorithmic modules, which can help architects to explore their own intentions beyond geometric features of design objects.

5. Conclusions

While critics have suggested that the use of generative tools can result in the complexification of simple things [3], generative algorithms should potentially be able to go beyond geometric intentions. However, programming and computer science knowledge typically becomes the biggest obstacle for architects who wish to implement algorithms of their intentions beyond geometric knowledge. Since software frameworks can dramatically simplify and accelerate the development of an application, the STG*f* framework proposed in this paper can also help designers to simplify and accelerate the development of generative algorithms used in parametric architectural design.

One purpose of the MVC pattern is to divide complex systems programming tasks into independent objects. The STG*f* framework divides parametric design into three procedural algorithmic steps, and can implement generative algorithms developed by different designers/scripters. As building projects become more complex, instead of requiring architects to possess knowledge in other domains, it would be better to hand over programming/scripting tasks to professional scripters, and performance optimization to MEP engineers. The time has therefore come to embed architects' design intentions in the parameters, variables, and algorithms used in parametric architectural design. However, since there still have no effective version control system for a graphic programming language like Grasshopper, there are technical difficulties in tracking users' processes of developing an algorithm in Grasshopper. Post commentary and textual annotations in Grasshopper are the only data that can be accumulated at present. Therefore, one of the future works is to analysis the accumulated information in order to improve the functions of STG*f*.

Acknowledgements

The Ministry of Science and Technology of Taiwan supported this paper under grant number MOST 105-2221-E-165-002.

ORCID

Chieh-Jen Lin http://orcid.org/0000-0001-9981-5864

References

[1] Alexander, C.; Ishikawa, S.; Silverstein, M.: *A Pattern Language : Towns, Buildings, Construction*, Oxford University Press, New York, 1977.

[2] Bazalo, F.; Moleta, T.J.: Responsive Algorithms - An investigation of computational processes in early stage design, in: Proceedings of the 20th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA 2015), Daegu, 2015, 209-218.

[3] Burry, M.: *Scripting Cultures: Architectural Design and Programming*, John Wiley and Sons, Ltd., Chichester, UK, 2011.

[4] Chang, M.-C.; Shih, S.-G.: A Hybrid Approach of Dynamic Programming and Genetic Algorithm for Multi-criteria Optimization on Sustainable Architecture Design, *Computer-Aided Design and Applications*, 12(3), 2014, 310-319. http://doi.org/10.1080/16864360.2014.981460

[5] Eastman, C.; Teicholz, P.; Sacks, R.; Liston, K.: *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*, 2nd ed., John Wiley & Sons Inc., Hoboken, N.J., 2011. http://doi.org/10.1002/9780470261309

[6] Gursel, I.; Sariyildiz, S.; Stouffs, R.; Akin, Ö.: Contextual Ontology Support as External Knowledge Representation for Building Information Modelling, in: T. Tidafi, T. Dorta (Eds.) Joining Languages, Cultures and Visions: CAADFutures 2009, PUM, 2009, 487–500.

[7] Ho, H.-Y.; Wang, M.-H.: Meta Form as a Parametric Design Language, in: eCAADe 2009, Istanbul, Turkey, 2009, 713-718.

[8] Kotnik, T.: Digital Architectural Design as Exploration of Computable Functions, *International Journal of Architectural Computing*, 8(1), 2010, 1-16. http://doi.org/10.1260/1478-0771.8.1.1

[9] Leach, N.: Parametrics Explained, *Next Generation Building*, 1(1), 2014, 33–42.

[10] Lin, C.-J.: Design Criteria Modeling - Use of Ontology-Based Algorithmic Modeling to Represent Architectural Design Criteria at the Conceptual Design Stage, *Computer-Aided Design and Applications*, 13(4), 2016, 549-557. http://doi.org/10.1080/16864360.2015.1131551

[11] Lin, C.-J.: The STG pattern – Application of a “Semantic-Topological-Geometric” Information Conversion Pattern to Knowledge-Based Modeling in Architectural Cnceptual Design, *Computer-Aided Design and Applications*, 14(3), 2017, 313-323. http://doi.org/10.1080/16864360.2016.1240452

[12] Su, H.-P.; Chien, S.-F.: Revealing Patterns: Using parametric design patterns in building façade design workflow, in: Proceedings of the 21st International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2016), Melbourne, 2016, 167-176.

[13] Tedeschi, A.; Wirz, F.; Andreani, S.: AAD_Algorithms-Aided Design : parametric strategies using Grasshopper, Le Penseur Publisher, Brienza, Italy, 2014.

[14] Woodbury, R.: *Elements of Parametric Design*, Routledge, New York, 2010.

[15] Wortmann, T.; Nannicini, G.: Black-Box Optimisation Methods for Architectural Design, in: 21st International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2016), Melbourne, 2016, 177-186.

[16] Yu, R.; Gero, J.; Gu, N.: Architects' Cognitive Behaviour in Parametric Design, *International Journal of Architectural Computing*, 13(1), 2015, 83-102. https://doi.org/10.1260/1478-0771.13.1.83