When the Geometry Informs the Algorithm

A hybrid visual/textual programming framework for facade design

Inês Caetano[1] *, António Leitão*[2]
[1,2] *INESC-ID/Instituto Superior Técnico, University of Lisbon*
[1,2] *{ines.caetano|antonio.menezes.leitao}@tecnico.ulisboa.pt*

Facade design is becoming increasingly complex, forcing architects to more frequently resort to analysis and optimization processes. However, these processes are time-consuming and require the coordination of multiple tools. Algorithmic Design (AD) has the potential to overcome these limitations through the use of algorithms implemented in Textual Programming Languages (TPLs) or Visual Programming Languages (VPLs). VPLs are more used in architecture due to their smoother learning curve and user-friendliness, but TPLs are better suited than VPLs for handling complex AD problems. To make TPLs more appealing to architects, we incorporated VPLs' features in the textual paradigm, namely, Visual Input Mechanisms (VIMs). In this paper, we propose an extension to an existing AD framework for the design exploration, analysis, and optimization of facades to support a TPL-based approach that handles VIMs.

Keywords: *Algorithmic Design, Facade Design, Textual Languages, Visual Input*

INTRODUCTION

Architectural design needs to consider the currently growing design constraints, e.g., environmental (Picco et al. 2014), structural (El Sheikh 2011), and aesthetical (Schulz 1971; Schittich 2006), often forcing architects to resort to analysis and optimization processes (Machairas et al. 2014), although with two serious limitations: (1) these processes are typically time-consuming and, as the available time is usually short, the final solutions are often only partially improved (Nguyen et al. 2014); and (2) analysis processes involve multiple tools with different data formats, causing information losses and the accumulation of design errors resulting from imperfect data transfers between tools (Leitão et al. 2017). These shortcomings hinder the adoption of analysis/optimization processes, confining their application to latter design stages (Turrin et al. 2011; Shi 2010) where, unfortunately, their impact in the solution's performance is limited (Konis et al. 2016). Algorithmic Design (AD) promises to overcome these limitations by facilitating the analysis and optimization of architectural designs (Alfaris and Merello 2008) from early stages. AD uses algorithms to create designs, allowing for more complex design solutions, facilitating changes, and automating repetitive tasks, in particular, the *generation-analysis-regeneration* cycle, typical of optimization processes.

AD can be implemented using Textual Programming Languages (TPLs) or Visual Programming Languages (VPLs). Architects tend to prefer VPLs not only due to their visual nature, but also because TPLs are less intuitive, as well as more difficult to learn and use than VPLs. Additionally, TPLs rarely support the Visual Input Mechanisms (VIMs) of most VPLs that allow for the direct use of graphical entities, such as points, curves, and surfaces, as inputs to the AD program. However, TPLs show greater potential for large-scale designs (Leitão et al. 2012; Celani and Vaz 2012; Wortmann and Tunçer 2017) and, once mastered, they tend to be more productive. Given TPLs' and VPLs' different advantages, it is appropriate to consider their combination, either by incorporating textual programming components in VPLs, as demonstrated by *Grasshopper*'s *Python* component, or by extending TPLs with VIMs (Sammer et al. 2019). The latter will be the focus of this paper.

We propose extending a TPL-based AD framework, whose domain of application is building facades, to support VIMs. The extended framework provides several predefined algorithms for the design, analysis, and optimization of facades, now coupled with the most relevant VIMs for specifying the constraints that affect facade design.

In the past, this framework addressed architectural problems integrating environmental factors (Caetano et al. 2018) and design rationalization (Caetano and Leitão 2018). In this paper, we focus on its combination with VIMs. The research entailed three stages: (1) studying the differences between a Visual Input (VI) and a Textual Input (TI); (2) understanding how a VI can be incorporated within a text-based AD approach; (3) implementing a set of VIMs in the algorithmic framework for facade design.

BACKGROUND

AD requires programming experience, which most architects do not have. Fortunately, several Programming Languages (PLs) were developed to facilitate the learning process: some are textual, such as *Processing*, *Racket*, and *Python*, but the most popular ones among architects are visual, such as *Generative Components*, *Dynamo*, and *Grasshopper*. They allow users to create AD programs by manipulating graphical elements (Janssen 2014), thus being more intuitive and productive for beginners when compared with TPLs (Zboinska 2015). VPLs also integrate features that facilitate the programming task, namely traceability, real-time feedback (Leitão et al. 2014), and VIMs.

Despite their appeal, VPLs still present several limitations: (1) they lack scalability, making large programs difficult to understand and modify (Leitão and Santos 2011; Janssen 2014); (2) they confine users to the predefined modules/components available in each VPL, hindering solutions that need more advanced features (Zboinska 2015); and (3) they lose their real-time feedback capabilities when dealing with more computationally-intensive designs (Leitão et al. 2014). As a result, for complex design problems, architects tend to use TPLs (Leitão et al. 2012), proving the inability of VPLs for long-term use (Noone and Mooney 2018).

Unfortunately, it is difficult to transition from a VPL to a TPL, and more so when one is accustomed to VIMs, since these mechanisms are rarely available in TPLs. A possible strategy to facilitate this transition and flatten TPLs' learning curve is to incorporate some VPLs features in TPLs to make them more intuitive and, thus, more adequate for teaching purposes; a perspective already addressed in the past (Sammer et al. 2019). In this paper, we go further by extending a TPL-based AD framework for facade design to also support VIMs, while preserving its capability to generate, analyze, and optimize large-scale designs. In the following sections we explain the framework and the proposed extension, and we evaluate it with a case study.

ALGORITHMIC FRAMEWORK FOR FACADE DESIGN

Any AD framework must address the typical variability of architectural design in a way that a computer understands. Computers operate upon a set of instructions transmitted via PLs and, while the first PLs were difficult to learn and use, the current ones are getting closer to the universal language of mathematics. Therefore, the implementation of the AD framework for facade design considered the formalism of mathematics, being organized in a fivefold classification: (1) *Geometry*, to define the overall geometry of the building facade; (2) *Pattern*, to explore several geometric facade patterns through the combination of different shapes and geometric transformations; (3) *Distribution*, to create different facade grids and pattern distributions; (4) *Optimization*, to improve the design according to fitness goals; and (5) *Rationalization*, to make a balance between the conceptual design intent and the solution's feasibility. Each category contains a set of predefined algorithms that are portable between different modelling tools, namely *AutoCAD*, *Rhinoceros*, and *Revit*, and analysis tools, namely *Robot* and *Radiance*, among others. Despite originally supporting only TIs, in this paper, we extend the AD framework to also deal with VIs.

In general, a VI represents a manually created geometry in a design tool, whereas a TI comprises a textual description. While the latter can be directly combined with any algorithm available in the framework, the former needs to be first converted into a format suiting the algorithm. To this end, we extended the AD framework with some VIMs that allow the use of shapes created in the modelling tools as input to the framework's algorithms. These functionalities apply principles explored in previous research (Sammer et al. 2019), thus evaluating their suitability for architectural design and, more specifically, facade design.

The available VIMs rely on the selection of one or more shapes in the modeling tool, which are then provided as VI to the algorithms. During this process, users can decide whether they want to preserve a dependency between the AD program and the selected shapes. In the affirmative case, the AD program is automatically rerun every time the input shapes are changed in the modeling tool; otherwise, it only considers these changes when the shapes are re-selected. For the latter scenario, the framework resorts to *metaprogramming* techniques, i.e., the use of programs to generate other programs (Czarnecki et al. 2002) that, in this case, correspond to equivalent algorithmic descriptions of the existing VIs in the modeling tool. To make the AD program independent from a given VI, we replace the program fragment representing the VIM with the generated program fragment reproducing that same VI. This allows the latter to be stored in the AD program, no longer reacting to changes made to its original shape, and to be reproduced either in the same or in another modeling tool. This enables the framework to have a general application, not restricting the available functionalities to a specific tool, while permitting to take advantage of VIs produced in multiple modelling environments. Nevertheless, the resulting algorithmic descriptions are non-parametric, which means they represent and reproduce only the original VI and not variations of it. Further details on the available functionalities and their implementation can be found in Sammer et al. (2019).

EVALUATION

In this section, we evaluate the framework in a case study, following a three-phase process: first, we model the original case study using a purely AD approach; then, we explore different design variations of it by applying only TIs; and, lastly, we generate the same design variations but using only VIs. In this evaluation, we use *Khepri* (Leitão et al. 2019), a descendent of the AD tool *Rosetta* (Lopes and Leitão 2011).

Case Study

Our case study is inspired by the facade of the *Formstelle* building designed by *Format Elf Architekte*, which is composed by several hexagonal apertures of different sizes, creating a gradual stain in the facade's central area (figure 1, top). We selected this case study due to its large design space, that can be easily explored using VIs. To reproduce this design using the algorithmic framework, we select algorithms from: (1) the *Geometry* category, to produce the surface's spatial locations on which we want to distribute the hexagonal openings (figure 1A); (2) the *Distribution* category, to arrange the apertures in a rhombus mesh (figure 1B); and (3) the *Pattern* category, to

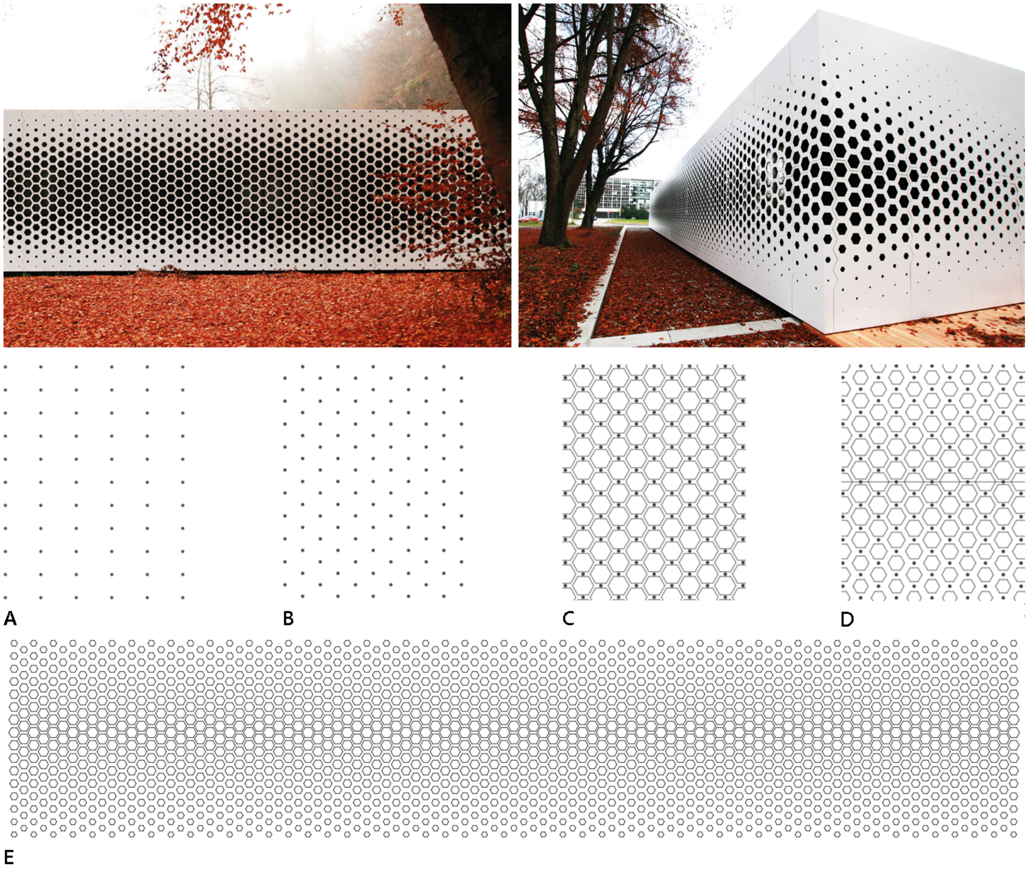

Figure 1
Top: Formstelle building in Töging am Inn, Germany (courtesy of Format Elf Architekten); Bottom: the case study's algorithmic development: A. surface original points; B. rhombus mesh creation; C. hexagonal elements' placement; D. hexagonal elements' size variation; E. final facade design.

create a hexagonal aperture in each location (figure 1C) and to control their size according to a rule that, in this case, corresponds to the elements' distance to the facade's horizontal axis (figure 1D). Then, we combine these algorithms through function composition techniques: we use the first algorithm (straight) as input for the second one ($\text{grid}_{\text{rhombus}}$), which returns a set of locations distributed in a rhombus-shaped grid; then, these locations are used as input for the other two algorithms (regularPolygon and scale), which create the hexagonal elements, in the first case, and control their size according to their distance to the facade's horizontal axis, in the second case.

Geometric Exploration

This section focuses on the application of design changes to our case study following two strategies: one using only TIs and the other using VIs.

Textual Inputs. At this stage, we explore several variations of our case study using only TIs. To obtain the original design (figure 1E), we combine the scale

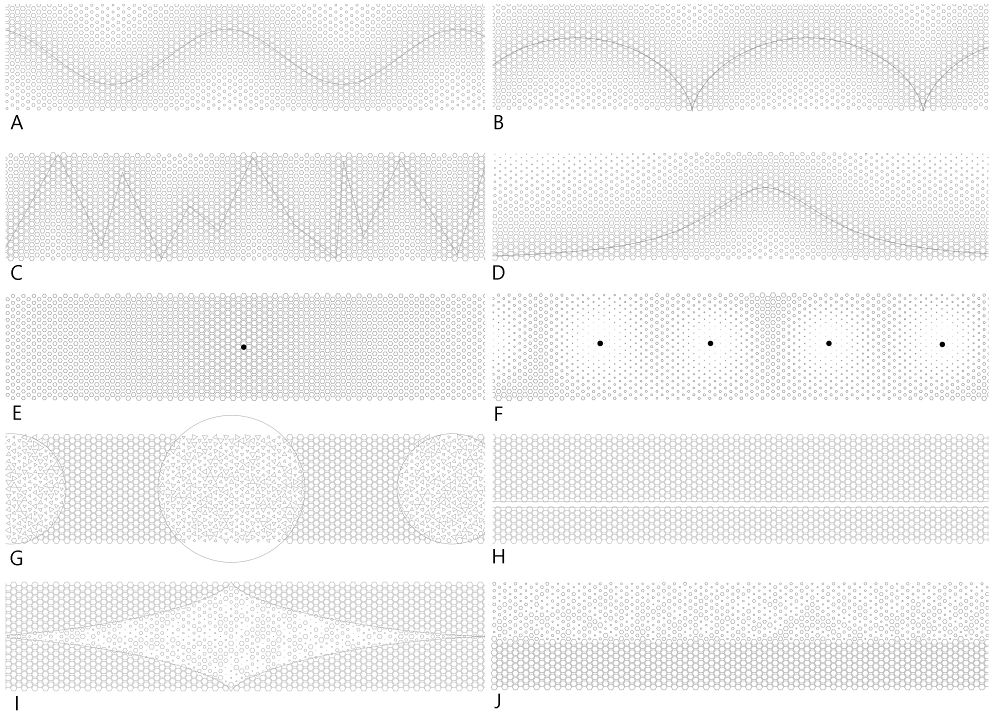

Figure 2
Case study design variations using TIs: defining curves (A-D) or points (E-F) to change the size of the apertures, or areas to apply certain design variations (G-J).

algorithm with another one acting as an *attractor* ($\text{attractor}_{\text{curve}}$), which receives as input a curve and an attraction intensity factor (from 0 to 1) and returns, for each facade element, a value that differs according to its distance to the curve. We provide this algorithm with another one describing a horizontal straight curve.

To change the original design, we can use other curves (figures 2A to 2D) or use the $\text{attractor}_{\text{points}}$ algorithm and set one (figure 2E) or more points (figure 2F) as attractors. We can also select the facade areas where we want to apply design variations, like creating elements of different shapes (figure 2G) or sizes (figures 2I and 2J) or simply having no elements (figure 2H). For this we have the $\text{affected}_{\text{areas}}$ algorithm, which receives closed curves delimiting the area(s) to change and the transformation rules to apply and their intensity factors. In practice, this algorithm tests if each element matches the given areas and, in case it does, it applies the transformation(s) to it, otherwise, it keeps the element unchanged.

Visual Inputs. To change our case study using VIs, we have to manually produce in the design tool different shapes according to the requirements of the selected algorithms, e.g., points (figure 3E) and curves (figures 3A to 3D) in case we use the $\text{attractor}_{\text{points}}$ and $\text{attractor}_{\text{curve}}$ algorithms, or closed curves delimiting areas in case we apply the $\text{affected}_{\text{areas}}$ algorithm (figures 3F to 3J). Note that, here, the $\text{attractor}_{\text{points}}$, $\text{attractor}_{\text{curve}}$, and $\text{affected}_{\text{areas}}$ algorithms receive as input one or more geometric entities manually produced in the modelling tool. To allow this, the framework resorts to *metaprogramming* techniques to convert non-algorithmic elements into

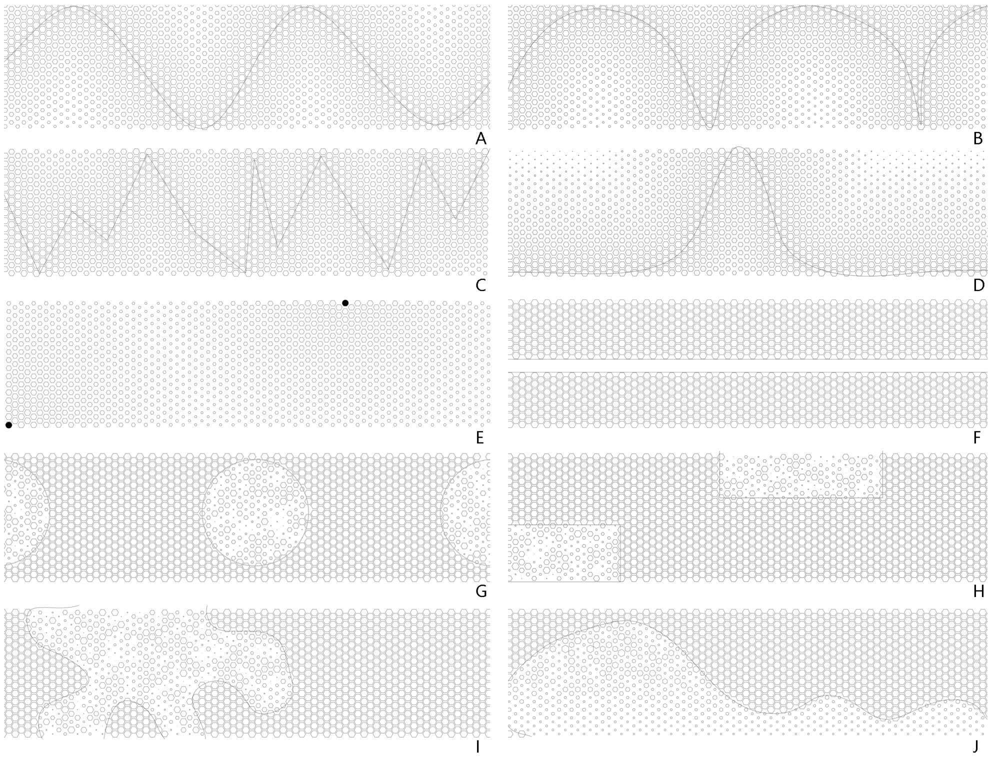

Figure 3
Case study design variations using VIs: creating curves (A-D), points (E) in the modeling tool to change the apertures' size, or areas to apply different design variations (F-J).

algorithmic ones. The resulting scenario enables us to constantly interact with the design tool and iteratively combine the VIs with the framework algorithms, including analysis and optimization ones. Also, it allows us to change the VIs' resulting algorithmic descriptions to achieve aesthetic or performance goals, as well as to store them in the framework, making them available for future projects.

As an example, to optimize our case study in terms of natural lighting and privacy levels, we need to iteratively analyze several design variations to find the solution that best suits both goals. In a typical design process using VIs, i.e., where the selected input is a shape manually created in the design tool, the analysis of different design configurations requires us to manually change the input shape before starting a new analysis cycle; a scenario that clearly hinders the automation of multiple analyses in an optimization process. Our framework overcomes this limitation, as the available algorithms handle all inputs in the same way, independently of being textual or visual inputs. In practice, while the former directly informs the optimization process, the latter is first converted into an algorithm to then inform the optimization. This means that we can use both VIs and TIs to drive optimization processes. Figure 4 shows an example where the optimization algorithm automatically changes both the position of a VI, i.e., a straight curve, and its intensity factor, producing a pool of results from which the architect can choose the one that most pleases him.

DISCUSSION

This section analyzes the previous approaches, making some final considerations on the benefits of integrating VIMs in a textual-based AD framework.

Portability

Concerning the portability of each approach, we conclude TIs are more advantageous because they (1) are independent from the modeling tool, (2) generate the exact same results in different tools, (3) store all the information needed to reproduce the results of their application at any time, and (4) can be easily converted into their corresponding visual representation. Contrarily, as a VI is typically created in a specific modelling tool, the AD programs using it become dependent on both the tool and the file containing the VI. Our framework overcomes these limitations by converting VIs into TIs, allowing the former to benefit from some of the latter's advantages in terms of portability. Also, when a VI is stable, i.e., when we no longer want to modify it, we can store its algorithmic description, becoming henceforth just part of a larger algorithm. As an example, consider figure 3I: in a first stage, we modelled different curves in the design tool, while applying the same transformation rule to the elements, i.e., a random size variation. When satisfied, we stored the final curve and, thereafter, we explored various transformations to the hexagonal elements without ever having to reselect the curve in the design tool (figure 5).

Ability to Generate Geometries

Regarding the ease that each approach has in generating different types of geometric information, we conclude that both have advantages over the other depending on the context. In a situation where model-user interaction is critical, i.e., where the architect has little or no AD experience, the use of VIs is essential, as it allows the user to freely create/manipulate the VI in the modeling tool and immediately see the resulting solution. VIs are also advantageous when we need to use pre-existing geometries or create organic/free-form shapes, whose corresponding mathematical descriptions are often difficult to derive. Contrarily, TIs are advantageous when we need to use known mathematical curves or surfaces as input or to provide the output of an AD program as input to another.

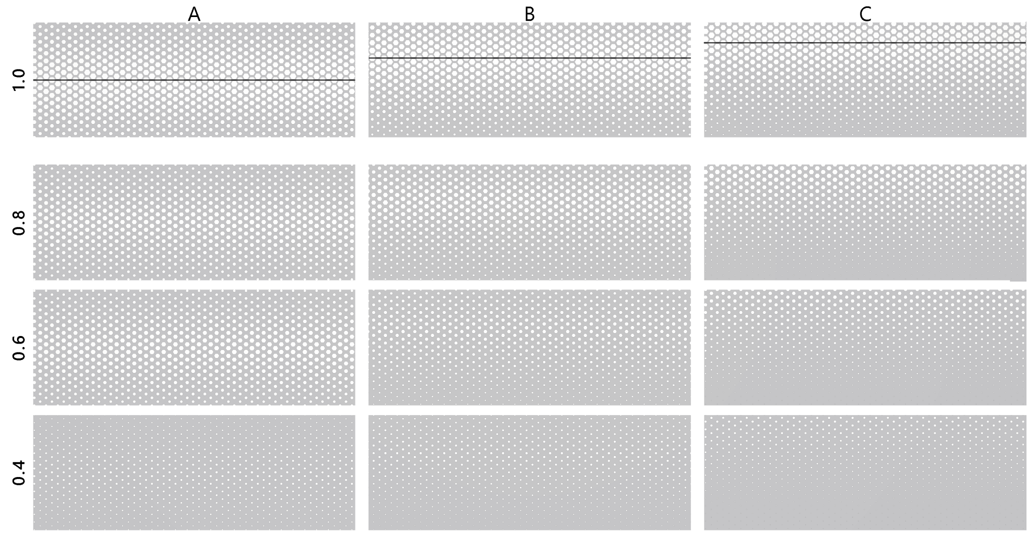

Figure 4
Case study optimization: analysis of solutions resulting from different intensity factors (0.4 to 1.0) and attractor-curve positions; (A) original position; (B-C) two of the positions tested.

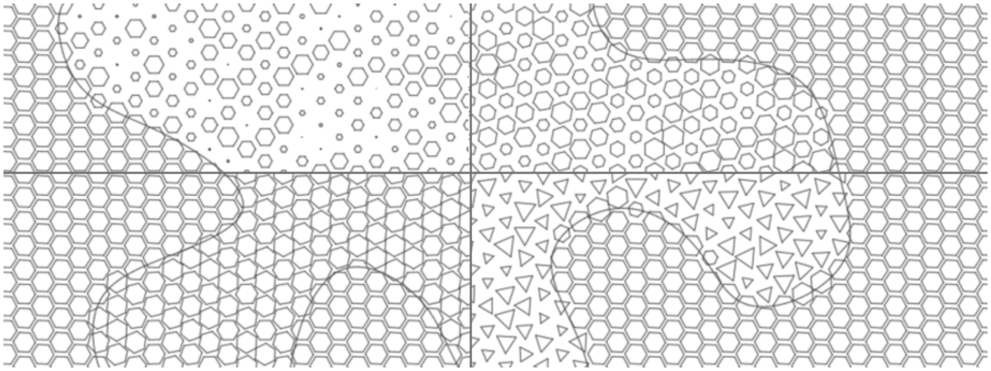

Figure 5
Different design variations using the same free-form curve of figure 3I.

These relative advantages are visible in the previous examples: while, in figure 2A, the mathematical sine curve resulting from a TI evidences a more perfect sinusoidal effect than the one manually created in figure 3A, the implementation of TIs representing the free-form curves in figures 3I and 3J would require us to algorithmically describe each point of the curve, which would be more laborious and time-consuming than modeling them directly in the modelling tool.

Flexibility

Based on the above considerations we conclude that TIs are usually more accurate and portable than VIs. Nevertheless, as our framework handles VIs and TIs similarly, both end up having almost the same flexibility. Still, while most TIs correspond to parametric algorithmic descriptions, the conversion of VIs into TIs results in non-parametric ones. To obtain parametric descriptions, we need to spend extra time and effort with their implementation, which also requires programming experience. Moreover, finding the parametric descriptions of VIs can be a difficult task, as it happens with the curves in figures 3C, 3I, and 3J.

As an example, consider again the designs in figure 2A and 3A. In the first case, the sine curve is described by a parametric function, allowing us to easily alter its parameters, namely amplitude, phase, and frequency, and obtain different curves as result. This does not happen with the second example, because it uses a manually produced curve and, although the framework can convert the curve into an algorithm, the latter is not parametric. Thus, to alter its amplitude, frequency, or phase, we need to either implement its parametric description or manually create a new curve in the modelling tool with the desired modifications.

The differences between VIs and TIs become relevant in analysis and optimization processes, where VIs evidence more limitations due to their lack of portability, limited flexibility, and CAD-dependency. TIs, due to their greater flexibility, end up giving more freedom and autonomy to the optimization process, widening the range of design variations explored and, therefore, increasing the likelihood of finding better-performing solutions.

As a final example, consider the designs in figures 2D and 3D. The first one uses the parametric description of the *witch of Agnesi* curve as TI, allowing us to easily obtain different curve instances by changing its parameters. The second one uses an approximation of the same curve modelled in the design tool as VI, which is then converted by the framework into its algorithmic but non-parametric description. In practice, when combining the TI with the optimization algorithm, we can make the latter control the former's parameters throughout the entire process, obtaining better-performing solutions. When combining the VI with the optimization algorithm, the latter can also modify the former by changing its algorithmic description, however, it can only change the spatial locations that define the curve individually because there is no parametric dependency between them.

CONCLUSION

The architectural practice is moving towards an increasing use of analysis and optimization processes. However, these processes are time-consuming and the various types of analysis require data transfers between multiple tools, causing information losses and the accumulation of design errors (Leitão et al. 2017). These shortcomings hinder the adoption of analysis and optimization processes in architecture.

We can overcome these limitations by combining analysis and optimization with Algorithmic Design (AD) (Alfaris and Merello 2008), which is a design approach based on the use of algorithms that demonstrates great potential for automating the *generation-analysis-regeneration* cycle typical of optimization processes. AD can resort to Textual Programming Languages (TPLs) or Visual Programming Languages (VPLs), the latter being the most preferred ones by architects due to their visual nature and ability to support Visual Input Mechanisms (VIMs), making them more intuitive and easier to learn and use. However, TPLs are more expressive, supporting the development of large-scale designs (Leitão et al. 2012; Celani and Vaz 2012; Wortmann and Tunçer 2017). Given their different advantages, it is beneficial to combine them in a hybrid approach.

AD is not new and was already applied to carry out complex projects that otherwise would be unfeasible. It has also proved its ability to reduce construction costs and waste, while improving the quality of the design solutions. We expect that, in the near future, AD will be integrated in all stages of the design process, and in most design studios and construction companies.

In this paper, we extended a TPL-based AD framework for facade design, analysis, and optimization to support VIMs. We explained the features that were integrated in the framework to handle different types of Visual Inputs (VIs), making them portable across different design tools, as well as suitable to be integrated in a text-based AD approach. We used some of them to describe different graphical inputs, such as points, curves, and areas, that constrained the facade design. We evaluated the framework in the development of a case study, for which we used both VIs and Textual Inputs (TIs), concluding that both have advantages depending on the design situation. Still, TIs evidenced significant advantages regarding accuracy, portability, and flexibility. Moreover, when combined with the framework's optimization algorithms, VIs proved to be more limited due to lack of flexibility and parametricity.

As future work, we plan to extend the framework with more VIMs, such as images, 3D scans, and point clouds, and improve the already existing ones, especially to allow the direct conversion of a VI into its parametric representation. Also, we plan to research the use of VIMs for other AD problems, particularly, those addressing 3D spatial configurations, such as paths for cameras in rendering tasks, locations to operate as generators of 3D volumetric forms or as distributors of building elements in space, or surfaces for terrain modeling tasks or for creating architectural elements such as roof space frames and shading structures.

ACKNOWLEDGEMENTS

This work was supported by national funds through *Fundação para a Ciência e a Tecnologia* (FCT) with references UIDB/50021/2020 and PTDC/ART-DAQ/31061/2017, and by the PhD grant under contract of FCT with reference SFRH/BD/128628/2017.

REFERENCES

Alfaris, A and Merello, R 2008 'The Generative Multi-Performance Design System', *ACADIA 08 › Silicon + Skin › Biological Processes and Computation*, pp. 448-457

Caetano, I, Ilunga, G, Belém, C, Aguiar, R, Feist, S, Bastos, F and Leitão, A 2018 'Case Studies on the Integration of Algorithmic Design Processes in Traditional Design Workflows', *Learning, Adapting and Prototyping, Proceedings of the 23rd International Conference of the Association for Computer-Aided Architectural Design Research in Asia*, pp. 129-138

Caetano, I and Leitão, A 2018 'Algorithmic Patterns for Facade Design: Merging design exploration, optimization and rationalization', *FACADE TECTONICS 2018 World Congress Conference Proceedings*, pp. 413-422

Celani, G and Vaz, C 2012, 'CAD Scripting And Visual Programming Languages For Implementing Computational Design Concepts: A Comparison From A Pedagogical Point Of View', *International Journal of Architectural Computing*, 10(01), pp. 121-138

Czarnecki, K, Østerbye, K and Völter, M 2002, 'Generative Programming', in Hernández, J and Moreira, A (eds) 2002, *Object-Oriented Technology ECOOP 2002 Workshop Reader*, Springer, Berlin, Heidelberg, pp. 15-29

Janssen, P 2014 'Visual Dataflow Modelling: Some Thoughts on Complexity', *Fusion - Proceedings of the 32nd eCAADe Conference - Volume 2*, Newcastle upon Tyne, England, UK, p. 305–314

Konis, K, Gamas, A and Kensek, K 2016, 'Passive performance and building form: An optimization framework for early-stage design support', *Solar Energy*, 125(February), pp. 161-179

Leitão, A, Castelo Branco, R and Cardoso, C 2017 'Algorithmic-based Analysis: Design and Analysis in a Multi Back-end Generative Tool', *Protocols, Flows and Glitches, Proceedings of the 22nd International Conference of the Association for Computer-Aided Architectural Design Research in Asia*, Suzhou, China, p. 137–146

Leitão, A, Castelo Branco, R and Santos, G 2019 'Game of Renders: The Use of Game Engines for Architectural Visualization', *Intelligent and Informed - Proceedings of the 24th International Conference on Computer-Aided Architectural Design Research in Asia*, Victoria University of Wellington, New Zealand, pp. 655-664

Leitão, A, Lopes, J and Santos, L 2014 'Illustrated Programming', *Acadia 2014: Design Agency*, Los Angeles, California, US, pp. 291-300

Leitão, A and Santos, L 2011 'Programming Languages for Generative Design: Visual or Textual?', *Respecting Fragile Places: Proceedings of the 29th eCAADe Conference*, Ljubljana, Slovenia, pp. 139-162

Leitão, A, Santos, L and Lopes, J 2012, 'Programming Languages For Generative Design: A Comparative Study', *International Journal of Architectural Computing*, 10(1), pp. 139-162

Lopes, J and Leitão, A 2011 'Portable Generative Design for CAD Applications', *Integration Through Computation - Proceedings of the 31st Annual Conference of the Association for Computer Aided Design in Architecture*, Calgary/Banff, Alberta, Canada, pp. 196-203

Machairas, V, Tsangrassoulis, A and Axarli, K 2014, 'Algorithms for optimization of building design: A review', *Renewable and Sustainable Energy Reviews*, 31(1364), pp. 101-112

Nguyen, AT, Reiter, S and Rigo, P 2014, 'A review on simulation-based optimization methods applied to building performance analysis', *Applied Energy*, 113, pp. 1043-1058

Noone, M and Mooney, A 2018, 'Visual and textual programming languages: a systematic review of the literature', *Journal of Computers in Education*, 5(2), pp. 149-174

Picco, M, Lollini, R and Marengo, M 2014, 'Towards energy performance evaluation in early stage building design: A simplification methodology for commercial building models', *Energy & Buildings*, 76, pp. 497-505

Sammer, M, Leitão, A and Caetano, I 2019 'From Visual Input to Visual Output in Textual Programming', *Intelligent & Informed, Proceedings of the 24th International Conference of the Association for Computer-Aided Architectural Design Research in Asia*, p. 645–654

Schittich, C (eds) 2006, *Building Skins*, Birkhäuser

Schulz, CN 1971, *Existence, space & architecture*, New York: Praeger. Stamps

El Sheikh, MM 2011, *Intelligent building skins: Parametric-based algorithm for kinetic facades design and daylighting performance integration*, Ph.D. Thesis, University of Soutern California

Shi, X 2010, 'Performance-based and performance-driven architectural design and optimization', *Frontiers of Architecture and Civil Engineering in China*, 4(4), pp. 512-518

Turrin, M, Von Buelow, P and Stouffs, R 2011, 'Design explorations of performance driven geometry in architectural design using parametric modeling and genetic algorithms', *Advanced Engineering Informatics*, 25(4), p. 656–675

Wortmann, T and Tunçer, B 2017, 'Differentiating parametric design: Digital workflows in contemporary architecture and construction', *Design Studies*, 53, pp. 173-197

Zboinska, MA 2015, 'Hybrid CAD/E platform supporting exploratory architectural design', *Computer-Aided Design*, 59, pp. 64-84