Sketching Algorithmic Design

Renata Castelo-Branco[1,2], Inês Caetano[1], Inês Pereira[1], and António Leitão[1]

[1]INESC-ID/Instituto Superior Técnico, University of Lisbon, Portugal

[2]renata.castelo.branco@tecnico.ulisboa.pt

Note: This document is a draft version of a manuscript published as part of the *Journal of Architectural Engineering*, © 2022 American Society of Civil Engineers, ISSN 1076-0431. **DOI:** 10.1061/(ASCE)AE.1943-5568.0000539

ABSTRACT

In the last decades, architecture has experienced paradigm shifts prompted by new computational tools. Algorithmic Design (AD), a design approach based on algorithms, is one such example. However, the architectural design practice is strongly based on visual and spatial reasoning, which is not easy to translate onto algorithmic descriptions. Consequently, even using tailored AD tools, AD programs are generally hard to understand and develop, independently of one's programming abilities. To address this problem, we propose a methodology and a design environment to support AD in a way that is more akin to the workflow typically employed by architects, who represent their ideas mostly through sketches and diagrams. The design environment is implemented as a computational notebook, with the ability to intertwine code, textual descriptions, and visual documentation in an integrated storytelling experience that helps architects read and write AD programs.

INTRODUCTION

Architectural drawings have for centuries been done by hand (Mitchell 2004). In the last few decades, however, representation methods have changed due to the invention of digital tools and, more recently, due to the emergence of Algorithmic Design (AD), a design approach based on

algorithms (Caetano et al. 2020). Using AD, architects design in a fundamentally different manner, exploring design ideas through algorithmic descriptions, i.e., algorithmic representations of design concepts, written as computer programs. AD not only reduces the modelling effort (Burry 2011) but, when coupled with analysis and optimization routines (Nguyen et al. 2014), also motivates the search for better-performing solutions.

Despite these advantages, AD has a major drawback: it imposes a translation process from the abstract imagery in the creator's mind into concrete, often textual, descriptions (Boshernitsan and Downes 2004). These descriptions are not as intuitive as other representation means, such as sketches (Victor 2012), thus hindering communication in collaborative projects, an ever more present reality. Naturally, architectural projects also rely on multiple file formats to share information, which can also work as comprehension support for the AD descriptions. However, this brings about another common issue that AD must overcome: compatibility (Kensek and Noble 2014). Furthermore, the simple exchange of data among team members, particularly when working with different digital tools, motivates inconsistencies and version conflicts.

This research tackles three main difficulties identified in the use of AD in the industry: (1) understanding algorithmic descriptions developed by others (Myers 1990; Davis et al. 2011); (2) addressing design tasks that require different design tools (O'Donnell et al. 2013; Eastman et al. 2008; Martinho et al. 2020; Lopes and Leitão 2011); and (3) ensuring a consistent view of the design among team members (Baker 2016; Sanchez 2016). To address these problems, we propose the AD Sketchbook, a methodology inspired by the idea of the digital visual narrative that promotes the intertwining of code with textual and visual documentation for a storytelling experience that helps architects read, write, and share AD programs.

METHODS

The present research aims at responding to the question: *how can we improve collaborative AD practices?* The proposed answer is a methodology to support readability, compatibility, and reproducibility in shared AD solutions. To that end, we adopt a design research focused on the study of design processes and practices (Cross 2006). Inspired by Research through Design

paradigms, which regard design methods and processes as generators of new knowledge (Stappers and Giaccardi 2017; Isley and Rider 2018), this investigation encompasses the following stages: (1) literature review, (2) methodology development, (3) project evaluation, and (4) data analysis and discussion.

The first stage identifies the existing cognitive barriers, the limitations of current solutions, and which strategies may lead to successful results. Based on the collected information, the second stage focuses on developing a set of ideas and design guidelines addressing the research question. More specifically, we develop a theoretical methodology for collaboratively developing AD programs and implement a design environment that supports it.

The evaluation stage entails the development of an architectural project using both the proposed methodology and design environment. Albeit subjective, the exploratory design research method used for the evaluation is adequate to gain a better understanding of the potential improvements to collaborative AD practices. The last stage encompasses the analysis of the results, aiming to answer our research question. Based on the previous findings, we identify the weaknesses and strengths of our proposal. Finally, we draw conclusions while outlining future research paths.

LITERATURE REVIEW

AD uses algorithms to develop architectural designs (Caetano et al. 2020). Since the design entities are logically connected, changes applied to the algorithm's parameters are automatically propagated through the entire program (Burry 2011), allowing effortless exploration of a variety of design ideas. Combined with architects' innate tendency to challenge the limits of creativity, AD has led to increasingly complex design solutions that simultaneously comply with the growing requisites of the building industry. This scenario has been motivating the need for collaborative design environments integrating different experts and tools (Laing 2019), where design representations need to be easily understood and handled by all participants.

Barriers

Design thinking and computational thinking are two very distinct processes. While the former is a nonlinear, messy, iterative process (Cross 2006), highly reliant on artistic sensibility and

intuitive playfulness (Terzidis 2006), that typically aims to find a context-specific solution (Kelly and Gero 2021), the latter forces designers to move from the iconic representation plane, where the referred playfulness typically takes place, onto the analogue and symbolic planes, where designs are abstractly represented by their properties and their potential, instead of their literal form (Mitchell 1975). In computational thinking, solutions usually have more general applications (Kelly and Gero 2021), which in the case of AD means designers can expand their ideation process from particular design solutions towards a design space containing potentially infinite solutions (Ameireh 2007).

In trying to merge the two worlds, AD programs tend to become unstructured products (Davis et al. 2011): the computational equivalent to the experimentation process that characterizes design thinking (Woodbury 2010). Therefore, in addition to the already challenging task of using programming languages to represent design concepts (Myers 1990), architects struggle to understand AD programs developed by others or by themselves too far in the past for the memory to reach. More specifically, in collaborative AD projects, where the same algorithmic descriptions are being edited by different parties, the propensity for mismatches is high, more so if the parties involved fail to understand each other's work or fail to reproduce it faithfully (Wang et al. 2019). Although some of these issues have already been addressed by programming environments, these are mostly tailored to general-purpose software development and, thus, practitioners find it hard to use them as design tools.

Programming Paradigms

Specialized design environments for AD have been developed in the past, with two main paradigms standing out: Visual Programming (VP) and Textual Programming (TP). VP describes programs by interconnecting elements that can be interactively manipulated (Myers 1990), whereas TP describes them as text. As such, while TP disregards our natural inclination towards imagery (Zhang 2007), VP explores it.

Nevertheless, the features that make VP so appealing also hinder its use at large scales (Leitão et al. 2012; Janssen 2014): graphic representations tend to rapidly overflow the bounds of the screen, with node relations obscuring the program's structure (Nardi 1993). In contrast, and despite

having a steeper learning curve, TP offers more scalability (Sammer et al. 2019; Burnett 1999), resolving information density problems with abstraction and filtering mechanisms. Unfortunately, independently of the programming paradigm, architects still struggle to understand AD programs, since translating designs to/from algorithmic representations is a considerable challenge even for the most gifted and creative ones (Boshernitsan and Downes 2004).

Program Comprehension

Literate Programming (Knuth 1984) proposes developing computer programs as literary works for humans to read, by documenting the implementation details along with the rationale behind them. Unfortunately, the idea is not directly applicable to architecture, because it is mostly based on textual explanations, while architectural design thinking requires different types of visualizations and graphic elements, such as sketches and drawings (Seitamaa-Hakkarainen and Hakkarainen 2000; Bresciani 2019).

Most architects sketch while designing and most do so as well when programming. In fact computer scientists alike often draw to help translate abstract concepts onto a more concrete representation such as a program (Stasko and Patterson 1992). In the architectural context, these sketches often represent the architect's intentions towards their AD program, explaining the logic behind its conception and what they expect it to produce. It is then essential that they fuse with the algorithmic description itself, taking part in the understanding process as well.

Some authors developed documentation techniques directly targeting textual AD. Illustrated programming (Leitão et al. 2014), for instance, proposed the inclusion of sketches, images, and renders in textual AD programs to establish correlations between these graphical elements, the program, and the generated model. However, these correlations had scalability problems, making their use unviable in large-scale projects. Furthermore, the proposal lacked other useful forms of program documentation, like formatted text and mathematical formulas. Some VP languages, such as Grasshopper, also allow for the inclusion of imagery and formulae in midst program.

Other features that greatly contribute to the comprehension of AD programs are traceability, i.e., the ability to relate program parts with the model parts they generate and vice versa, and

interactivity, i.e., a fast response of the system that allows users to rapidly visualize the impact of the changes made. One- or bi-directional traceability is supported by most VP languages used in architecture and by some TP languages as well (Leitão et al. 2014; Castelo-Branco et al. 2020b). Interactivity is more commonly found in the former, less so in latter (Alfaiate et al. 2017), but in either case scalability is an issue.

DESIGN SKETCHBOOK

In a creative process, such as architectural design, the narrative of how we got to the final design solution is just as important to comprehend it as the design itself. The sketchbook traditionally used by architects stands as proof to this. This creative journal gathers the history of the design's evolution based on drawings, schemes, and textual descriptions, helping architects remember, summarize, or even reuse design ideas. We propose carrying this concept over to AD, creating a design environment inspired by the architect's sketchbook.

The Computational Equivalent

Considering the above-mentioned difficulties in developing AD programs and the promising comprehension attributes of a design sketchbook, its computational equivalent has the potential to not only make AD more akin to the typical design process, but also facilitate later comprehension and reuse of algorithmic descriptions. This is particularly beneficial for collaborative work scenarios, where the person trying to decipher the AD project is rarely its creator. Finally, it should support exploration and explanation, two fundamental aspects of the development of any architectural project, and integrate the plethora of tools that typically take part in these projects as well.

For the computational sketchbook to function as an executable depiction of the design process, it must track the AD program's development history. Although there are already competent solutions addressing this problem (e.g., Githopper for Grashopper), implementing such an idea on top of VP languages is a troublesome task, since in the data flow paradigm it becomes harder to distinguish between functional and esthetical changes to the program. Considering that TP languages have for decades been the object of version control research and do not share VP's scalability issues, our preference lies with TP.

Computational Notebooks

Some of the proposed ideas, namely illustration, interactivity, and reproducibility, are at the core of the computational notebook paradigm (Rule et al. 2018). Embodied in tools such as Mathcad, Mathematica, or Jupyter, the concept was designed to support reproducible computational narratives, allowing for the incremental development of programs with immediate feedback on their results, as well as the intertwining of code with textual and visual documentation.

Although computational notebooks have been around for years, only more recently have they started to be widely used across multiple fields (Perez and Granger 2007; Randles et al. 2017). Notebooks allow users to simultaneously execute, document, and communicate their experiments (Rule et al. 2018), making their work reliable, reproducible, and comprehensible to others, and even to themselves at a later date. Due to these features, the same notebook can serve multiple purposes, such as tutorials, interactive manuals, presentations, or even scientific publications (Perkel 2018).

Given the above-mentioned advantages, we propose an adaptation of the computational notebook workflow to the architectural practice, providing a richer, more interactive, programming experience for architects, and an easier way for them to understand and reproduce each other's work in collaborative environments (Baker and Penny 2016; Wang et al. 2019).

ALGORITHMIC DESIGN SKETCHBOOK

To address program comprehension and maintenance in collaborative AD, we propose the AD Sketchbook, a methodology and a design environment that allow for the creation of algorithmic descriptions in an incremental and documented way. Our approach targets architects with programming experience who, nevertheless, feel the need for more adequate design mechanisms in collaborative AD approaches. The integration of the AD Sketchbook with the typical architectural design workflow involves not only incorporating the most used graphic-based representation methods, but also communicating and sharing information across multiple digital design tools.

Algorithmic Design Methodology

We previously identified three main issues with the use of AD in the industry: (1) understanding algorithmic descriptions; (2) addressing design tasks that require different design tools; and (3)

avoiding the inconsistencies that occur in shared AD projects. Directly responding to these problems, we propose a methodology divided in the three main principles illustrated in Figure 1: (1) storytelling to aid the comprehension of AD program; (2) compatibility among the digital design tools used in an AD project; and (3) reproducibility to support collaborative AD projects.

Storytelling implies preserving the history of design development, keeping the tale of the creative process available for future reference. To promote it, the AD Sketchbook allows architects to store the artifacts produced along the way in an organized fashion, entailing three concepts:

- *Incremental development* - to support the development of AD solutions piecemeal by defining small fragments of code and testing them right after, visualizing the result in the AD Sketchbook itself. The aim is to make the programming activity a more responsive and comprehensible endeavor, and help track the changes made from one iteration to another.
- *Interactivity* - to test program fragments in a reactive manner, with interactive mechanisms that facilitate design exploration, e.g., using sliders and toggles to more intuitively manipulate the design's parameters and, thus, better understand their impact. The aim is to bestow upon TP some of the features that make VP easier to comprehend.
- *Documentation* - to effectively tell the story of the design development, while guaranteeing the program is yielding the expected results. The aim is to turn the program into a narrative that others can follow and reproduce; thus, both textual and visual documentation is supported. The former includes textual descriptions explaining both the program and the images, and mathematical formulas illustrating the program's computations. The latter includes handmade sketches, rendered images, and other artifacts resulting from incremental development and interactivity.

The second principle is **compatibility**, which allows architects to use a single AD representation to interoperate with Computer-Aided Drafting (CAD) tools, Building Information Modelling (BIM) tools, and analysis tools, among others. The intention is to merge the AD methodology with the architect's typical workflow by encompassing its multiple design stages, namely exploration,

analysis, and optimization, and design outputs, such as the production of renders for presentation and technical documentation for fabrication.

Finally, the AD Sketchbook methodology encourages **reproducibility**, which guarantees the results are replicable at any time and by any of the parties involved in the project. To this end, the AD Sketchbook makes all software dependencies explicit, including datasets and software versions used. This ensures that several developers can work simultaneously in the same AD program, overcoming many of the difficulties architects face when sharing code.

Implementation

Despite their novelty in architecture, some ideas behind the proposed AD Sketchbook are already addressed in the computational notebooks described above. Particularly, the fact that they motivate users to write interactive computational narratives instead of programs. Jupyter is an example of a browser-based and open-source notebook (Perkel 2018), whose interface and exploratory style were originally inspired on Wolfram's, and which currently benefits from the support of a vast community of users and developers.

Given the aforementioned characteristics, we decided to adapt the computational notebook concept to fit AD processes by implementing the AD Sketchbook methodology on top of the Jupyter notebook editor. To make the resulting AD Sketchbook compatible with the design tools typically used in the field, we coupled Jupyter to Khepri (Sammer et al. 2019), an AD tool capable of communicating with several CAD, BIM, game engine, rendering, analysis, and optimization tools (Castelo-Branco and Leitão 2017; Martinho et al. 2020). Khepri supports the development of AD programs that generate equivalent models in these tools, while automating their analysis and optimization regarding structural, thermal, and lighting performance. This equivalence is ensured by Khepri's frontend/backend software architecture, where the frontend provides abstract modeling operations that have different implementations depending on the backend tool used. As an example, the abstract operation *wall* generates, in a CAD backend, just the geometry of the wall; in a BIM backend, the geometry plus the necessary construction detail encoded in a wall family; in a game engine, the geometry plus rendering textures; and in a lighting analysis tool, just a set of surfaces

with the corresponding lighting characteristics.

The AD Sketchbook communicates with Khepri through a bi-directional channel: evaluating program fragments in the AD Sketchbook prompts Khepri to create the intended digital models in the selected design or analysis tool and return the evaluation results to the AD Sketchbook in the latter case. However, to improve the storytelling experience and make the AD Sketchbook independent of the named tools, we embedded additional web-based visualizers that allow users to see and keep different kinds of graphical results next to the program fragments that generate them.

EVALUATION

In this section we evaluate the AD Sketchbook methodology in a collaboratively developed architectural project. Three of the architects among the authors of this article were involved in the project, working remotely on the same AD Sketchbook. The project comprised an office building façade in Lisbon, Portugal, whose design was inspired by tiling techniques. Figure 2 shows some conceptual drawings of the project integrated in the AD Sketchbook. The complete project can be found at `https://github.com/KhepriNotebook/FacadeTiling`.

Within the AD Sketchbook, the team coordinated several design tasks: the algorithmic development of both the façade design and the building's pre-existing geometry and surroundings, the analysis/optimization of the façade design in terms of indoor lighting performance, and the generation of fabrication schemes. Since the methods chosen to evaluate the proposed solution rely on a hands-on approach to test the AD Sketchbook methodology and implementation, the ensuing sections describe and illustrate the development of the project from a first-person point of view. The authors summarized their observations from the experience in three parts, corresponding to each of the AD Sketchbook's features.

Storytelling

The proposed methodology defends the preservation of the history of the algorithmic development. *Incremental development* allowed the architects to develop the AD program by performing small changes or additions that could be immediately tested, producing a new graphical representation each time. Each development step and corresponding tests and results were kept available

for future consultation. Figure 3 shows two tests performed during the façade pattern development that were recorded in the AD Sketchbook, together with the explanation and documentation of the design decisions made. In this case, the applied design changes resulted from the sensitivity analysis study of the daylight performance conducted midway, which showed that some interior spaces were below the standard metrics.

Interactivity motivates architects to explore their designs using sliders, toggles, and other widgets, to visually manipulate parameters. In the developed project, this feature was important to quickly explore design variants for the façade (Figure 4), assess the differences between the obtained solutions, and understand the impact each parameter had on the design.

The AD Sketchbook provides both textual and visual *documentation* mechanisms to explain its content and tell the creative story of the project. In this project, the three architects took advantage of both types of documentation to improve the collaborative design process. On the one hand, textual documentation was particularly important to structure the project's development history according to the different design stages and explain the design decisions (Figures 2 and 3), as was the use of mathematical formulas to illustrate the algorithms in a more comprehensible notation (Figure 3). On the other hand, visual documentation was useful to ensure the algorithm was producing the expected results: as we often introduce bugs in the program without noticing, having an image of the intended result available for comparison was crucial in debugging (Figures 3 and 4).

Compatibility

The use of multiple tools in design projects is a common practice motivated by the tools' different advantages (Castelo-Branco and Leitão 2017; Martinho et al. 2020). The AD Sketchbook allows designers to specify, in the algorithmic description itself, the tools they want to use at each stage of the project.

In the developed project, compatibility allowed the architects involved to centralize all the information and coordinate the entire design process from within the AD Sketchbook. In practice, they took advantage of the integrated visualizers during the design exploration stage due to their real-time feedback and more direct correlation between the AD program and its result; for more

complex geometric modelling tasks they switched to a CAD tool (Rhinoceros 3D); and to integrate construction details and produce documentation for the building structure, they transited to a BIM tool (Revit). To produce the technical documentation for the fabrication of the façade tiles, they once more resorted to a CAD tool (AutoCAD) as the laser-cutting process required *.dwg* files. They also used a game engine (Unity) to rapidly visualize and navigate through the complete model and a rendering tool (POV-Ray) to produce realistic films and other presentation imagery (Figure 5). Finally, they used an analysis tool (Radiance) and Khepri's optimization module to evaluate and optimize the daylight performance of the solution, without ever leaving the AD Sketchbook's environment (Figure 6). In all these cases, the exact same AD description of the design was used.

Note that the AD Sketchbook's features frequently intertwine with one another. For instance, in the context of performance optimization, the parametric nature of AD allows the architects to write scripts that automate the time-consuming iterative evaluation of design variations (Aguiar et al. 2017). However, since optimization results are frequently hard to visualize and interpret, the AD Sketchbook also provides interactive plots to help bridge part of this comprehension gap via storytelling. In this project, the architects benefited from such mechanisms to visualize the results of the performed daylight optimization, creating a Parallel Coordinates plot (Figure 6 bottom right) and an interactive Pareto front (Khazaii 2016): by clicking on a point in the Pareto front graph (Figure 6 top right), they could visualize the corresponding 3D model (Figure 6 bottom left). This facilitated the understanding of the trade-offs between the esthetical quality and lighting performance of the project.

Reproducibility

The workflow motivates listing all the necessary software packages and their respective versions in the AD Sketchbook itself, which contributes to the reproducibility of shared AD programs. Moreover, as it is compatible with several design tools, different co-workers are free to choose their preferred tool to generate their models. This notion is further stretched by the AD Sketchbook's embedded visualizers, which allow users to visualize the results in the same environment where the algorithm is being developed.

In the façade project, the constant testing of the developed AD solutions, promoted by the interactive development process, and the documentation of the obtained results proved to increase its reproducibility, enabling all team members to easily compare results. Figure 7 presents part of the evolution of the façade design: successive versions of the algorithm were stored in the AD Sketchbook, accompanied by explanatory images, texts, and tests, which were critical to document the creative process and the changes each co-worker made to it. Furthermore, as this example relies on the AD Sketchbook's specific visualizers, the results could be easily reproduced on the co-workers' machines.

DISCUSSION

In this section we discuss the benefits and limitations of the AD Sketchbook. Naturally, architectural projects and design workflows can vary significantly (Rittel and Webber 1973) since they depend on ever-changing variables (Isley and Rider 2018), e.g., the architect's interpretation of the problem and both its temporality and site-dependency. Thus, we do not expect our proposal to generically apply to all circumstances and design scenarios. Instead, the AD Sketchbook features must be molded to each design brief and team, responding to their own issues and synergies.

Moreover, the presented evaluation was conducted through exploratory research, having three of the authors of the article been the test subjects as well. Both the previously described experience and the ensuing discussion correspond to their agreed opinions on the impact each feature had on the project's development, as well as their benefits and shortcomings. While a joint opinion may obscure individual impressions, we believe these to be of lesser importance in a methodology that must, in any case, abide by specific design circumstances and team workflows.

A Tale of Design Development

Storytelling has shown to promote program comprehension and increase reproducibility in this shared programming experience. Having the design history in the AD Sketchbook can help architects (1) comprehend how and why certain design decisions were made, (2) reproduce each step taken, and (3) recall and revise their own past decisions. In the developed project, storytelling was critical for those involved to understand each other's work and to locate themselves in the

design process: as design changes were documented in the AD Sketchbook, each architect could easily be put up to date on the project status and, therefore, proceed with the design process in a coherent and coordinated way. Nevertheless, there is one setback to storytelling: the verbosity resulting from incremental development due to the accumulation of (1) tests that may no longer be necessary; (2) repeated code pieces resulting from multiple iterations over the same design detail; and (3) scattered code pieces that impair the organization of the sketchbook.

In this project, the architects were able to reduce some of this verbosity by using mechanisms available in the Jupyter environment to join, hide, and summarize dispersed code (Castelo-Branco and Leitão 2021). This process may partially relinquish the design's history, but it is a necessary trade-off, since the order in which one creates a story may not necessarily correspond to the order in which one wishes to tell it. Many users may, in fact, prefer to work with a summarized version of the sketchbook throughout. In such cases, instead of embedding the narrative in the program itself, the storytelling process can be managed by a version control system. More on this topic is explored in Castelo-Branco et al. (2020a).

More Is More

To merge with the architect's typical workflow, the AD Sketchbook is compatible with different tools. To keep the AD Sketchbook's consistency, the interactivity and documentation features have the same behavior with all supported tools. The difference between tools therefore lies in the advantages they offer to the design process, as well as on how they support incremental development.

As an example, the AD Sketchbook's integrated visualizers allow the graphics to coexist with the AD program: the resulting images appear right next to the program fragment that originated it (see Figure 7). However, there is a trade-off: these visualizers do not have the capabilities of external visualizers, such as CAD tools or game engines, and thus are only suitable for conceptual exploration processes. In this project, the architects strived for a balance between the two extremes, utilizing both integrated and external visualization tools throughout the entire process depending on the design problems faced at each stage and each collaborator's preferences.

Remotely and Collaboratively

The third goal of the AD Sketchbook is reproducibility; a feature that is heavily dependent on the two previous ones. On the one hand, storytelling can help architects comprehend each other's work and guarantee that the AD program yields the expected results. On the other hand, compatibility allows architects to choose the tools that best fit their design requirements. Both features contribute to reproducibility, helping anyone obtain identical results anywhere and at any time. This is particularly important for collaborative and remote work, two increasingly common scenarios nowadays due to the growing complexity of architectural projects and the general tendency towards global networks.

This was also the case in the presented façade design, where reproducibility allowed participants to individually develop and test parts of the project and, afterwards, share them with the other members of the design team, who could then understand, reproduce, and further develop them. The collaboration made possible by reproducibility not only accelerated the design process, as multiple architects were simultaneously contributing to it, but also improved the quality of the final solution, as different minds have different perspectives on the same design problem. Notwithstanding, the use of the AD Sketchbook in a collaborative work scenario must be coordinated among practitioners to avoid conflicting versions or repeated work.

CONCLUSION

Architecture is experiencing paradigm shifts prompted by new computational tools, with new design approaches emerging as a result, namely Algorithmic Design (AD). Nevertheless, architectural practice is strongly based on visual and spatial reasoning, whereas AD is based on algorithms, requiring architects to overcome the non-trivial task of translating their ideas onto algorithmic descriptions. As such, even using tailored AD tools, AD programs are generally hard to understand and develop. Three main problems were identified, namely (1) understanding programs developed by others or ourselves sometime in the past, (2) using different design tools as required by different design tasks or preferences; and (3) reproducing the resulting designs in different circumstances.

We addressed these problems by proposing the AD Sketchbook, a computational analogy

to the architect's creative sketchbook, which embodies a design medium where architects can incrementally and interactively develop and document their AD descriptions in a more responsive and comprehensible programming activity. Our proposal includes a methodology and a design environment implemented on top of the Jupyter computational notebook and the Khepri AD tool.

To evaluate the proposal, three of the authors collaboratively and remotely developed an experimental architectural project using the AD Sketchbook. Their experience was documented in this article. The proposed methodology allowed each of them to (1) independently develop parts of the project in the same environment, resulting in a storytelling process that facilitated the comprehension of design decisions; (2) use different design tools according to the design task; and (3) remotely execute the program parts developed by others and obtain the same results, which was critical for keeping the coherency of the design process.

Given that our proposal primarily targets architecture professionals that intend to use AD, it must contemplate the variety of existing design workflows and fuse with them in a suiting manner for each professional. There will doubtfully ever be a silver bullet befitting all possible design approaches. Nevertheless, we believe the AD Sketchbook can promote the adoption of AD approaches, whether individually or collaboratively, improving the otherwise difficult process of developing AD solutions. Furthermore, although the proposal focused on architectural design, the solution can be easily adapted to other design areas.

As future work, we plan on exploring different interaction mechanisms for different types of algorithmic parameters. Additionally, we will augment the current one-way interaction with the CAD and BIM paradigms by integrating visual inputs mechanisms (Sammer et al. 2019), allowing direct manipulation of geometry used as input to the AD program. Finally, we plan on conducting user studies on the use of the AD Sketchbook methodology, comparing its performance in terms of comprehension, compatibility, and reproducibility, with traditional means of employing AD in collaborative architectural projects.

Data Availability Statement

Some or all data, models, or code generated or used during the study are available in a repository online in accordance with funder data retention policies (`https://github.com/KhepriNotebook/FacadeTiling`)

Acknowledgments

This work was supported by national funds through *Fundação para a Ciência e a Tecnologia* (FCT) (references UIDB/50021/2020 and PTDC/ART-DAQ/31061/2017) and PhD grants under contract of FCT (grant numbers SFRH/BD/128628/2017, DFA/BD/4682/2020, and DFA/BD/06302/2021).

REFERENCES

Aguiar, R., Cardoso, C., and Leitão, A. (2017). "Algorithmic design and analysis fusing disciplines." *Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, Cambridge, Massachusetts, USA, 28–37.

Alfaiate, P., Caetano, I., and Leitão, A. (2017). "Luna Moth: Supporting creativity in the cloud." *Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, Cambridge, Massachusetts, USA, 72–81.

Ameireh, O. M. (2007). "Abstract thinking: An introduction to creative thinking in basic design." *International Conference of the Arab Society for Computer Aided Architectural Design (ASCAAD)*, Alexandria, Egypt, 527–542.

Baker, M. (2016). "1,500 scientists lift the lid on reproducibility." *Nature*, 533, 452–454.

Baker, M. and Penny, D. (2016). "Is there a reproducibility crisis?." *Nature*, 533(7604), 452–454.

Boshernitsan, M. and Downes, M. S. (2004). "Visual programming languages: A survey." *Report no.*, University of California, Berkeley, USA.

Bresciani, S. (2019). "Visual design thinking: A collaborative dimensions framework to profile visualisations." *Design Studies*, 63, 92–124.

Burnett, M. M. (1999). "Visual programming." *Wiley Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, ed., John Wiley & Sons, Inc., 275–283.

Burry, M. (2011). *Scripting Cultures: Architectural Design and Programming*. John Wiley & Sons, West Sussex, UK.

Caetano, I., Santos, L., and Leitão, A. (2020). "Computational design in architecture: Defining parametric, generative, and algorithmic design." *Frontiers of Architectural Research*, 9(2), 287–300.

Castelo-Branco, R., Caetano, I., Pereira, I., and Leitão, A. (2020a). "The collaborative algorithmic design notebook." *International Conference of the Architectural Science Association (ANZAScA)*, Auckland, New Zealand, 1056–1065.

Castelo-Branco, R. and Leitão, A. (2017). "Integrated algorithmic design: A single-script approach for multiple design tasks." *Education and research in Computer Aided Architectural Design in Europe (eCAADe) Conference*, Vol. 1, Rome, Italy, 729–738.

Castelo-Branco, R. and Leitão, A. (2021). "Comprehending algorithmic design." *Design Imperatives: Proceedings of the Computer-Aided Architectural Design Futures (CAAD Futures) Conference*, Springer, Berlin, Heidelberg, University of Southern California, USA. (to appear).

Castelo-Branco, R., Leitão, A., and Brás, C. (2020b). "Program comprehension for live algorithmic design in virtual reality." *International Conference on the Art, Science, and Engineering of Programming (<Programming'20> Companion)*, Porto, Portugal, ACM, New York, NY, USA, 69–76.

Cross, N. (2006). *Designerly Ways of Knowing*. Springer-Verlag, London, UK.

Davis, D., Burry, J., and Burry, M. (2011). "Understanding visual scripts: Improving collaboration through modular programming." *International Journal of Architectural Computing*, 9(4), 361–376.

Eastman, C., Teicholz, P., Sacks, R., and Liston, K. (2008). *BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility Managers*. Wiley, Hoboken, New Jersey.

Isley, C. G. and Rider, T. (2018). "Research-through-design: Exploring a design-based research paradigm through its ontology, epistemology, and methodology." *Design Research Society (DRS)*

International Conference, Limerick, Ireland, 25–28.

Janssen, P. (2014). "Visual dataflow modelling - some thoughts on complexity." *Education and research in Computer Aided Architectural Design in Europe (eCAADe) Conference*, Vol. 2, Newcastle upon Tyne, England, UK, 547–556.

Kelly, N. and Gero, J. S. (2021). "Design thinking and computational thinking: A dual process model for addressing design problems." *Design Science*, (May), 1–15.

Kensek, K. and Noble, D. (2014). *Building Information Modeling: BIM in Current and Future Practice*. Wiley, Hoboken, New Jersey.

Khazaii, J. (2016). *Advanced Decision Making for HVAC Engineers: Creating Energy Efficient Smart Buildings*. Springer, Switzerland.

Knuth, D. (1984). "Literate programming." *The Computer Journal*, 27(2), 97–111.

Laing, R. (2019). *Digital Participation and Collaboration in Architectural Design*. Taylor & Francis (Routledge), Abingdon.

Leitão, A., Lopes, J., and Santos, L. (2012). "Programming languages for generative design: A comparative study." *International Journal of Architectural Computing*, 10(1), 139–162.

Leitão, A., Lopes, J., and Santos, L. (2014). "Illustrated programming." *Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, Los Angeles, California, USA, 291–300.

Lopes, J. and Leitão, A. (2011). "Portable generative design for CAD applications." *Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, Banff, Canada, 196–203.

Martinho, H., Pereira, I., Feist, S., and Leitão, A. (2020). "Integrated algorithmic design in practice: A renovation case study." *Education and research in Computer Aided Architectural Design in Europe (eCAADe) Conferencee*, Vol. 1, Berlin, Germany, 429–438.

Mitchell, W. J. (1975). "The theoretical foundation of computer-aided architectural design." *Environment and Planning B*, 2(2), 127–150.

Mitchell, W. J. (2004). "Foreword." *Architecture's New Media: Principles, Theories, and Methods*

505 *of Computer-Aided Design*, Y. Kalay, ed., MIT Press, ix–xii.

506 Myers, B. (1990). "Taxonomies of visual programming and program visualization." *Journal of*
507 *Visual Languages & Computing*, 1(1), 97–123.

508 Nardi, B. A. (1993). *A Small Matter of Programming: Perspectives on End User Computing*. MIT
509 Press, Cambridge, MA, USA.

510 Nguyen, A.-T., Reiter, S., and Rigo, P. (2014). "A review on simulation-based optimization methods
511 applied to building performance analysis." *Applied Energy*, 113, 1043–1058.

512 O'Donnell, J., Maile, T., Rose, C., Mrazovic, N., Morrissey, E., Regnier, C., Parrish, K., and
513 Bazjanac, V. (2013). "Transforming BIM to BEM: Generation of building geometry for the
514 NASA Ames sustainability base BIM." *Report no.*, Lawrence Berkley National Laboratory,
515 Berkeley, USA, <https://eta.lbl.gov/publications/transforming-bim-bem-generation>.

516 Perez, F. and Granger, B. E. (2007). "IPython: A system for interactive scientific computing."
517 *Computing in Science Engineering*, 9(3), 21–29.

518 Perkel, J. M. (2018). "Why jupyter is data scientists' computational notebook of choice." *Nature*,
519 563(7729), 145–146.

520 Randles, B. M., Pasquetto, I. V., Golshan, M. S., and Borgman, C. L. (2017). "Using the jupyter
521 notebook as a tool for open science: An empirical study." *ACM/IEEE Joint Conference on Digital*
522 *Libraries (JCDL)*, Toronto, Ontario, Canada, 1–2.

523 Rittel, H. W. J. and Webber, M. M. (1973). "Dilemmas in a general theory of planning." *Policy*
524 *Sciences*, 4, 155–169.

525 Rule, A., Tabard, A., and Hollan, J. D. (2018). "Exploration and explanation in computational note-
526 books." *CHI Conference on Human Factors in Computing Systems*, Vol. 2018-April, Montreal,
527 Canada, Association for Computing Machinery, 1–12.

528 Sammer, M. J., Leitão, A., and Caetano, I. (2019). "From visual input to visual output in textual
529 programming." *International Conference of the Association for Computer-Aided Architectural*
530 *Design Research in Asia (CAADRIA)*, Vol. 1, Wellington, New Zealand, 645–654.

531 Sanchez, J. (2016). "Massive re-patterning of the urban landscape." *Architectural Design*, 86(5),

48–51.

Seitamaa-Hakkarainen, P. and Hakkarainen, K. (2000). "Visualization and sketching in the design process." *The Design Journal: An International Journal for All Aspects of Design*, 3(1), 3–14.

Stappers, P. J. and Giaccardi, E. (2017). "Research through design." *The Encyclopedia of Human-Computer Interaction*, C. Ghaoui, ed., IGI Publishing, Hershey, PA, Chapter 43.

Stasko, J. T. and Patterson, C. (1992). "Understanding and characterizing software visualization systems." *IEEE Workshop on Visual Languages*, IEEE, 3–10.

Terzidis, K. (2006). *Algorithmic Architecture*. Architectural Press, New York.

Victor, B. (2012). "Stop drawing dead fish, <https://vimeo.com/64895205>. Accessed 1 May 2021.

Wang, A. Y., Mittal, A., Brooks, C., and Oney, S. (2019). "How data scientists use computational notebooks for real-time collaboration." *Proceedings of the ACM on Human-Computer Interaction*, 3.

Woodbury, R. (2010). *Elements of parametric design*. Routledge, Oxon and New York.

Zhang, K. (2007). *Visual languages and applications*. Springer-Verlag US, New York.

List of Figures

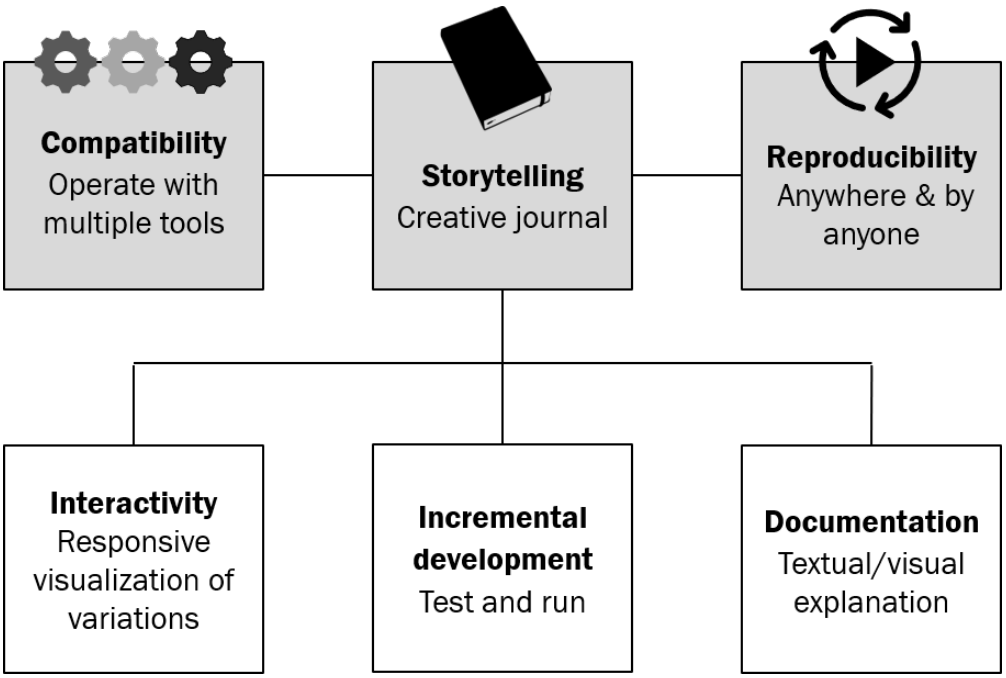


Fig. 1. Design environment's supported features.

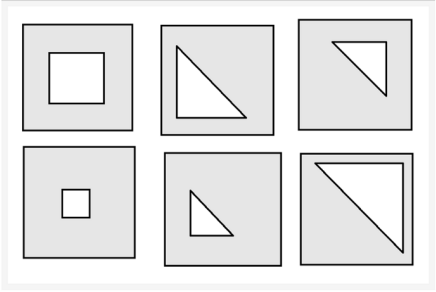
4 Facade Concept

Tiling

A tiling of the plane is a family of sets - called tiles - that cover the plane without gaps or overlaps. Tiling is also known as tessellation, paving, or mosaics; they have appeared in human activities since prehistoric time, including architecture. This technique has found several applications in architecture, namely, to explore both its tectonic and visual expressiveness to generate building skins with multiple geometric patterns and performance behaviors (e.g., shading, ventilation, lighting proprieties, among others).

In this particular design, we use tiling to create a visually interesting/dynamic geometric pattern, as well as to shade the interior spaces of the building: we opted for using irregular triangular tiles, which we materialize as triangular-shaped elements that can have different sizes and opacity levels.

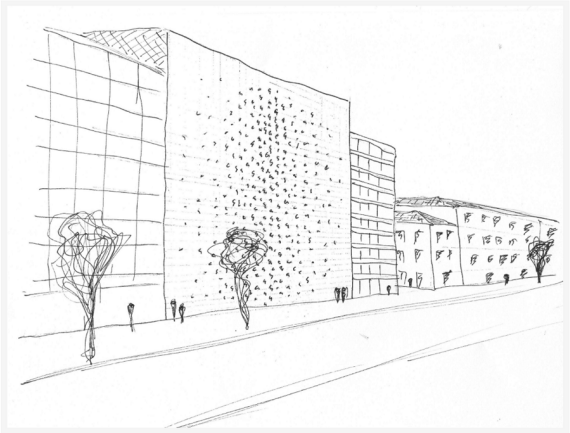
Tile aperture shapes and sizes



On site

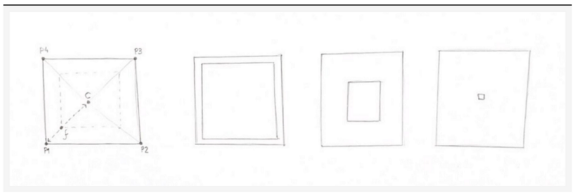
The application of the pattern to the building's façade intends to both create a visually interesting/dynamic geometric effect, and to shade the interior spaces of the building from the intense natural daylight typical of southern European countries. The pattern creates an interesting light effect on the inside office spaces.

Hand-made drawing of the initial design intent for the location

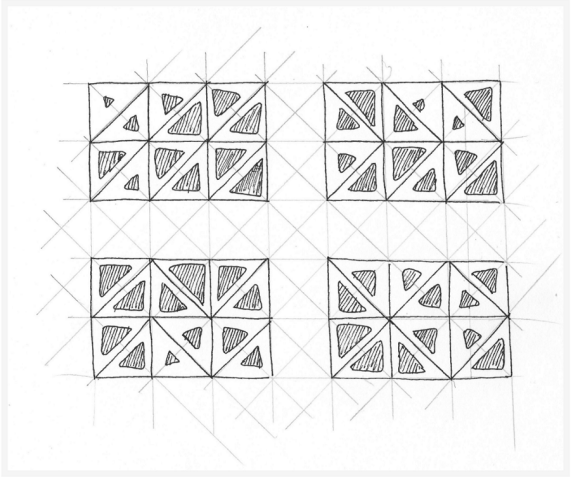


5.3 Pattern

Tile pattern conceptual sketch: left - design variables, right - design variations



Triangular tile concept with varying opacity levels



Tiling facade with triangular tiles of varying opacity levels

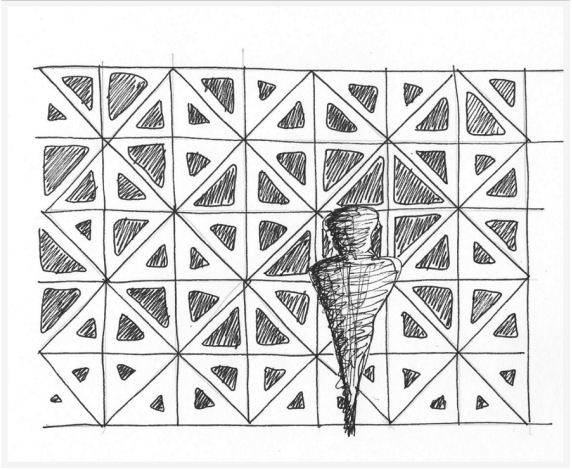


Fig. 2. Design concept documentation in the AD Sketchbook.

Multiple stains pattern distribution

Second pattern distribution choice

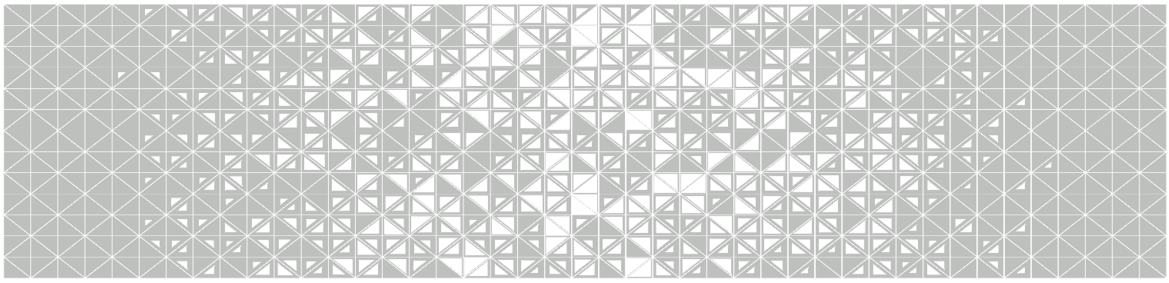
After some preliminary analysis we concluded the light coming into the service spaces and corner offices was not enough for our standard metrics and we changed the distribution to accommodate the same design effect but several times along the façade's width. Specifically, we indented the sinusoidal wavelength to match the number of large office division inside, so that each big room could appreciate the fading effect from the inside in its entirety.

old mathematical expression for aperture pattern:

$$\mathrm{F}_{aperture}(p) \in \left[F_{min}, \max\left(F_{min}, \sin\left(\frac{p_x}{length} \times \pi\right) - (1 - F_{max})\right)\right]$$

```
@test begin
    test_facade_tiles((fmin,fmax,length)->
        pts->random_range(fmin, max(fmin, sin(pts[1][1].x/length*π)-(1-fmax))))
end
```

Expected result:



new mathematical expression for aperture pattern:

$$\mathrm{F}_{aperture}(p) \in \left[F_{min}, \max\left(F_{min}, \left|\sin\left(\frac{p_x}{length} \times n_{rooms} \times \pi\right)\right| - (1 - F_{max})\right)\right]$$

```
@test begin
    test_facade_tiles((fmin,fmax,length)->
        pts->random_range(fmin, max(fmin, (abs ∘ sin)(pts[1][1].x/length*n_rooms*π)-(1-fmax))))
end
```

Expected result:

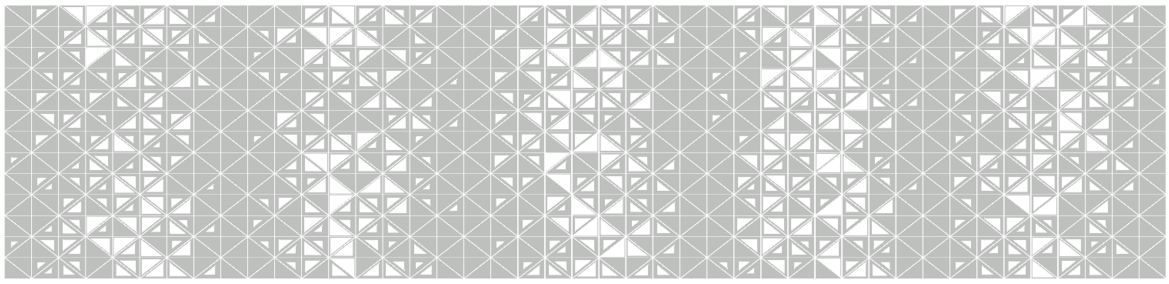


Fig. 3. Façade design options: central stain effect (top) and vertical stains effect (bottom).

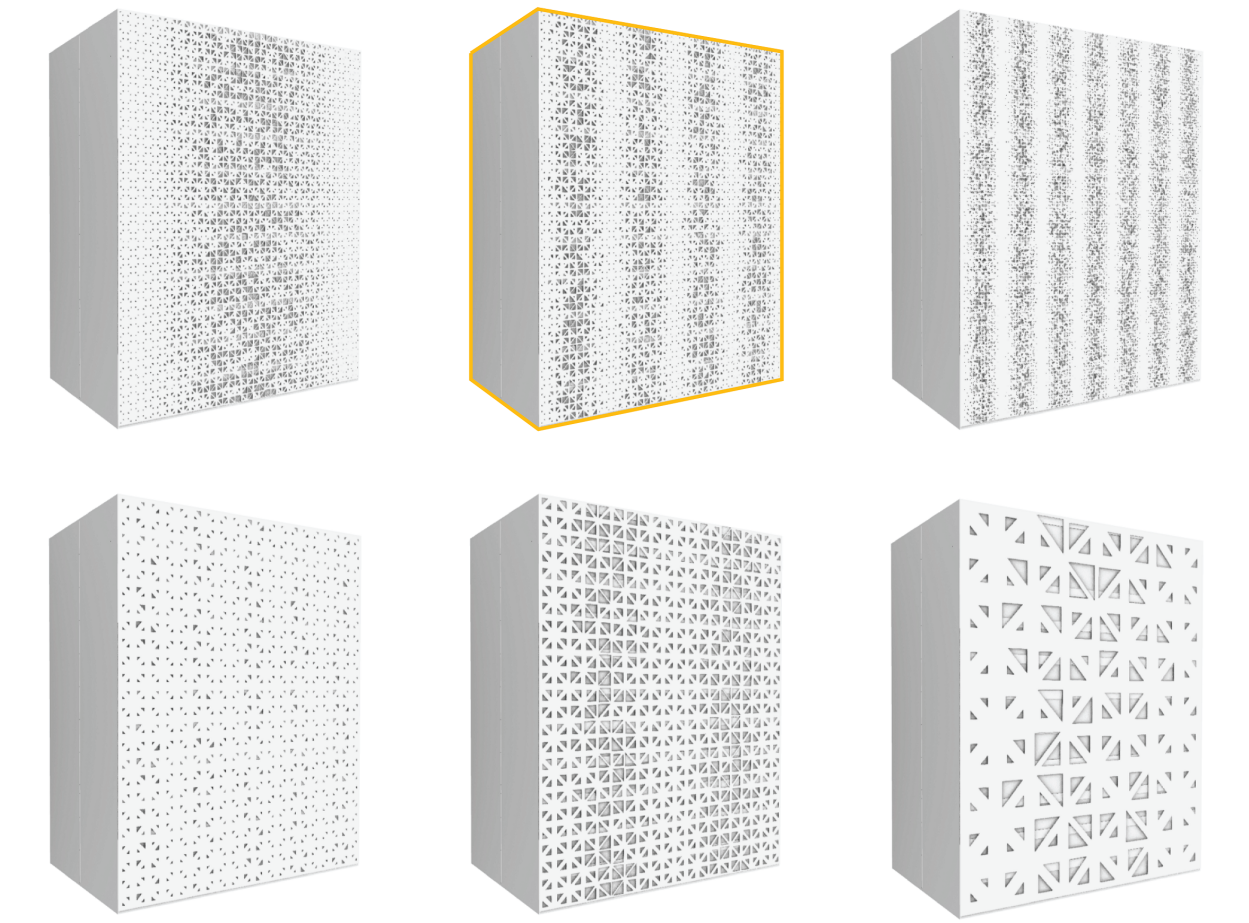


Fig. 4. Parametric manipulation of façade parameters.

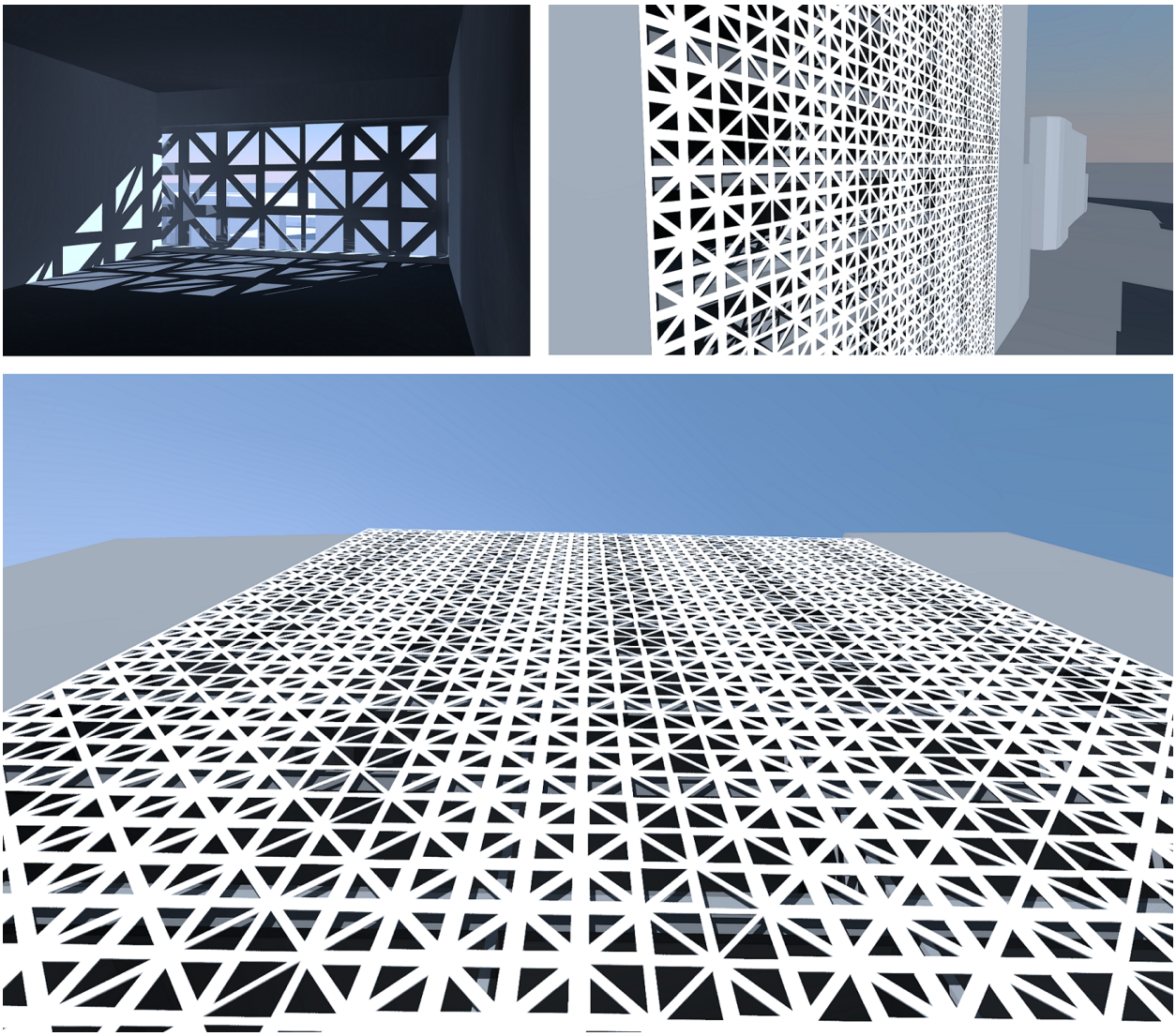
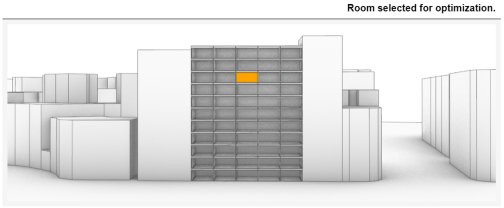


Fig. 5. Rendered images of the final model in POV-Ray.

6 Optimization

Each one of the designed rooms has its own lighting conditions, meaning they should be analyzed individually to guarantee ideal conditions. However, for this work, we decided to optimize just one of them.

Room selected for optimization.



The two metrics involved in the optimization process were Spatial Daylight Autonmy (sDA) and Annual Sun Exposure (ASE), and the objectives were:

$$\begin{aligned} &\text{minimize} && \text{ASE}_{1000,250lx}(x_1, x_2) \\ &\text{maximize} && \text{sDA}_{300\|50\%}(x_1, x_2) \end{aligned}$$

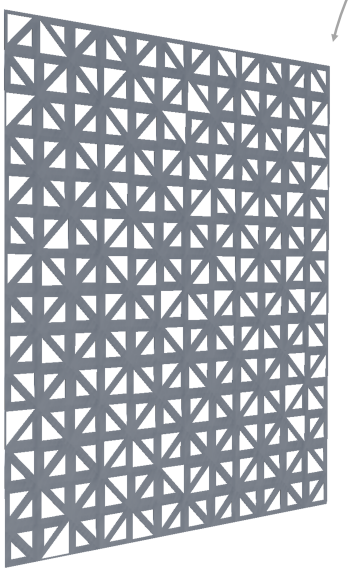
x_1 and x_2 are the variables for our optimization problem, representing: the min value for openings $x_1 \in \{0.1, 0.15, \ldots, 0.85\}$, and the spacing between the panels $x_2 \in \{10, 11, \ldots, 50\}$.

Finally, we tested the NSGA-II algorithm, with 250 solutions grouped in populations of 25.

Initial Population

Generates the initial population for the metaheuristic algorithms, based on the Latin Hypercube algorithm.

```
unscale(value, nmin, nmax, omin=0, omax=1) =
        (value .- omin) ./ (omax - omin) .* (nmax - nmin) .+ nmin
```



Pareto Front Plot

Choose the backend to visualize the Pareto front solutions:

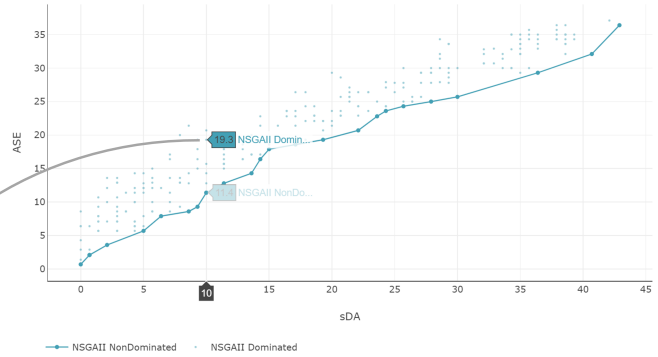
```
backend(autocad)
```

DataFrameRow

Row	min_opening Float64	n Float64	ASE Float64	sDA Float64
10	0.65	15.0	23.6	24.3

Create interactive Pareto front plot:

```
create_pfs(pfs, x=sDA, y=ASE, tpf=nothing, names=all_algorithms,
           colorscale=nothing, colors=["#43a0b5", "#B5557A", "#eb911c"],
           draw_dominated=true, layout=layout_tiling)
```



Uncostrained:

```
unconstrained_trace = parcoords(;line = attr(color=df.sDA, colorscale="YlGnBu"),
              dimensions = [
                  attr(range = [0.1,0.85],  label = "min_opening [m]", values = df.min_opening),
                  attr(range = [10,50], label = "n", values = df.n),
                  attr(range = [0,40], label = "ASE [%]", values = df.ASE),
                  attr(range = [0,50], label = "sDA [%]", values = df.sDA)
              ]);
unconstrained_plot = plot(unconstrained_trace)
```

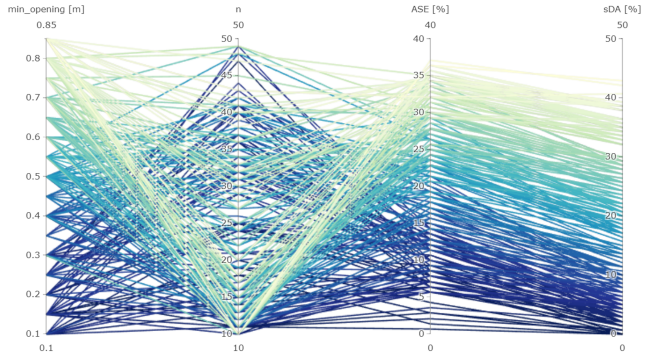


Fig. 6. Optimization of the façade project: problem description (top left), interactive Pareto Front (top right), generation of the selected solution in the chosen visualization tool (bottom left), and parallel coordinates graph (bottom right).

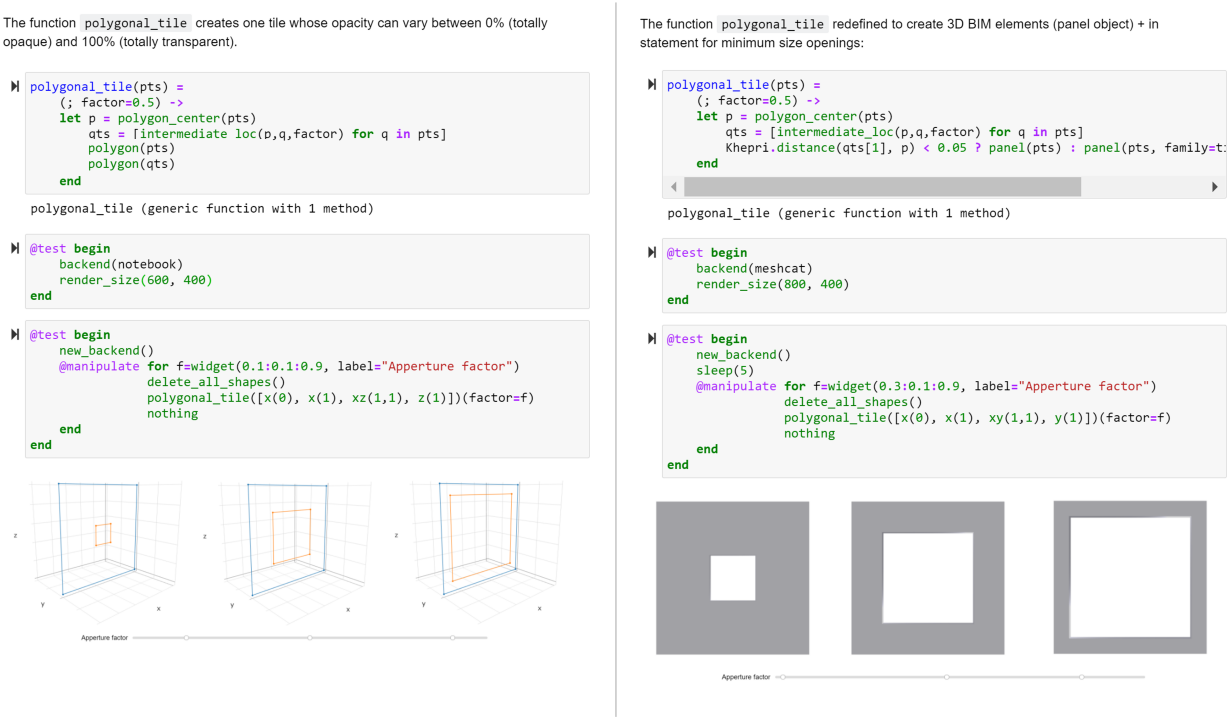


Fig. 7. Part of the development history of the polygonal tile algorithm: first (on the left) and final version (on the right).