ALGORITHMIC DESIGN EXPLAINED

Decomposing parametric 3D problems into 2D visual illustrations

RENATA CASTELO-BRANCO [1] and INÊS CAETANO[2] and ANTÓNIO LEITÃO[3]
[1,2,3]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa.
[2]BUILT CoLAB
[1]renata.castelo.branco@tecnico.ulisboa.pt, ORCID: 0001-9965-9558
[2]ines.caetano@tecnico.ulisboa.pt, ORCID: 0003-3178-7785
[3]antonio.menezes.leitao@tecnico.ulisboa.pt, ORCID: 0001-7216-4934

Abstract. Algorithmic Design (AD) is a promising approach that merges two distinct processes - design thinking and computational thinking. However, it requires converting design concepts into algorithmic descriptions, which not only deviates from architecture's visual nature, but also tends to result in unstructured programs that are difficult to understand. Sketches or diagrams can help explain AD programs by capitalizing on their geometric nature, but they rapidly become outdated as designs progress. In ongoing research, an automatic illustration system was proposed to reduce the effort associated with updating 2D diagrams as ADs evolve. This paper discusses the ability of this system to improve the comprehension of AD programs that represent complex 3D architectural structures. To understand how to best explain parametric 3D models using 2D drawings, this research explores problem decomposition techniques, applying them in the visual documentation of two case studies, where illustration aids different comprehension scenarios: illustrating for future use, and illustrating while designing as part of the AD process.

Keywords. Algorithmic Design, Automatic Illustration, Design Documentation, Design Representation.

1. Introduction

Algorithmic Design (AD) defines architectural designs through algorithms (Gerber & Ibañez, 2014), merging two very distinct but complementary processes, design thinking and computational thinking (Kelly & Gero, 2021). In doing so, AD increases design flexibility and reduces the effort required to explore several design ideas (Burry, 2013). Besides, AD can be coupled with analysis and simulation tools, facilitating the search for more sustainable and cost-efficient design solutions (Nguyen et al., 2014).

Despite AD's numerous advantages (Peters, 2013), the conversion of design thinking into computer programs (Woodbury, 2010) tends to result in unstructured products (Davis et al., 2011). The problem is aggravated with programs representing extensive 3D projects handled by multiple colleagues. Therefore, in addition to the challenging task of using algorithms to represent design concepts, members of the development team also struggle to understand the structure and behaviour of the AD programs developed, by others or by themselves in the past. In fact, comprehending programs is so central to the programming task that we spend more time reading code than writing it (Martin, 2008).

This paper extends previous research on the comprehension of AD programs that represent parametric architectural structures. It explores problem decomposition techniques to find the best strategies to visually explain these structures using static 2D drawings, while assessing the ability of an automatic illustration system to improve the comprehension of the corresponding AD programs.

1.1. DOCUMENTATION

Computer science addresses the program comprehension issue through textual documentation that explains the programs' structure and behaviour. Sadly, despite its importance for software maintenance, documentation is a dreadfully tiresome task often avoided by programmers (Bass et al., 2012). This has motivated the creation of automatic documentation tools (Allamanis et al., 2016; Iyer et al., 2016), which essentially translate algorithms into textual descriptions.

Nevertheless, AD often represents geometric concepts that are better expressed through sketches or diagrams rather than text (Self, 2019). Including these means of expression in AD documentation, particularly the drawings architects do during the creative process, is essential for a proper understanding of the architects' intentions toward their ADs. We can capitalize on the generated model, overlaying labels to visually illustrate parametric dimensions (Kelly et al., 2015). However, this is but a small part of the algorithmic logic in need of comprehension aid.

This belief triggered the creation of tools that allow for the integration of imagery in AD programs, serving as visual documentation (Castelo-Branco et al., 2022; Castelo-Branco & Leitão, 2022). Click or tap here to enter text.However, using hand-made drawings as documentation also has downsides; the most serious being that they rapidly become outdated as the design evolves.

1.2. AUTOMATIC ILLUSTRATION

Just as it happened with textual documentation, automation can alleviate the shortcomings of visual documentation. Castelo-Branco & Leitão (2023) proposed an automatic illustration system for AD that generates computer-made geometric illustrations explaining relevant aspects of the algorithmic description. Their envisioned workflow is for architects to generate illustrations with as little extra work as possible, and then automatically update them whenever changes are made to the AD. The illustration system promotes the decomposition and simplification of complex problems into smaller, manageable bundles of information, with a focus on 2D geometric illustrations. However, it has, thus far, only been used for the creation of proof-of-concept examples. The decomposition of large-scale and/or complex architectural solutions raises several other questions, which will be addressed next.

1.3. PROBLEM DECOMPOSITION

Architectural solutions are intrinsically 3D, thus posing some challenges to their textual or even 2D representation. Although the production technical drawings (such as plans and sections) allow architects to represent 3D elements through 2D descriptions with some rigor, the effort involved becomes unfeasible when the designs do not follow clear patterns of symmetry and orthogonality. The potential of AD to explore less conventional and more complex solutions only worsens the problem.

This research is inspired by the typical architectural design workflow and is based on the decomposition of the design problem into independent parts that can be described in 2D line-based schematics. This allows designers to expose a complicated design idea in simpler terms, by scaling down the complexity of the algorithmic description through incremental explanations. For such, we explore the capabilities of the automatic illustration system proposed in (Castelo-Branco & Leitão, 2023) to explain complex 3D shapes using 2D drawings.

2. Automatic Illustration Application

The above-mentioned illustration system, hereby referred to as *illustrator*, is evaluated in two architectural case studies. The evaluation focuses on its ability to improve the comprehension of (1) the AD programs' structure and behaviour; (2) their relationship with the corresponding 3D models; and (3) the impact of each design parameter on the 3D models' shape. The two case studies are parametric interpretations of existing buildings – the Al-Bahar Towers and the Lusail Stadium. Both were modelled using the Khepri AD tool running on top of the Julia programming language, to which the automatic illustration system is coupled.

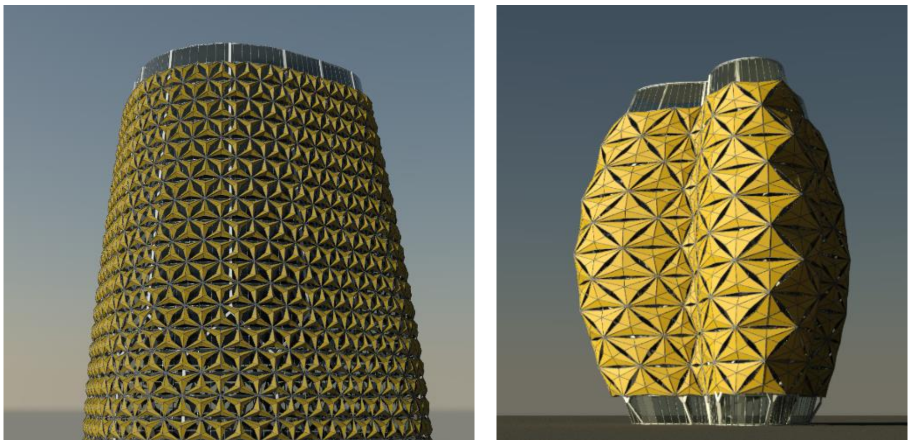
Figure 1. Al-Bahar Towers parametric interpretation variations.

2.1. AL-BAHAR TOWERS

The Al-Bahar towers in Abu Dhabi (Figure 1), UAE, designed by the Aedas studio, have one of the largest interactive sun-shading façade systems. To adapt to the local climate and reduce heat gains and glare, the two office towers were covered with a shading system inspired by the Islamic traditional wood-lattice screens (*mashrabiya*) that reacts to sun exposure by opening and closing its 1000 panels. The following analysis will address the parametric modelling of the building slabs and façade lattices.

2.1.1. Slabs

The building floor plan is shaped like an isosceles triangle with rounded corners, whose diameter varies across the building's height. In this AD interpretation, the shape of the slabs is described through a circular sinusoid that allows for multiple shape variations and contour deformations (see Figure 2).

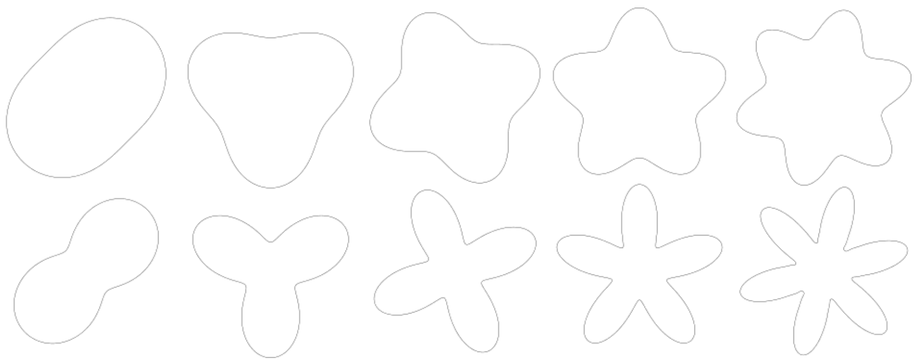

Figure 2. Slab illustration with different shapes (number of sinusoid cycles - **npc** parameter) and contour deformations (a 0 to 1 factor affects the sinusoid's amplitude - **Δrf** parameter).

```
@illustrator begin
 sind(α,ω,φ,t) = α*sin(ω*t+φ)

 pol_pts(p,r,△rf,φ,npc,n) =
  [p+vpol(r+sind(△rf*r,npc,0,β), β)
    for β in division(φ,φ+2π,n)]

slab_contour(p,r,△rf,npc,n) =
 closed_spline(
  pol_pts(p,r,△rf,0,npc,n))
end

with(call_depth, 1) do
 @illustrator pol_pts(x(0),6,0.1,0,3,5)
 @illustrator slab_contour(x(0),6,0.1,3,100)
 @illustrator_plus circle(x(0),6)
end
```

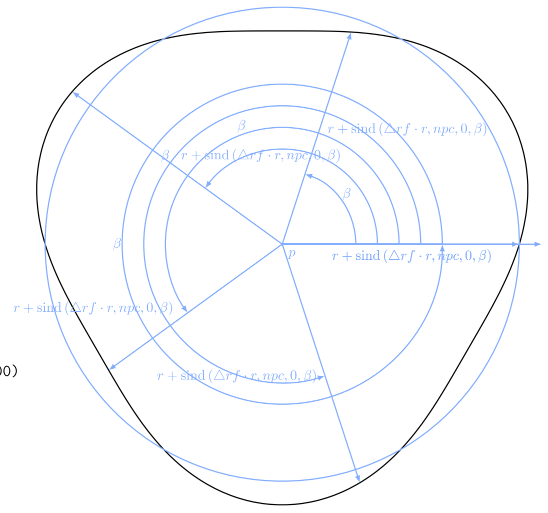
$r + \mathrm{sind}\,(\triangle rf \cdot r, npc, 0, \beta)$
β
$r + \mathrm{sind}\,(\triangle rf \cdot r, npc, 0, \beta)$
β
β
β
p
$r + \mathrm{sind}\,(\triangle rf \cdot r, npc, 0, \beta)$
$r + \mathrm{sind}\,(\triangle rf \cdot r, npc, 0, \beta)$
$r + \mathrm{sind}\,(\triangle rf \cdot r, npc, 0, \beta)$

Figure 3. Slab function algorithm and illustration.

Figure 3 presents the algorithm describing the slabs' shape and the illustration produced. Upon calling the illustrator (with the annotation **@illustrator**), the algorithm complements the generation of the slab's contour with a descriptive explanation of its parameters. The resulting illustration visually explains (1) the creation, using polar coordinates (function **pol_pts**), of spatial locations around point **p**, at a radial distance of **r+sind(△rf*r,npc,0,β)** and angular distance **β**; and (2) the materialization of such coordinates by passing a spline curve through them (function **slab_contour**).

Although we use the illustrator on a specific function call, this function typically calls another, and that one calls another, and so on, creating a cascade of function calls - the call chain. To control the depth in the call chain up until which the illustrator will illustrate, users may modify the **call_depth** parameter. In this case, we only wanted the illustrator to access one level, since **slab_contour** calls **pol_pts** within it, but we also call it, separately, for illustration purposes with a smaller number of points.

Finally, to include additional illustrations, other than the default ones provided by the illustrator, we can use the annotation **@illustrator_plus**. Figure 3 exemplifies this functionality with the addition of the imaginary circle around which the polar sinusoidal coordinates develop.

2.1.2. Façade Lattices

The building's façade lattices are mobile three-point stars that open and close in response to sunlight. In this AD interpretation, these elements have a variable number of vertices and aperture factors (Figure 4).

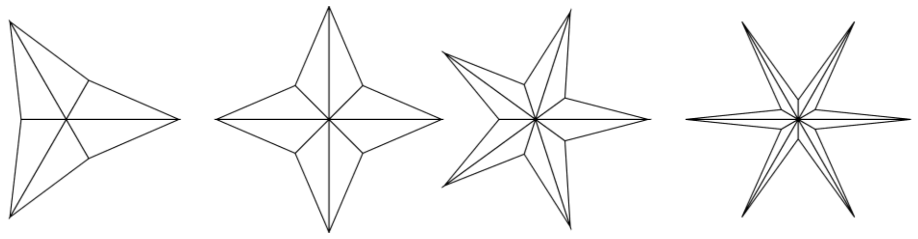
Figure 4. Illustration of the lattice with different number of vertices (**n** in **pol_pts**) and **f** factors controlling the lattice's aperture.

Figure 5 presents the algorithm that shapes the lattice and its illustration. In this case, we can identify a cascade of at least 3 function calls: **lattice** calls **mid_locs** and **mid_locs** calls **mid_loc**. The illustrator discriminates each function call by using different colours in the illustration. Moreover, to prevent cluttery illustrations, it also decreases the opacity level of the annotations as it goes down the call chain.

Upon calling the illustrator with the **call_depth** set to 3, we obtain the lattice's star-shape contours together with a three-color illustration scheme. In this scheme, the illustrations in purple (relative to the **mid_locs** function) represent the star's outer point pairs (**v1** and **v2**), which are used to calculate the intermediate locations **m**, and then the intermediate locations between the multiple **ms** and the centre **c,** given factor **f** (computations made by the **mid_loc** function and represented in yellow). The illustrations in blue represent the centre, inner and outer points of the lattice's star-shape (**in** and **out** pairs over which polygons are mapped).

In this case study, we showed the illustrator's default operations, which recognize and illustrate Julia primitives (such as *array comprehensions*) and Khepri primitives (such as **vpol** or **intermediate_loc**). Nevertheless, the default illustrations might be too general to properly explain any given case. As such, users can also create custom illustrations that explain specific functions. In the following case study, we devise custom illustrations to document the program and support the modelling process.

```
@illustrator begin
 mid_loc(c, v1, v2, f) =
  let m = intermediate_loc(v1, v2)
    intermediate_loc(m, c, f)
  end


 mid_locs(center, vertices, f) =
  let v1s = vertices[1:end-1],
      v2s = vertices[2:end]
    [mid_loc(center, v1, v2, f)
             for (v1,v2) in zip(v1s,v2s)]
  end


 lattice(vs, f) =
  let c = polygon_center(vs),
      in_vs = mid_locs(c, vs, f)
    [polygon(c, out, in) for (out,in) in zip(vs, in_vs)]
    [polygon(c, in, out) for (in,out) in zip(in_vs, [vs[2:end]...,vs[1]])]
  end
end
```

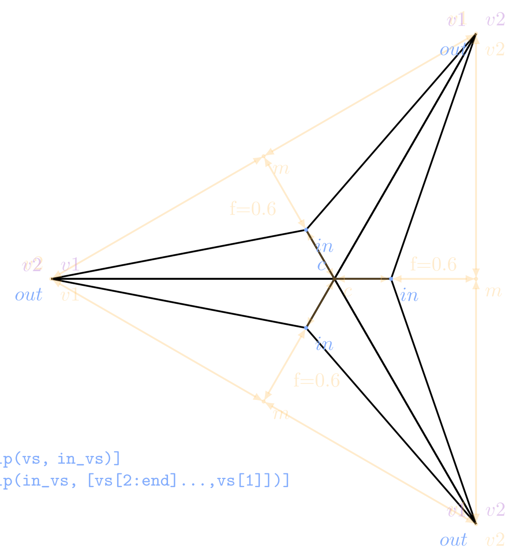

Figure 5. Lattice algorithm and illustration.

2.2. LUSAIL STADIUM

The Lusail Stadium, designed by Foster + Partners for the 2022 FIFA World Cup, resembles a golden cup with perforations that mirror the inside truss structure and control the amount of natural light passing through. Matching the sinuous movement of the facade, the pringle-shaped roof structure is composed of plastic membranes distributed in a radial pattern. Figure 6 shows two variations allowed by our AD interpretation of the stadium. The ensuing analysis will elaborate on the modelling of the façade supporting truss and the roof point matrix (see Figure 7).

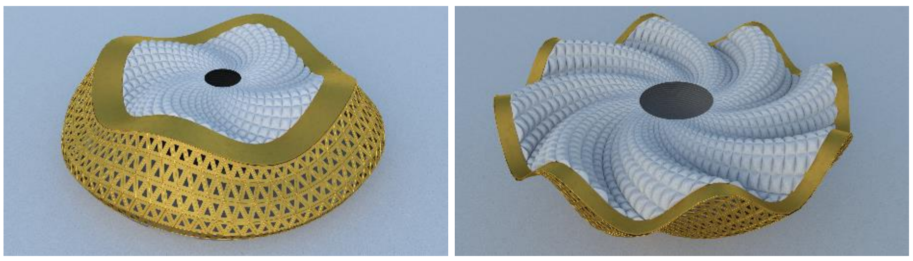

Figure 6. Lusail Stadium parametric variations.

2.2.1. Truss Structure

To create the planar Warren truss with verticals that sustains the façade, as shown in Figure 7 (left), we implemented the **planar_warren_truss** function (Figure 8 top) that receives two sets of locations (**as** and **bs**). Based on the given locations, the function creates truss bars according to the proposed layout. Along with the function, we defined a custom illustration to place numbered labels on the provided sets of points and to replace the **truss_bar** instruction with a **line**.

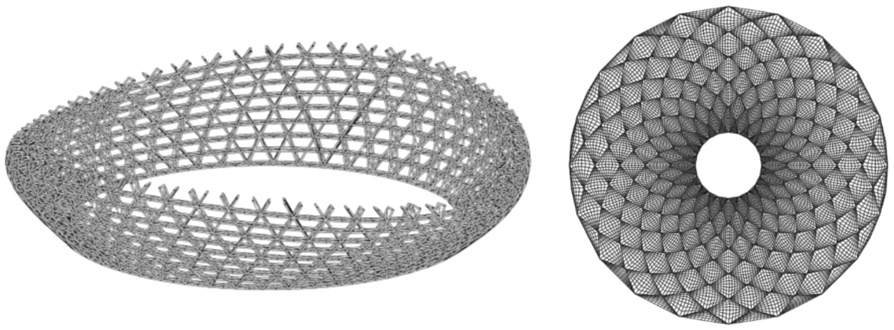

Figure 7. Details of the planar Warren truss structure holding up the façade (left) and the radial plastic membrane distribution on the roof (right).

```
planar_warren_truss_op(as, bs) =
  begin
    map(truss_bar, as, as[2:end])
    map(truss_bar, bs, bs[2:end])
    map(truss_bar, as, bs)
    map(truss_bar, as, bs[2:end]) → map(truss_bar, odds(as), evens(bs))
    map(truss_bar, bs[2:end], as[3:end]) → map(truss_bar, odds(bs[2:end]), evens(as[2:end]))
  end
```

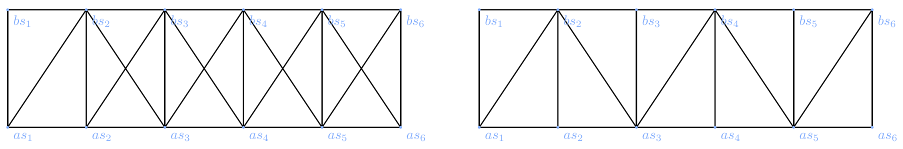

Figure 8. Planar Warren truss algorithm and illustration of its successive versions.

As shown in Figure 8 (top), we started by connecting all **as** with the following **as** and all **bs** with the following **bs** to create the upper and lower frames. We then connected all **as** to all **bs** to create the verticals; and all **as** to following **bs**, and the following **bs** to their ensuing **as** to create the slanted bars. The result, however, was not the intended one, with the illustration clearly showing that there were too many bars (Figure 8 bottom left) and that we should only connect odd **as** to even **bs**. After rewriting the algorithm (by replacing the red lines), we obtained the intended truss (Figure 8 bottom right).

2.2.2. Roof Matrix

The plastic membranes composing the roof have one of two shapes: diamonds in the inner rows and triangles in the rims (see Figure 7 right). We developed an algorithm to produce the roof point matrix and two other algorithms to produce each type of membrane based on either 3 or 4 of these points. However, with the first point configuration we developed for the matrix, it was difficult to identify the sequences of 3 and 4 points required for each membrane. Figure 9B shows the erroneous connections between the points of the original matrix.

To understand this problem, we developed a custom illustration for point matrices, which exposes the matrix's structure through numbered rows and columns (Figure 9A). With this visual aid, we could understand how to rearrange the matrix's columns to get the correct diamonds and triangle points. The result after modifying the matrix is illustrated in Figure 9C and the resulting point connections are shown in Figure 9D.

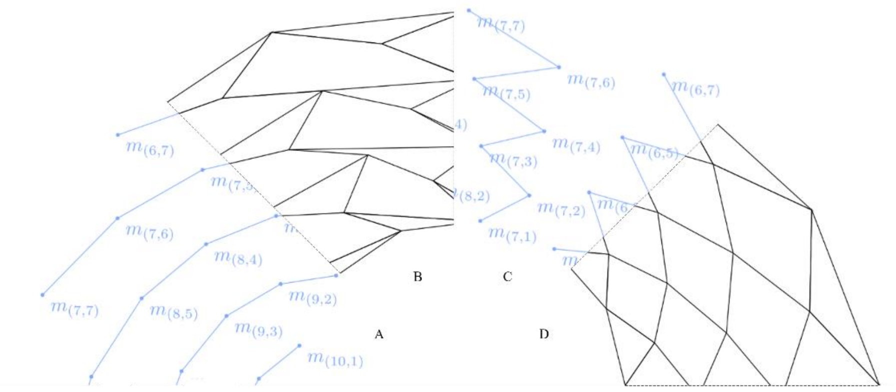

Figure 9. Roof point matrix illustration and application of polygons representing the membranes.

2.3. DISCUSSION

Each case study evaluated the illustrator in two scenarios: supporting comprehension and debug during the development of the algorithms and explaining them after development. To do so, we elaborated on how problem decomposition techniques can help document 3D elements in 2D illustrations.

For example, in the Al-Bahar towers, the complex façade pattern problem can be decomposed in three steps: (1) defining the lattice shape, (2) defining the façade point matrix, and (3) mapping the lattices onto the 3D matrix. The illustration presented here focused on the first step, where the problem is further decomposed into 3 functions. The descriptive scheme generated by the illustrator explains how, from an aperture factor and a set of vertices provided for the lattices contour, we can form a star-shaped lattice. More specifically, the illustration exposes the role of each of the 3 functions in this task: calculating one intermediate location, calculating multiple intermediate locations, and combining those location with the original vertices.

Naturally, limiting the illustrations to 2 dimensions leaves several AD computations out of the scope. In the Al-Bahar example, we are ignoring the 3D volumetry of the lattice, which is later achieved by adding to its central point a vector normal to the lattice surface plane. A second scheme could have been developed for this modified version of the lattice, depicting this additional computation. Withal, we argue that this is a rather typical design workflow: starting with small and simplified versions of the problem (in this case, a 2D lattice made of polygons) and eventually progressing to more refined and complex versions (a 3D lattice with sets of extruded surfaces instead of polygons).

The same applies to the slab case. Further developments to the slab algorithm transform the closed spline into an extruded 3D object and affect the slabs' radius with another sinusoid function responsible for the building's curvature along its height.

The Lusail Stadium roof structure is initially processed with polygons as well, for performance and debugging sake, but the final plastic membranes were achieved by lofting a series of curves. Applying the latter algorithms to the original point matrix would have likely yielded several modelling failure errors.

The truss case in the Lusail Stadium bypassed this workflow by starting off directly with the final 3D geometry (a truss bar) and defining the illustration as line depictions instead. It is, however, also resorting to a simplification of the problem by developing the algorithm over a simple 2D truss test. As can be seen in Figure 7 (left), the Planar Warren truss algorithm is later applied to more elaborate sets of locations in space.

In sum, both examples revealed the illustrator's capabilities to automate the documentation task, reducing the time and effort spent in creating descriptive schemes and in adjusting them as the AD evolves. They also demonstrated that there can hardly ever be a one-size-fits-all solution. Therefore, the illustrator allows users to extend its capabilities as needed. When reasonably abstract, these custom illustrations may then be integrated in the illustrator, enriching the existing domain library for future use.

3. Conclusion

This paper extended research on the comprehension of Algorithmic Design (AD) programs representing parametric 3D architectural structures, by combining program decomposition techniques with the creation of semi-automatic 2D drawings. It applied an automatic illustration system (the *illustrator*) to visually document two case studies. As is typical in design processes, to reduce the complexity of documenting 3D elements, problem decomposition techniques are used to break down the tridimensionality of these elements into simpler hierarchical 2D parts. The case studies assessed the capability of the illustrator to document each of these parts in different stages of the design process, particularly during and after design exploration - to boost comprehension while developing and to enhance comprehension in future use - as well as its capacity to adapt to the specific requirements of each design. In the latter case, the paper elaborated on how the illustration features can be extended or customized.

Although still under development, the illustrator has proved to be advantageous for automating visual documentation tasks, creating, and adapting 2D schemes that explain AD programs. We are currently conducting user studies to assess the impact of illustrations on the comprehension of AD programs. Future developments for the automatic illustration system include the extension of the operations currently supported and the program patterns recognized. The more patterns the illustrator is taught to recognize, the more design details it will know how to illustrate, and less work is required on the user's part. We also plan to explore the creation of 3D illustrations.

Acknowledgements

This work was supported by national funds of *Fundação para a Ciência e a Tecnologia* (FCT) with references UIDB/50021/2020 (DOI:10.54499/UIDB/50021/2020) and DFA/BD/4682/2020.

References

Allamanis, M., Peng, H., & Sutton, C. (2016). A convolutional attention network for extreme summarization of source code. In *Proceedings of the 33rd International Conference on Machine Learning,* 48, 2091–2100. https://proceedings.mlr.press/v48/allamanis16.html

Bass, L., Kazman, R., & Clements, P. (2012). *Software Architecture in Practice* (3rd ed.). Pearson Education (US).

Burry, M. (2013). Scripting Cultures: Architectural Design and Programming. In *Architectural Design Primer*. John Wiley & Sons. https://doi.org/10.1002/9781118670538

Castelo-Branco, R., Caetano, I., Pereira, I., & Leitão, A. (2022). Sketching Algorithmic Design. *Journal of Architectural Engineering,* 28(2). https://doi.org/10.1061/(ASCE)AE.1943-5568.0000539

Castelo-Branco, R., & Leitão, A. (2022). Comprehending Algorithmic Design. In *Computer-Aided Architectural Design Futures - Design Imperatives: The future is now!* (pp. 15–35). Springer, Cham. https://doi.org/10.1007/978-981-19-1280-1_2

Castelo-Branco, R., & Leitão, A. (2023). Illustrating algorithmic design. In *Computer-Aided Architectural Design. INTERCONNECTIONS: Co-computing Beyond Boundaries* (Issue 1819, pp. 36–50). Springer, Cham. https://doi.org/10.1007/978-3-031-37189-9_3

Davis, D., Burry, J., & Burry, M. (2011). Understanding visual scripts: Improving collaboration through modular programming. *International Journal of Architectural Computing,* 09(04), 361–376. https://doi.org/10.1260/1478-0771.9.4.361

Gerber, D., & Ibañez, M. (Eds.). (2014). *Paradigms in Computing: Making, Machines, and Models for Design Agency in Architecture*. eVolo Press.

Iyer, S., Konstas, I., Cheung, A., & Zettlemoyer, L. (2016). Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics,* 1, 2073–2083. https://doi.org/10.18653/v1/P16-1195

Kelly, N., & Gero, J. S. (2021). Design thinking and computational thinking: a dual process model for addressing design problems. *Design Science,* 7, e8. https://doi.org/10.1017/dsj.2021.7

Kelly, T., Wonka, P., & Mueller, P. (2015). Interactive Dimensioning of Parametric Models. *Computer Graphics Forum,* 34(2), 117–129. https://doi.org/10.1111/cgf.12546

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson.

Nguyen, A.-T., Reiter, S., & Rigo, P. (2014). A review on simulation-based optimization methods applied to building performance analysis. *Applied Energy,* 113, 1043–1058. https://doi.org/10.1016/j.apenergy.2013.08.061

Peters, B. (2013). Computation works: the Building of Algorithmic Thought. *Architectural Design: Computation Works: The Building of Algorithmic Thought,* 222(02), 8–16. https://doi.org/10.1002/ad.1545

Self, J. A. (2019). Communication through design sketches: Implications for stakeholder interpretation during concept design. *Design Studies,* 63, 1–36. https://doi.org/10.1016/j.destud.2019.02.003

Woodbury, R. (2010). *Elements of Parametric Design*. Routledge.