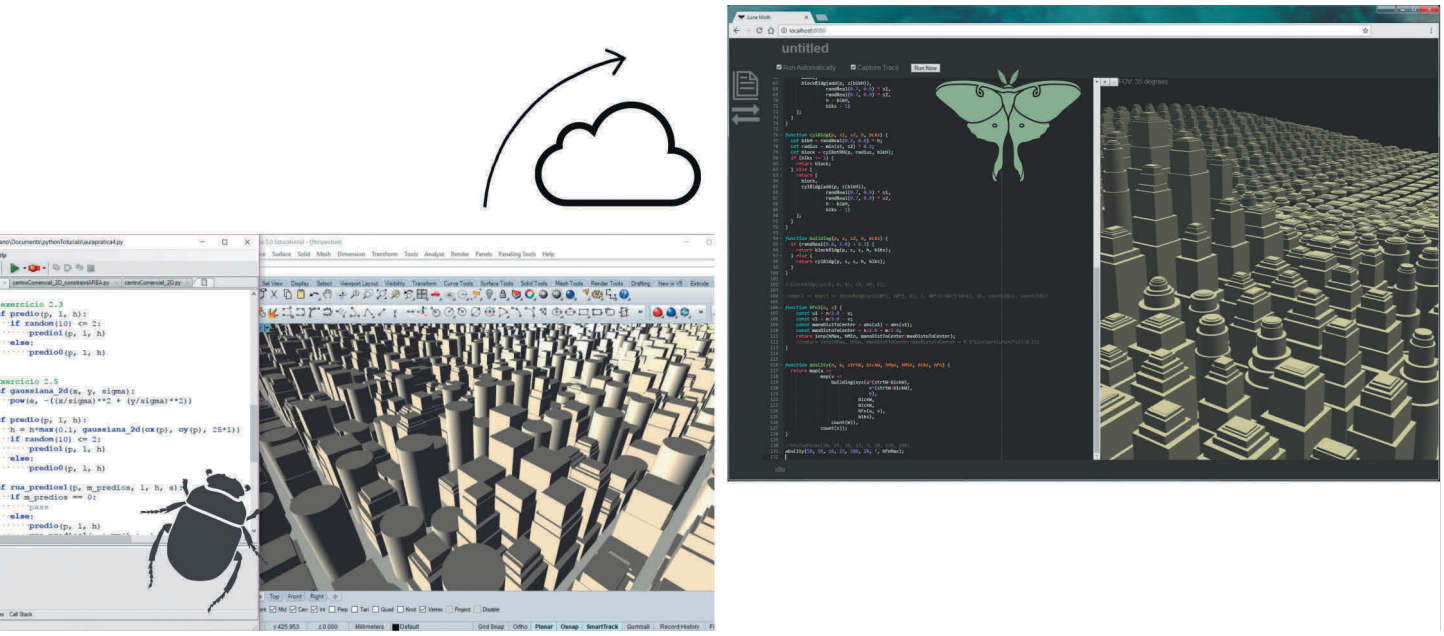
Luna Moth

Supporting Creativity in the Cloud

Pedro Alfaiate
Instituto Superior Técnico / INESC-ID

Inês Caetano
Instituto Superior Técnico / INESC-ID

António Leitão
Instituto Superior Técnico / INESC-ID



1

ABSTRACT

Algorithmic design allows architects to design using a programming-based approach. Current algorithmic design environments are based on existing computer-aided design applications or building information modeling applications, such as AutoCAD, Rhinoceros 3D, or Revit, which, due to their complexity, fail to give architects the immediate feedback they need to explore algorithmic design. In addition, they do not address the current trend of moving applications to the cloud to improve their availability.

To address these problems, we propose a software architecture for an algorithmic design integrated development environment (IDE), based on web technologies, that is more interactive than competing algorithmic design IDEs. Besides providing an intuitive editing interface which facilitates programming tasks for architects, its performance can be an order of magnitude faster than current aalgorithmic design IDEs, thus supporting real-time feedback with more complex algorithmic design programs. Moreover, our solution also allows architects to export the generated model to their preferred computer-aided design applications. This results in an algorithmic design environment that is accessible from any computer, while offering an interactive editing environment that integrates into the architect's workflow.

1 Migration from desktop application to the cloud.

INTRODUCTION

Throughout the years, computers have been gaining more ground in the field of architecture. In the beginning, they were only used for creating technical drawings using computer-aided design (CAD) software, but later, they began to integrate 3D modeling capabilities. However, modeling a complex building is still a time-consuming activity that requires several repetitive tasks that are not trivial to accomplish using just the functionalities provided by 3D modeling software.

This has led to the emergence of new design approaches, such as algorithmic design (Terzidis 2004). Algorithmic design is based on the use of programming, enabling architects to create their own program by describing their modeling intentions and letting the computer do the modeling task for them. Also, compared to the manual approach, it promotes a quicker and simpler handling of changes coming from uncertain design intents and emergent requirements (Leitão, Fernandes, and Santos 2013), allowing the designer to explore a broader design space.

To create their scripts, architects need to use a programming language, its runtime environment (including CAD and building information modeling [BIM] applications to visualize the obtained models), and an integrated development environment (IDE). Unfortunately, programming is not a trivial activity, particularly for those who are not specifically trained for it. With the aim of integrating programming in architects' workflow, some tools and programming languages have been carefully designed to facilitate this task.

PROGRAMMING IN ARCHITECTURE: PERFORMANCE, TRACEABILITY, AND SCALABILITY

Programmers need IDEs for general purpose development, but architects and designers need IDEs suitable to the algorithmic design domain. We claim that such IDEs need to provide two important features: traceability and real-time feedback. By traceability, we mean creating a relationship between parts of the program and those of the generated model. This is particularly important for the comprehension, maintenance, and debugging of algorithmic design programs (Leitão, Lopes, and Santos 2014). Real-time feedback is relevant for immediately understanding the effects of changes in the algorithmic design programs, helping architects to develop and change those programs.

Some IDEs have been developed to support visual programming languages, which are considered to be more intuitive than textual programming languages and, therefore, more attractive for those who do not have any programming experience (Zboinska 2015). Grasshopper and Dynamo are two examples of visual programming languages that also support traceability between the program and the model: when the user selects a component in the program, the corresponding 3D model components are highlighted. Another advantage of these visual programming languages is that they support real-time feedback of parametric changes when using sliders to represent the parameters.

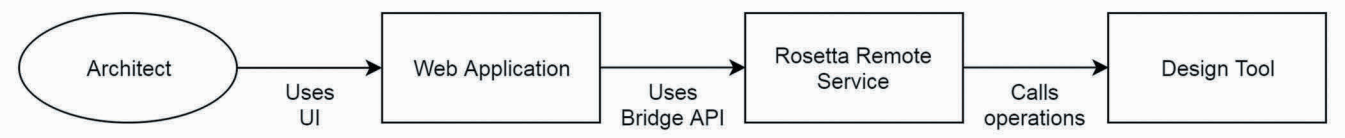
Unfortunately, visual programming languages do not scale well when applied to more complex projects (Leitão, Santos, and Lopes 2012). Visual programming scripts not only become unreadable and thus very difficult to change, but the real-time feedback of the sliders also begins to vanish as the overall performance decreases. Moreover, the traceability between the program and the model only exists in one direction, from program to generated model.

On the other hand, textual programming languages have mechanisms that make complexity more manageable, allowing algorithmic design programs to be smaller. Still, typical textual programming language IDEs lack ways to show the connection between program and results. Some research on this topic has been done (e.g., the Illustrated Programming approach, implemented in the Rosetta IDE [Leitão, Lopes, and Santos 2014]), to improv the comprehension of algorithmic design programs by integrating both sketch program and program model correlations. As it uses textual programming languages, it does not suffer as much from scalability problems. Additionally, the existing traceability is supported in both directions, i.e., from program to model and from model to program. Unfortunately, the current performance of the Rosetta IDE is not sufficient to support efficient traceability and much less to allow real-time feedback of non-trivial algorithmic design programs.

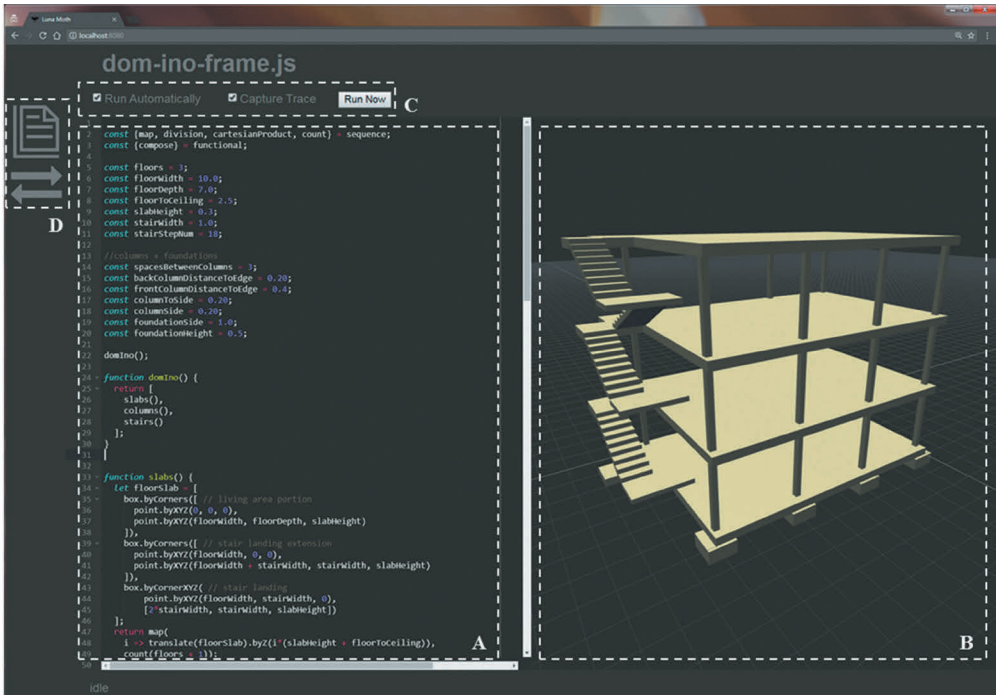
CLOUD-BASED PROGRAMMING ENVIRONMENTS

There are several IDEs that already explore web technologies: Light Table, a code editor implemented using web technologies that explores ways to improve programming in TPLs (Granger 2015); IPython, a programming environment for scientific computing that is used through a web page (Pérez and Granger 2007); and Processing, a programming language for the visual arts that can run on the web (Reas and Fry 2007; McCarthy et al. 2015; Reas et al. 2008).

These, however, do not target the algorithmic design field, which is better served with OpenJSCAD, a web application for 3D modeling using a textual programming language (Muller 2015), and Möbius, an algorithmic design environment implemented as a web application that combines block-based programming with associative programming (Janssen, Li, and Mohanty 2016).



2 The software architecture is composed of a web application that provides the development environment for architects, and the Rosetta Remote Service that connects the environment to their other design tools.



3

3 A print screen of the Luna Moth IDE: A) Text editor; B) 3D visualizer; C) Controls for running programs; D) Interface for exporting to external applications.

Apart from this, as web applications, OpenJSCAD and Möbius can always be kept up to date without requiring architects to do anything.

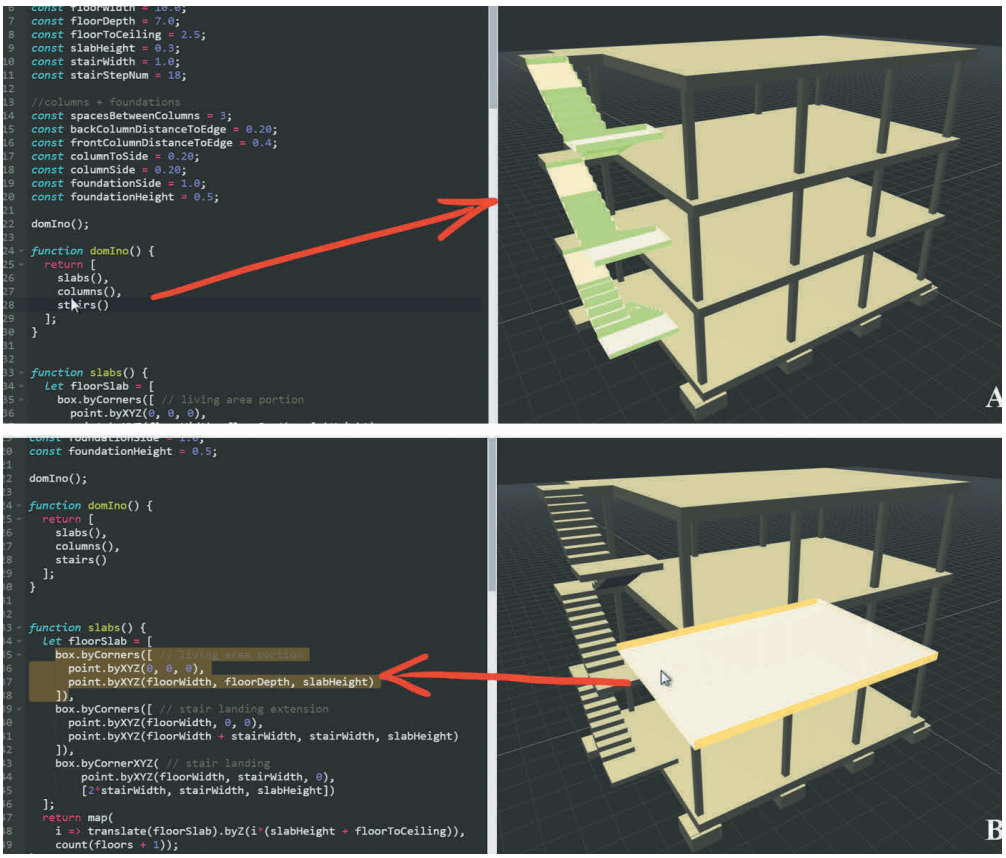
Even so, these two lack remote storage as provided by Onshape, a cloud-based CAD application (Hirschtick and McEleney 2012), and Clara.io, a cloud-based 3D modeling application (Houston et al. 2013). Work done in Onshape and Clara.io is kept on a remote server, thus facilitating their use regardless of the computer used by the designer. In addition, they also support collaboration, meaning that their users can work on the same model simultaneously. Nevertheless, Onshape and Clara.io do not support algorithmic design.

On the other hand, the cloud-based IDEs that do support algorithmic design, namely OpenJSCAD and Möbius, both lack editing features found in desktop algorithmic design applications, such as Grasshopper and Rosetta's traceability, and the latter's interoperability.

To sum up, most IDEs used for algorithmic design are desktop applications, providing limited support for remote work and collaboration, and requiring constant updates by their users. On the other hand, cloud-based IDEs are always up to date, but unfortunately, there is no cloud-based IDE capable of supporting an algorithmic design approach for architectural modeling that also integrates well with the typical architectural workflow.

Our aim is to increase architects' productivity when using an algorithmic approach by giving them an appropriate cloud-based programming environment. As such, we present a software architecture for a design tool that advances the state of the art in IDEs for algorithmic design in four different areas (Figure 2):

1. It has the required performance for real-time feedback of complex scripts;
2. It enables the user to see the bidirectional traceability relationship between the program and the resulting model;
3. It is available as a web application, not requiring installation or updates, and available for remote use;
4. It smoothly integrates into architects' workflow by generating the obtained results in other CAD tools they use.



4 Two examples of the traceability mechanism: A) From program to 3D model; B) From 3D model to program.

LUNA MOTH

To evaluate our proposal, we implemented Luna Moth, a web-based IDE for algorithmic design that supports traceability and real-time feedback. It runs user-written programs and immediately displays the obtained models, while maintaining the relationship between the different parts of the program and the corresponding parts of the model. Additionally, Luna Moth is able to interact with Rosetta (Lopes and Leitão 2011), which is used here as a remote service, allowing portability with different CAD, BIM, and analysis applications.

Figure 3 presents the user-interface of the Luna Moth IDE. The controls for running programs (Figure 3c) are above both the text editor (Figure 3a) and the 3D model visualizer (Figure 3b). The option for running programs targeting external applications, as well as a list of available programs, is on the left (Figure 3d).

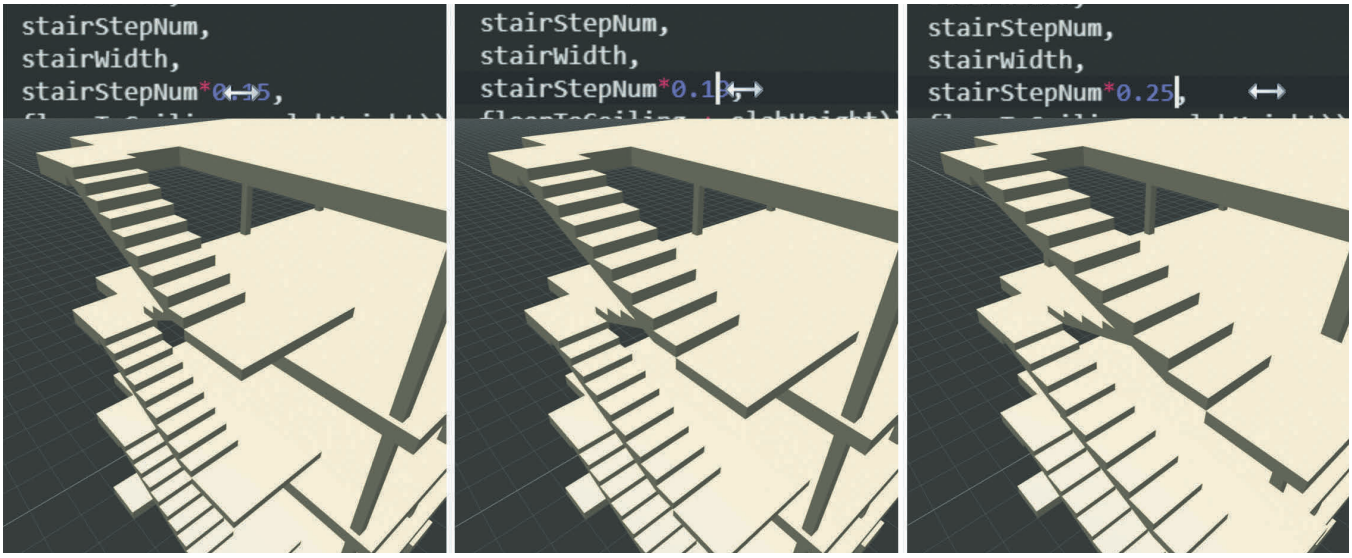
Handling Traceability

Traceability can be achieved by giving feedback to the programmer when interacting with the program or its results. Luna Moth embraces this in several ways. First, it lets the user adjust parameters more intuitively, while automatically rerunning the program and showing the new results in real time. Second, it reruns the program whenever its structure is changed, e.g., when a new statement is added to a function, an expression is changed, or a new function is added. Finally, it points out which part of the code corresponds to a certain 3D element or the other way around. In practical terms, when the user points at a function application, Luna Moth immediately highlights the shapes that it produced in the 3D view (Figure 4).

Note that Luna Moth does not track the execution of all program fragments, but only those related to function applications. This is a compromise to balance performance, since keeping track of traceability is a computationally intensive task that needs to be performed while the program runs.

Adjusting Literals

When a parameter, such as a number, is typed directly into a program's source code, it is called a literal value, or simply a literal. When using algorithmic design, architects find themselves repeatedly adjusting these literals to tweak the generated model. This is usually done by rewriting parts of the literal's text with higher or lower digits.



5 A sequence of examples adjusting literals, while automatically affecting the 3D model generated.

Adjusting literals this way often leads to errors, since it is easy to mistype characters. First, it is easy to increase the order of magnitude of the literal by adding one more digit by mistake. Second, it is also easy to make mistakes when increasing the literal in small increments. In most increments, users only replace the rightmost digit. They erase the digit and type the next one. They only need to move their hands to the key where the next digit is, i.e. to the right. However, when they reache the digit nine, they also need to increment the next and then type a zero. This requires a different hand movement. It takes more time to do and it is easier to accidentally hit the wrong keys.

These mistakes get amplified when the programming environment provides real-time feedback and begins rerunning the program before the error is corrected, thus leading to reduced responsiveness. The adjustment can be friendlier if done in a clearer way, as exemplified in Figure 5—instead of retyping the literal, Luna Moth enables the adjustment by simply "clicking and dragging," a movement that is similar to a slider.

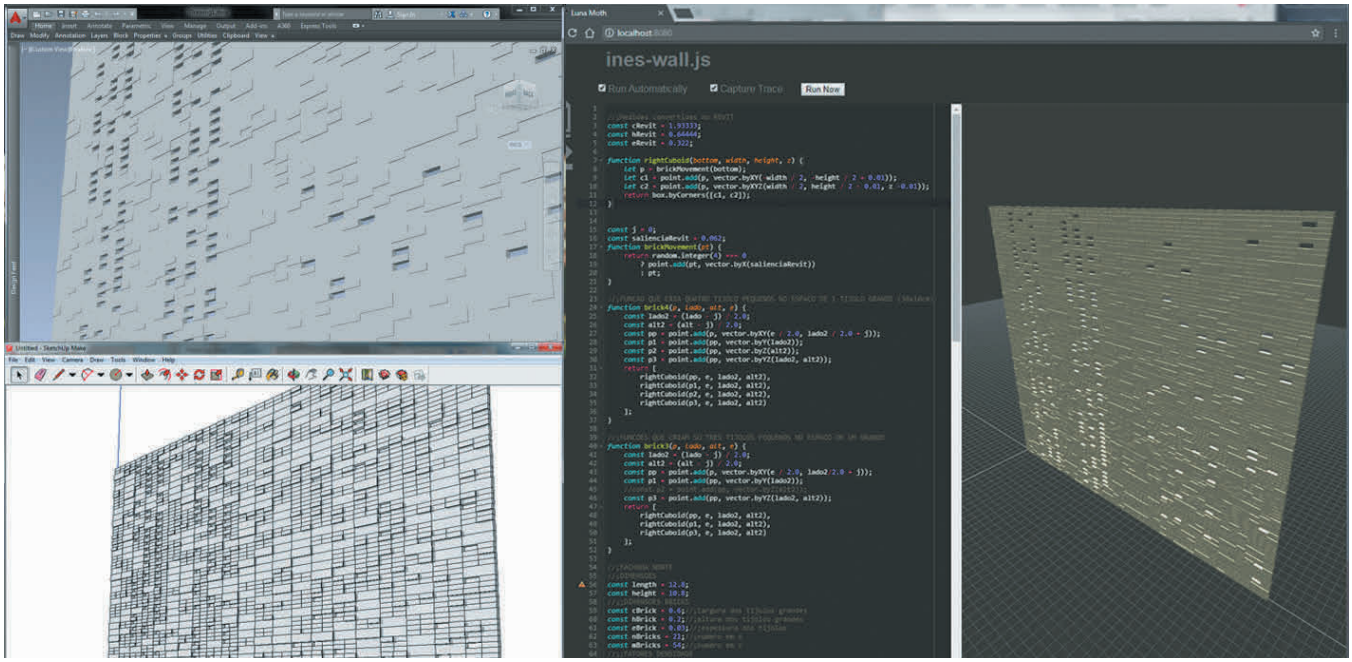
Running Programs

One of the fundamental parts of a programming environment is that it runs the programs written by its users. However, in the case of Luna Moth, running a program entails more than the typical process implemented in other IDEs, as it is necessary to ensure traceability between program and model. To this end, the program is syntactically analyzed, recovering its underlying structure, also known as its abstract syntax tree (AST). This AST is then transformed so that each function application also saves the produced results in a database. Finally, the modified AST is used to generate the actual program that runs in the browser. After execution and generation of a 3D model, every time the architect clicks on a function application or on an element in the model, the database is consulted, and the corresponding association is retrieved and then used to highlight both the expression and the model's element.

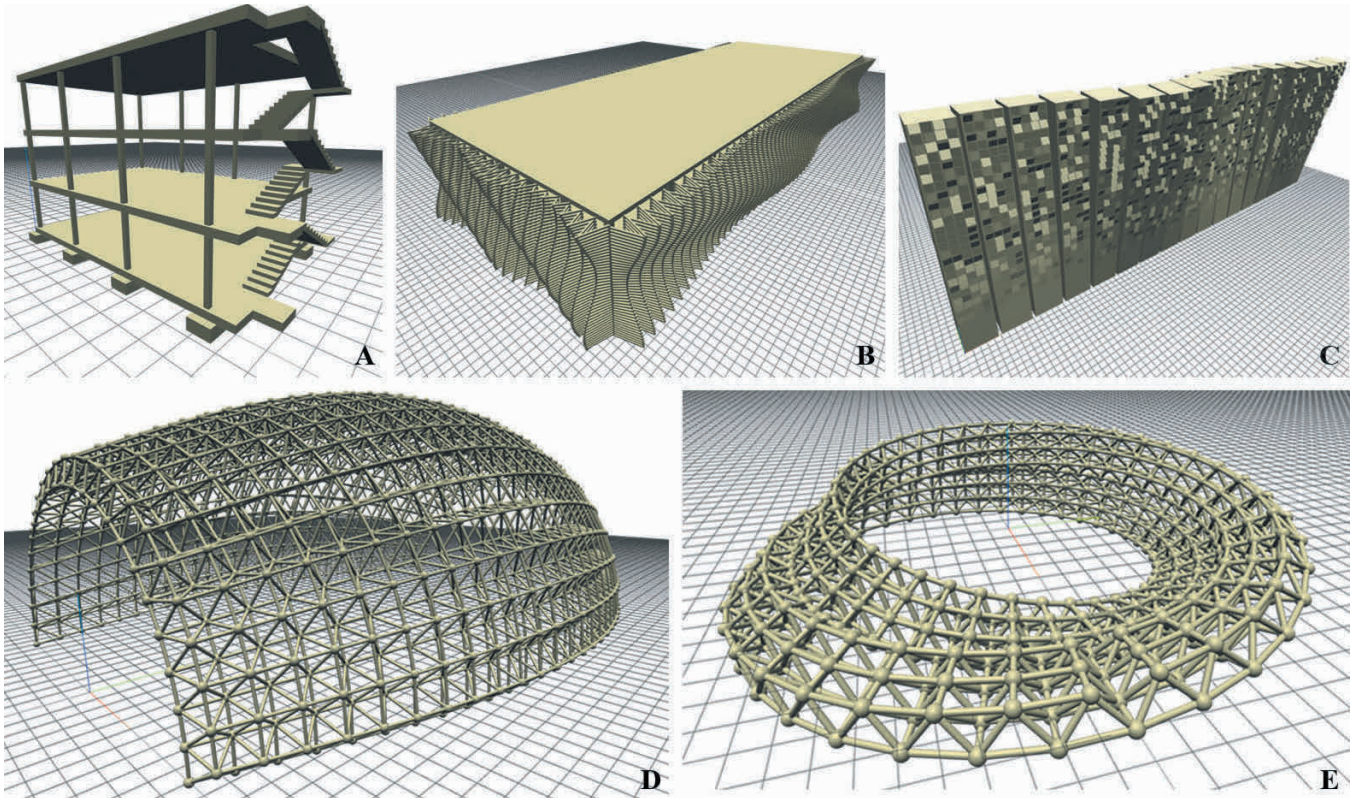
Workflow Integration

In order to integrate with the architect's workflow, cloud-based IDEs for algorithmic design should allow the generation of models in the traditional design tools that are typically used by architects. Some of the existing cloud-based tools have this critical limitation: even though architects can explore their models more easily and rapidly in the IDE, they turn out to be useless when it comes time to further develop them. This is one of the major factors that contributes to the unpopularity of some of these cloud-based IDEs in the architectural field, since they do not integrate well the design process of architects.

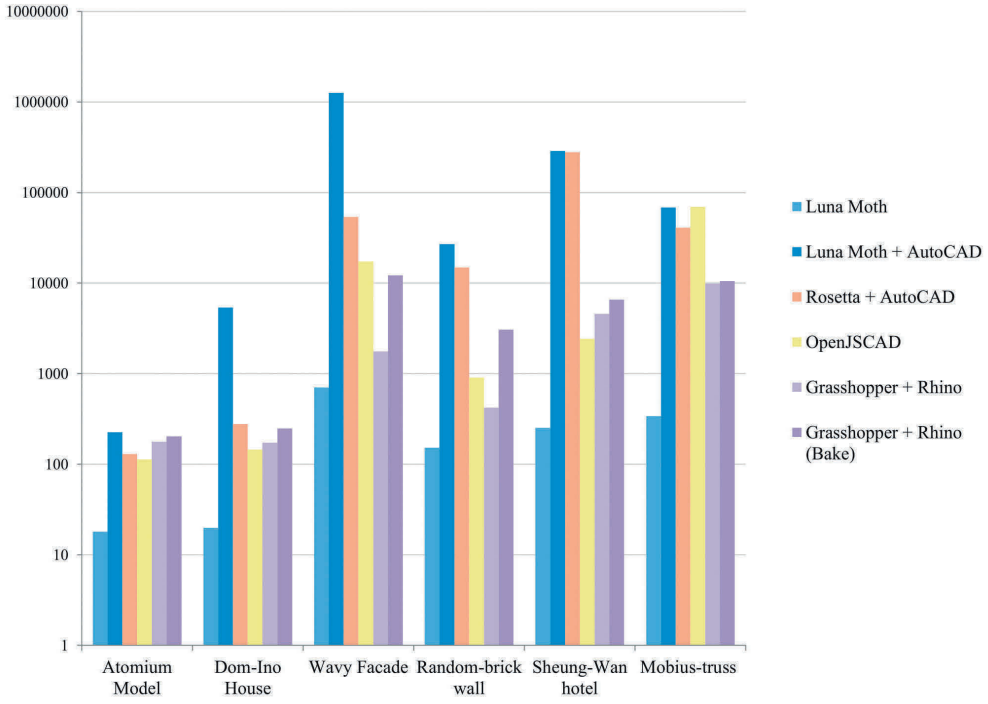
To address this need, we planned to connect Luna Moth to Rosetta (Lopes and Leitão 2011), which would open up the possibility of exploring all of its features, including 3D-modeling extensions, automatic visualization and rendering functionalities, and also the connection to other essential tools, like BIM tools and analysis tools. Unfortunately, due to security reasons, web applications are not allowed to communicate with applications running on the same computer. To overcome this limitation, we implemented an application to be installed on the architect's computer—the Rosetta Remote Service—that serves as a bridge between Rosetta and Luna Moth: when architects decide to further develop their designs using other tools, Luna Moth uses the bridging application to produce an identical model directly into the selected tool; the model is automatically generated from scratch using the same algorithms, thus avoiding any errors and



6 Workflow integration. The results of the program (on the right) have been passed to both AutoCAD and SketchUp (on the left).



7 Renders of the results explored using the Luna Moth IDE: A) Dom-Ino House; B) Wavy Façade; C) Nolan Façade; D) Arched Truss; E) Mobius Truss.



8 Running times for completing the generation of the test models. Note that the vertical axis uses a logarithmic scale.

8

loss of information that are typical of an export process.

Figure 6 shows an example of this functionality, where architects have created an algorithmic design program using the Luna Moth IDE, and when satisfied with the results, they select AutoCAD and SketchUp as modeling targets. Luna Moth then re-executes the algorithmic design program, generating identical models in both CAD applications.

RESULTS AND REFLECTIONS

Our goal was to propose a software architecture for bringing algorithmic design to the web browser, making it more easily used anywhere and, simultaneously, providing features to support architects in their programming tasks. To evaluate our work, we started by implementing the Luna Moth IDE. In a second stage, we developed several algorithmic design programs using this IDE, which reproduced some architectural examples that were previously explored using other algorithmic design environments. As shown in Figure 7, Luna Moth can produce interesting results with varying degrees of complexity, limited only by the number of modeling operations that are currently implemented. Actually, none of the selected examples used constructive solid geometry (CSG) operations, like intersection and difference, as these are not yet implemented. On the other hand, Luna Moth takes advantage of certain techniques, like higher-order functions (Leitão 2014), that have the ability to easily create complex designs.

Performance

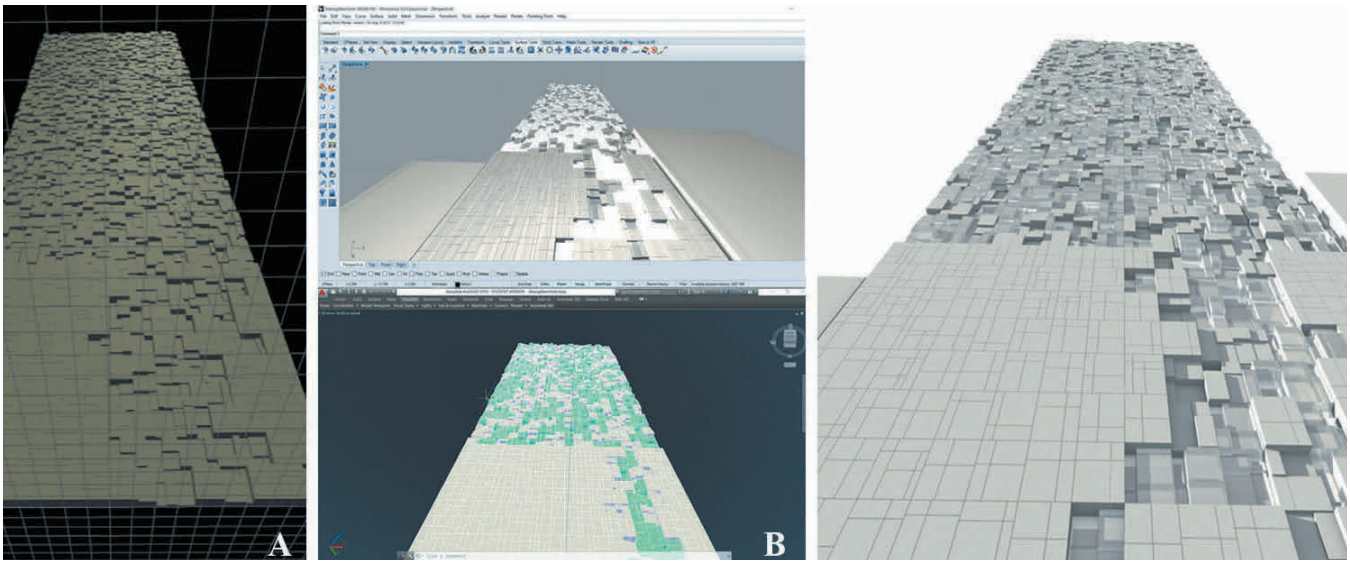
Performance plays an important role when using an algorithmic design environment. Therefore, we evaluated our solution's performance from three different perspectives by comparing:

1. The running performance of Luna Moth with other algorithmic design environments;
2. Its performance when connected to other design tools using the bridging application;
3. Its performance with and without traceability data collection.

For this, we timed several situations. First, we analyzed how long the process took while running in Luna Moth bridged to AutoCAD, and running in Rosetta connected to AutoCAD. Then, we measured the running times in OpenJSCAD and Grasshopper—the latter with and without baking geometry to Rhinoceros. Finally, we timed the running performance in Luna Moth with traceability both enabled and disabled. Note that this measurement of running times was done by generating identical models in each IDE.

We started by implementing versions of programs using the programming language of each environment, and then we measured the time each IDE took to generate the models (Figure 8).

The results in Figure 8 show that, first, Luna Moth is consistently faster than the other IDEs analyzed, sometimes with a difference



9 Architect's workflow process integration: A) Development of an Algorithmic Design program using Luna Moth; B) Usage of the bridge process to generate models in a CAD tool (above, in Rhinoceros 3D and, below, in AutoCAD); C) Render of the final model.

of at least one order of magnitude. Therefore, we can say that it provides faster feedback.

Second, running programs in Luna Moth alone is much faster than using it with the bridge process—e.g., "Luna Moth + AutoCAD"—whose times are 12–1800 times slower. Similarly, the bridging process is also slower than similar processes in other IDEs—"Rosetta + AutoCAD" and "Grasshopper + Rhino (Bake)"—which are up to 24 and 104 times faster, respectively. However, despite the considerable impact of the bridging process, it is only intended for a single use at the final stage of the design exploration process, i.e., when the architect is already satisfied with the resulting model, aiming to further develop it in their preferred CAD or BIM tool and thus its performance is not critical.

Lastly, we also measured the impact of traceability on performance. Our analysis shows that it increases running time by 10–50%, which is dramatically better than what is possible in Rosetta (Leitão, Lopes, and Santos 2014), and is within the limits expected from tools that can run programs with increased debugging information. Traceability data collection can be disabled by users to increase feedback speed, but the performance impact is worth taking when they want to get a better understanding of the program.

Workflow Integration

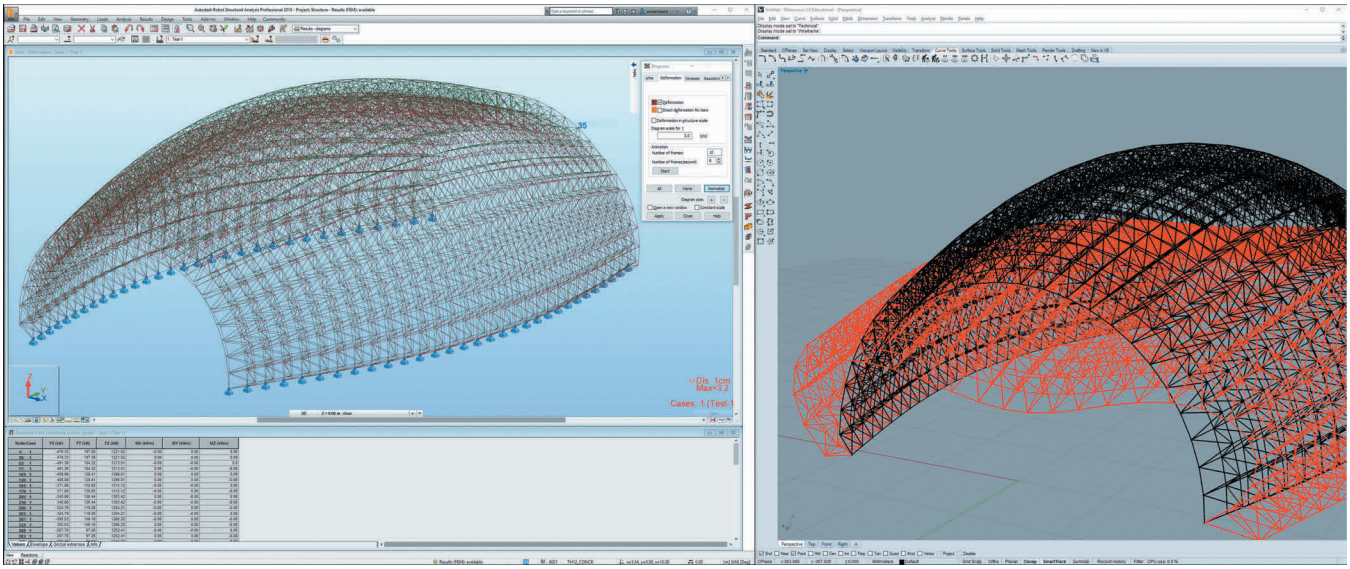
When exploring algorithmic design, architects benefit from a cloud-based IDE that is more intuitive, has better performance, and is available on every computer without installation or updates. Nevertheless, it is essential that the generated models can then be used in subsequent design stages, using the traditional tools that are typical of the design process.

Figure 9 represents the way Luna Moth integrates into the workflow of architects. From left to right, it shows the resulting model of a program developed using our cloud-based IDE, two identical models resulting from the bridged process of Luna Moth (one was generated in Rhinoceros and the other in AutoCAD), and finally, a render of the final model using Rhino's Renderer.

On the other hand, Figure 10 shows the bottom-left truss structure of Figure 7 being structurally analyzed using Robot. This demonstrates that a purely geometric model initially explored in Luna Moth can then be further developed, improved, and enriched with information until it reaches its final stage. Therefore, in regards to the continuous design process that is characteristic of architecture, we may state that Luna Moth adequately supports the initial phases and it does not limit subsequent ones.

CONCLUSION

One emerging trend in architectural practice revolves around the use of programming to explore new design possibilities, often called algorithmic design. However, not only are algorithmic design tools limited to desktop applications, but they also do not simplify the task of programming for architects. To overcome these limitations, we proposed a software architecture for a cloud-based IDE suitable for algorithmic design, which we then implemented as the Luna Moth IDE. Besides being available online, hence not requiring any installation or updates, Luna Moth integrates features intended to facilitate programming for architects. Such features include embracing traceability between



10 Structural analysis of the truss in Figure 7D. Left: the truss structure being structurally analyzed using Robot. Right: the corresponding truss model in Rhinoceros with the resulting (amplified) structure deformation in red.

program-model and model-program and supporting real-time feedback when changing both parts of the script as well as the input parameters.

In this paper, we explained the components of Luna Moth, namely, a web application to explore algorithmic design programs, and a desktop application to enable a bridging process to generate the models produced in the web application in the traditional tools used by architects, including CAD, BIM, and analysis tools. The latter requires installation, but only when architects decides to further develop their design using desktop tools.

We evaluated our solution by first presenting some examples explored using the Luna Moth IDE. Then we tested the performance of Luna Moth by measuring the programs' running times, and finally, we compared them to running times on other algorithmic design tools, such as Grasshopper, Rosetta, and OpenJSCAD. Based on the results, we concluded that Luna Moth can run programs faster than the other measured IDEs. The effects of keeping track of traceability on the program running times were also analyzed: although traceability makes programs around 10–50% slower, this is a huge improvement over the performance of Rosetta's traceability, which makes it usable on large programs as opposed to toy examples. Finally, we explored the way Luna Moth fits into architects' workflow and, to this end, we developed a bridging process. Despite not being as efficient as other approaches, it is intended to be used very infrequently, and only in the final phases of the algorithmic design process.

We plan to address the limitations of Luna Moth in the near future, namely by (1) improving the programming experience by performing static analysis and code completion; (2) supporting the addition of illustrations to programs as seen in Rosetta (Leitão, Lopes, and Santos 2014) and IPython; (3) supporting multiple programming languages; (4) improving the debugging experience; (5) improving the environment's traceability; (6) supporting common modeling primitives; and (7) improving the bridging process's performance. These improvements will not change the fundamental design ideas of Luna Moth but will make it even more competitive. The next result of this research is thus an algorithmic design tool that adequately serves the initial design stages, while enabling the project's continuity in the subsequent phases.

ACKNOWLEDGEMENTS

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013, and by the PhD grant under contract of University of Lisbon (UL), Instituto Superior Técnico (IST) and the research unit Investigação e Inovação em Engenharia Civil para a Sustentabilidade (CERIS).

REFERENCES

Granger, Chris. "Lighttable." 2015. http://lighttable.com/ (accessed May 15, 2017).

Hirschtick, Jon, and John McEleney. Onshape. 2012. https://www.onshape.org (accessed May 15, 2017).

Houston, Ben, Wayne Larsen, Bryan Larsen, Jack Caron, Nima Niketrat, Catherine Leung, Jesse Silver, Hasan Kamal-Al-Deen, Peter Callaghan,

Roy Chen, and Tim McKenna. 2013. "Clara.io: Full-Featured 3D Content Creation for the Web and Cloud Era." Presented as Studio Talk at ACM SIGGRAPH, article 8. Anaheim, CA: SIGGRAPH.

Janssen, Patrick, Ruize Li, and Akshata Mohanty. 2016. "Möbius: A Parametric Modeller for the Web." In *Living Systems and Micro-Utopias: Towards Continuous Designing, Proceedings fo the 21st Annual Conference of the Association for Computer-Aided Architectural Design Research in Asia*, 157–166. Melbourne, Australia: CAADRIA.

Leitão, António, R. Fernandes, and L. Santos. 2013. "Pushing the Envelope: Stretching the Limits of Generative Design." In *Proceedings of the 17th Conference of the Iberoamerican Society of Digital Graphics*, 235–238. Valparaíso, Chile: SIGRADI.

Leitão, António, J. Lopes, and L. Santos. 2014. "Illustrated Programming." In *ACADIA 14: Design Agency, Proceedings of the 34th Annual Conference of the Association for Computer Aided Design in Architecture*, edited by David Gerber, Alvin Huang, and Jose Sanchez, 291–300. Los Angeles: ACADIA.

Leitão, António., L. Santos, and J. Lopes. 2013. "Programming Languages for Generative Design: A Comparative Study." *International Journal of Architectural Computing* 10 (1): 139–162.

Leitão, António. 2014. "Improving Generative Design by Combining Abstract Geometry and Higher-Order Programming." In *Rethinking Comprehensive Design: Speculative Counterculture, Proceedings of the 19th International Conference on Computer-Aided Architectural Design Research in Asia*, edited by Ning Gu, Shun Watanabe, Halil Erhan, Matthias Hank Haeusler, Weixin Huang and Ricardo Sosa, 575–584. Kyoto: CAADRIA.

Lopes, Jose, and António Leitão. 2011. "Portable Generative Design for CAD Applications." In *Integration Through Computation: Proceedings of the 31st Annual Conference of the Association for Computer Aided Design in Architecture*, edited by by Joshua Taron, Vera Parlac, Branko Kolarevic and Jason Johnson, 196–203. Banff/Calgary, Canada: ACADIA.

McCarthy, L., Eastmond, E., Shiffman, D., Johnson, J., & Lavigne, S. 2015. "p5. js". https://p5js.org/ (accessed May 15, 2017).

Muller, R. K. 2015. "Openjscad.org." https://openjscad.org/ (accessed May 15, 2017).

Pérez, Fernando, and Brian E. Granger. 2007. "IPython: a system for interactive scientific computing." *Computing in Science and Engineering* 9: 21–29.

Reas, Casey, and Ben Fry. 2007. *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press.

Resig, John, Ben Fry, and Casey Reas. 2008. "Processing.js.". http://processingjs.org/ (accessed May 15, 2017).

Terzidis, Kostas. 2004. "algorithmic design: A Paradigm Shift in Architecture?" In *Architecture in the Network Society: Proceedings of the 22nd Conference on Education in Computer Aided Architectural Design in Europe*, 201–207. Copenhagen: eCAADe.

Zboinska, Malgorzata A. "Hybrid CAD/E platform supporting exploratory architectural design." *Computer Aided Design* 59 (2015): 64–84.

IMAGE CREDITS

All drawings and images by the authors.

Pedro Alfaiate holds an MSc in Information Systems and Computer Engineering, specialized in Software Engineering and Interaction and Visualization. He is passionate about programming, user-interfaces, web technologies, and everything 3D.

Inês Caetano is a Portuguese architect graduated at Instituto Superior Tecnico (University of Lisbon), a researcher at INESC-ID, and a PhD student at the same university, exploring the integration of algorithmic design methods in the design, analysis and optimization of façade designs.

António Leitão has a BSc in Mechanical Engineering, an MSc in Electronics Engineering, and a PhD in Computer Science and Engineering, all from Instituto Superior Técnico (IST) of the University of Lisbon. Currently he is Assistant Professor at the same university, Scientific Coordinator of the Software Engineering Group at INESC-ID, and Coordinator of the Architecture and Computation Group, teaching, lecturing, and researching on bringing together the fields of Computer Science and Architecture.