Luna Moth

A Web-based Programming Environment for Generative Design

Pedro Alfaiate[1], António Leitão[2]
[1,2]INESC-ID/Instituto Superior Técnico
[1,2]{pedro.alfaiate|antonio.menezes.leitao}@tecnico.ulisboa.pt

Current Generative Design (GD) tools require installation and regular updates. On top of that, programs that are created using them are stored as files, which have to be moved and shared manually with others. On the other hand, web applications are accessible using just a web browser and they can also store information remotely, meaning that it does not need to be moved and is easily shared with others. Consequently, GD tools should also be available as web applications to get the same functionality. We present Luna Moth, an IDE for GD available from the web that shows the relationship between a program and its results and integrates into the architect's workflow. Then, we give examples where Luna Moth's features help the architect during the programming process. Finally, we compare Luna Moth's performance with other IDEs, namely, Grasshopper, OpenJSCAD, and Rosetta.

Keywords: *Generative Design, Web application, Design tool integration,*

INTRODUCTION

Generative Design (GD) is a design process that comes partially from the automation of modeling tasks in Computer-Aided Design (CAD) applications using programming. GD uses computers as a new medium for artistic expression (Maeda 2001) that can be used by architects, as shown by Terzidis in Expressive Form (Terzidis 2003). Having a faster and more flexible process for building 3D models allows the architect to explore more variations of a design.

Furthermore, using GD as a new design process promotes a simpler handling of changes coming from uncertain design intents and emergent requirements, which evolve as the understanding of the problem improves or as the project's needs change (Fernandes et al. 2014). In fact, programs are unambiguous parameterized representations of designs, which only need small changes to parameters or functions to express changes.

Interactivity in GD IDEs

In order to improve the programming process, Integrated Development Environments (IDEs) and Programming Languages (PL) were developed, some of them directly addressing the needs of GD. For example, in Rhinoceros we have PLs like RhinoScript and RhinoPython, and in AutoCAD we have VisualLisp. They allow the architect to access the host CAD application's functionalities, such as operations to create or transform geometry. These PLs are textual programming languages (TPLs) meaning that their programs are represented as text.

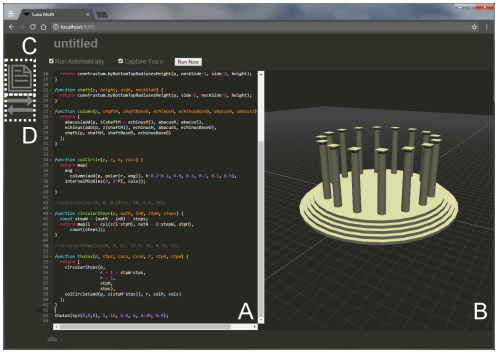


Figure 1
Luna Moth's editing interface

However, the mode of interaction with these PLs and their IDEs leaves much to be desired in the context of GD. The usual interaction with them follows a sequence that begins with the architect writing a portion of code, running the program, waiting for the program to finish running, looking at the generated model, and repeating the sequence until he is satisfied with the result. In this mode of interaction, the architect has no feedback regarding the changes he makes to the program while he is making it. It is not until he finally runs the program again that he gets the outcome. Given enough time between runs, he may start to miss the details that changed, which will make it harder to make sure the program does what is intended.

IDEs like Grasshopper for Rhinoceros and Dynamo for Revit tackle this problem by running programs continuously while changes are made. This allows the architect to immediately see their effects, which, in turn, allows him to make corrections as soon as he finds that the results are not what he intended. Moreover, while testing his programs, the architect can use tools like sliders to control input parameters and see the effects of changes almost immediately. Apart from this, these IDEs use visual programming languages (VPLs) where programs are represented graphically by connecting functions, represented by boxes, with wires that represent the flow of data from one function to another. As such, they can be seen as more intuitive for beginners.

Unfortunately, these IDEs quickly lose interactivity as programs grow. For bigger programs, sliders lose immediate feedback since the program takes too much time to run, which makes architects give up using sliders and, instead, change parameters textually. This performance problem comes both from the complexity of the GD program, which can make running times grow beyond the limit for immediate feedback, and from the way CAD applications are implemented, since they were designed to handle human interaction and not the volume of operations generated by GD programs. Solving the first cause might require the use of more efficient programming models and techniques to keep program complexity down, while the second can be alleviated by implementing dedicated visualizers to avoid using a CAD application entirely, e.g., an OpenGL viewer (Leitão et al. 2014).

Apart from improving performance to allow for better feedback, IDEs can make it easier for architects to understand programs. A program is a rather abstract representation of the design and it can be hard to understand what each part of it is intended to represent, even more so when it grows in complexity. One way to make programs easier to understand is to include documentation with them. As pointed out in Illustrated Programming (Leitão et al. 2014), architects already make sketches to help them formalize their design into a program which can serve as documentation. Therefore, the IDE should allow sketches to be part of programs. The work in (Ferreira 2016) went a little further by improving the perception of the relationship between sketch and program. In addition to documentation, it is also easier to understand programs with traceability as also pointed out in (Leitão et al. 2014), that is, being able to trace an element of the 3D model back to the parts of the program that created it. The opposite direction, from a part of the program to the parts of the 3D model, is also helpful. Unfortunately, GD IDEs that support traceability have limitations. Dynamo and Grasshopper only support it in one direction, from program to model, while Rosetta (Lopes and Leitão 2011) sup-

ports both directions but is too slow even for small programs.

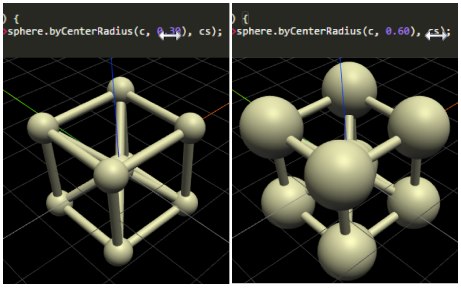


Figure 2
An example of numeric parameter adjustment. Clicking and dragging changes the value of the parameter.

Web applications

The web has seen a big increase in popularity, which has become even stronger by the standardization of web technologies, such as HTML5 (Hickson and Hyatt 2011) and WebGL (Marrin 2011), which allowed web applications to achieve user experiences on par with desktop applications. In addition, as web applications run on remote computers, they are always accessible without installation or updates. This has led to the creation of many web application counterparts of common desktop applications. For example, office productivity tools, like Microsoft Word, Excel and PowerPoint, have seen the appearance of their web application counterparts such as Microsoft Office 365. Furthermore, 3D modeling web applications and CAD web applications have also appeared. One example of the first is Clara.io (Houston et al. 2013), for 3D modeling and animation, and one regarding the second is OnShape [1], for Product Design/Engineering. Moreover, these web applications can save information remotely, which removes the need to transfer files between computers, and they can also support collaboration over great distances.

Closer to GD, experimental web applications have also appeared. One example is OpenJSCAD [2], for 3D modeling with Boolean operations using a TPL, and another is Möbius (Janssen et al. 2016), for 3D modeling using node- and block-based VPL. However, they do not have functionality for remote storage nor for collaboration found in the previous web applications. In addition, they also lack features that help understanding programs found in the GD IDEs from the previous section.

Goals

As mentioned in the previous section, web IDEs for GD are still lacking features that make them suitable for practical use. A modern GD IDE that addresses the previous problems needs to: (1) have good accessibility, being available on any computer with internet access and without requiring installation and updates; (2) be interactive, letting architects explore GD easily, giving them feedback and showing the relationship between program and results; (3) integrate easily with the CAD applications already used by architects, so that they can combine their GD experiments into their normal workflow.

In this paper, we present an experimental IDE, Luna Moth, a HTML5/JavaScript-based web application that harnesses the performance and graphical capabilities of modern web browsers and that can connect to the other CAD applications used by architects. Using Luna Moth, architects can write their GD programs, visualize the results without being chained to a particular computer, and can easily integrate results into their normal workflow.

We can summarize the structure of the paper as follows:

- We describe Luna Moth's interface for creating GD programs, which makes use of immediate feedback and traceability mechanisms.
- We describe the way Luna Moth integrates with other design tools used by the architect.
- We show how Luna Moth's features can help the architect during the programming process.
- We compare Luna Moth with OpenJSCAD, Grasshopper, and Rosetta by measuring the times for running programs and displaying their results.

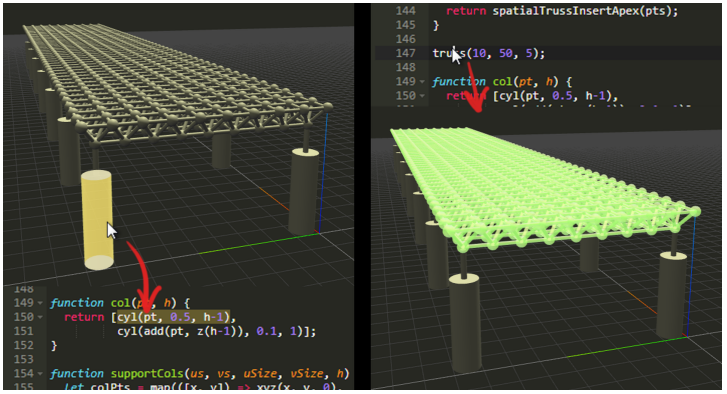


Figure 3
An example of Luna Moth's traceability. On the left, clicking on a part of the model shows the part of the program that created it. On the right, clicking on a part of the program shows the parts of the model that it created.

Figure 4
Luna Moth's software architecture

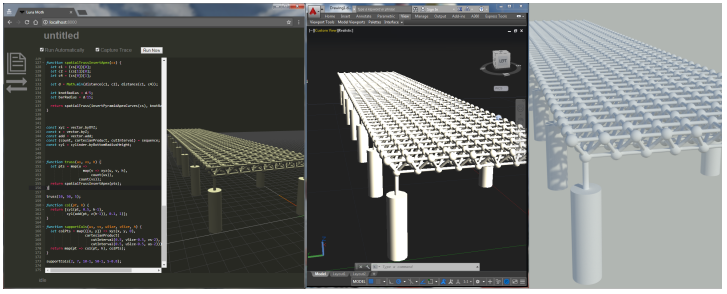


Figure 5
A truss created in Luna Moth, generated in AutoCAD, and then rendered using AutoCAD's renderer.

LUNA MOTH OVERVIEW

User interface

Luna Moth's editing interface consists of a source code editor (A) and a 3D view (B), as can be seen in figure 1. Apart from the main editing area, the interface also includes panels for managing programs (creating, opening, and deleting)(C) and for connecting Luna Moth to CAD applications (D).

The 3D model displayed in the 3D view is kept in sync with the results of the program in the source code editor, providing immediate feedback to the architect. Whenever program changes, Luna Moth reruns it and regenerates the 3D model. Moreover, Luna Moth also includes something akin to sliders to change numeric parameters. As such, instead of having to change the individual digits of the parameters, the architect can click and drag on parameters to change them. Figure 2 shows an example of use of this functionality.

The interface also makes it possible to understand which parts of the 3D model were created by an expression of the program by pointing at it. Like-

wise, it is also possible to go the other way, pointing at any part of the 3D model to know which expression created it. By letting the architect go both ways, the interface facilitates the understanding of the relationship between program and results – see figure 3.

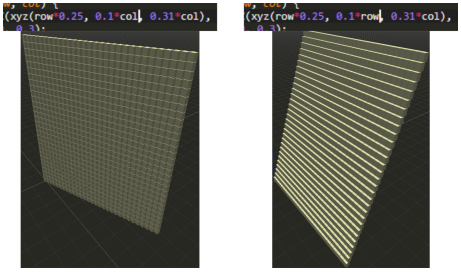


Figure 6
Correction of protrusion from depending on the column to depending on the row.

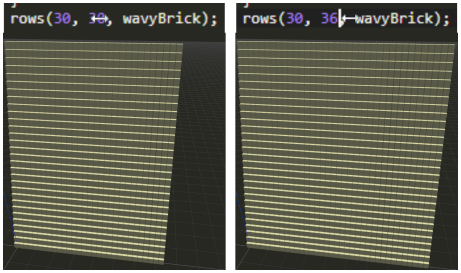


Figure 7
Increasing the wall's length by clicking and dragging.

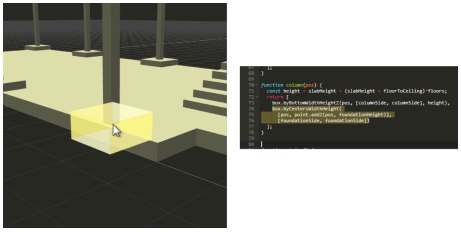


Figure 8
Clicking on an apparently wrong 3D element shows the function that created it.

Programming Language

Regarding the programming language used to write programs, Luna Moth supports the JavaScript TPL. We chose a TPL since, as described in (Leitão et al. 2012), although VPLs are more intuitive, they do not scale well for big programs. When visual programs grow in complexity, they become big nets of interconnected nodes that are hard to understand and modify. On the other hand, textual programming languages have mechanisms, like functions, that allow architects to create abstractions that hide how a certain task is performed from the rest of the program. As such, architects can create the rest of the program without worrying about all the details of each part, thus, focusing their attention on higher-level concepts. For example, creating a roof in a 3D model does not depend entirely on how the support below is created; both tasks share parameters, like the shape of the building, but they are otherwise independent from a 3D modeling perspective. Consequently, these can be packed in different functions. Afterward, they can be used in another part of the program without knowing their details.

We assume that someone using Luna Moth is at least comfortable with using TPLs, which can require more study upfront when compared to VPLs. Nonetheless, the initial investment quickly pays off since it is easier to adapt programs to accommodate more changes, given the increased flexibility of TPLs.

Workflow Integration

A GD IDE can only have a significant impact in the design process if it can integrate into the architect's workflow. This is typically supported in GD IDEs by exporting to a common file format which is recognized and/or required by other tools. Instead of exporting, Luna Moth connects to the CAD application and, then, generates the model from scratch there. To achieve the connection with other design tools, Luna Moth uses the software architecture shown in figure 4.

To use this functionality, architects have to run the Rosetta Remote Service on their computer, which lets Luna Moth know which design tools exist on that computer. Afterward, they select the desired design tool in Luna Moth and start the connection. Then, Luna Moth uses Rosetta Remote Service to generate the results of their program directly in the design tool. As such, when they reach the desired solution, they can then connect to a design tool, such as Au-

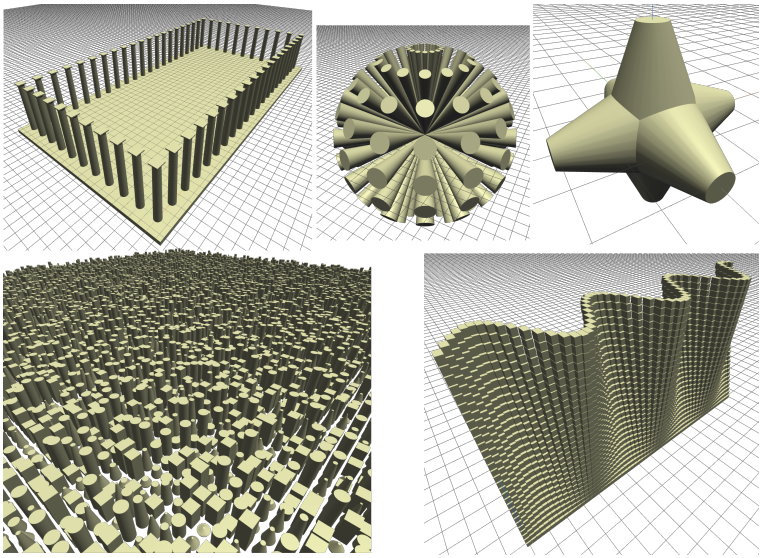


Figure 9
Other examples created with Luna Moth

toCAD, to generate drawings or render the solution with higher detail – see figure 5.

This intermediary step is necessary since there are no other means for web applications to know which applications exist in a computer. However, it does require the architect to run something on his computer in addition to the web browser, which goes against the principle that the web browser is the only software needed to use Luna Moth. Still, this is only true when the architect wants to connect Luna Moth to his design tools. The rest of the time, he can use Luna Moth without a problem with just a web browser.

As the Rosetta Remote Service uses Rosetta (Lopes and Leitão 2011) to connect to design tools, Luna Moth can connect to design tools other than AutoCAD. Due to recent extensions to Rosetta (Feist et al. 2016)(Leitão et al. 2017), Luna Moth can also connect to BIM and analysis tools, such as Revit and Radiance.

RESULTS AND REFLECTIONS

Programming Experience

As mentioned earlier, Luna Moth supports a TPL and keeps the view of results in sync with the current version of the program. In this section, we give examples of how Luna Moth can help with the development of a program.

Suppose an architect wants to create a façade composed of bricks. He created functions to make a straight grid of bricks and wants to control how much each brick is protruded. He decides that the protrusion should depend on the brick's position in the façade, i.e. its row and column. As Luna Moth has immediate feedback, he sees how each change affects the resulting façade. As such, if he wants the protrusion to increase as the row increases and starts changing the program to make it happen but, instead, makes it depend on the column, he will see that the result is wrong immediately. In this case, he can just as quickly correct the bug, as seen in figure 6.

Figure 10
Running times of four examples in Luna Moth, Rosetta, OpenJSCAD, and Grasshopper. The vertical axis represents time in milliseconds and uses a logarithmic scale.

After correcting this bug, he may want to tweak the code's numerical parameters to make the wall more or less steep. He can adjust those parameters by clicking and dragging them. The same goes for the number of bricks horizontally and vertically, in which case, the architect may want to see the effect on a bigger or lengthier wall (Figure 7).

In another situation, consider that an architect is developing a program that creates a model in separate parts, for example, a building skeleton with columns, slabs, and stairs. In this situation, he may notice that some columns are not appearing in the right locations, therefore, he can use Luna Moth's traceability, pointing at one of them to be directed to the function that creates it (Figure 8). From there, he can start to examine the function to understand why it is creating columns in the wrong location. Furthermore, the architect can also use traceability to find the remaining columns created by that function and check whether they are also incorrect. This time, he uses traceability in the reverse direction, from program to results.

In the same way the architect clicks on the model to get to a function that is not producing the right results, he can also do this to find a function that he wants to experiment on.

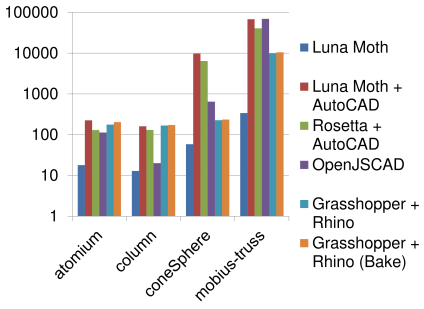
Examples

In addition to the previous examples, we also implemented other examples that can be seen in figure 9.

Performance

As part of the evaluation of Luna Moth, we also compared the running times of programs in Luna Moth with the running times in other IDEs, namely Grasshopper, OpenJSCAD, and Rosetta.

For each program, we measured the running times in Luna Moth not connected to design tools, in Luna Moth connected to AutoCAD, in Rosetta connected to AutoCAD, in OpenJSCAD, and in Grasshopper connected to Rhinoceros. Each IDE has a different programming language. Like so, we implemented each program using each IDE's programming language. These times can be seen in the chart from figure 10.



These measurements show that, when disconnected from other design tools, Luna Moth can run programs faster than Rosetta, OpenJSCAD, and Grasshopper, sometimes by one or more orders of magnitude. This tells us that Luna Moth can provide faster feedback to changes to programs compared to the other IDEs.

On the other hand, the measurements also show that, when connected to AutoCAD, Luna Moth is slower than the other IDEs. Nonetheless, this is explained by taking into account the communication time between Luna Moth, Rosetta Remote Service, and the connected design tool, and the time that the design tool takes to execute the desired commands. Nevertheless, this functionality is aimed to be used when the architect has already developed a program, in which case, it is used only once, therefore, fast feedback is not as important. This also means that Luna Moth is not connected to other design tools most of the time spent during the development of programs, consequently, it can be considered faster than the other IDEs.

CONCLUSION

As collaboration in architecture projects occurs between further and further apart teams, they have to use better ways of collaborating remotely. Web applications provide a possible path for supporting that collaboration. As such, design tools also have to make such collaboration possible. However, current GD IDEs were not designed with this collaboration in mind. Apart from that, a GD IDE also needs to

be adapted to architects so that they can understand and modify programs quickly. Moreover, the GD IDE must integrate into the architectural workflow, embracing the tools that are typically used in it.

With this in mind, we created Luna Moth – a web application that supports the programming task by providing immediate feedback to changes, a way to change parameters intuitively, traceability between a program and its results, and that also integrates into the workflow of the architect. In the paper, we showed an example of use of Luna Moth where we explained how the included features included help in the programming process. Lastly, we compared Luna Moth with other GD IDEs in terms of running times. Luna Moth performed better than the others when running programs by itself. On the other hand, when connected to other design tools, Luna Moth got slower than the others. In spite of this, Luna Moth is only expected to be connected to other design tools when the architect has already developed a program, meaning that it is faster than the other IDEs throughout most of the programming process.

Future work will focus on improving editing experience with features such as illustrated programming, code completion (and help in knowing the library of available functions), better code navigation, and further exploration of traceability. In addition, we plan to improve the performance of running programs when connected to other design tools and to add support for remote collaboration.

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 Rosetta project under contract PTDC/ATP-AQI/5224/2012.

REFERENCES

Feist, S, Barreto, G, Ferreira, B and Leitão, A 2016 'Portable Generative Design for Building Information Modelling', *Living Systems and Micro-Utopias: Towards Continuous Designing, Proceedings of the 21st International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*, Hong Kong, pp. 147-156

Ferreira, G 2016, *An Enhanced Programming Environment for Generative Design*, Master's Thesis, Instituto Superior Técnico

Hickson, I and Hyatt, D 2011, 'HTML5: A vocabulary and associated APIs for HTML and XHTML', in ., . (eds) 2011, ., Web Hypertext Application Technology Working Group

Houston, B, Larsen, W, Larsen, B, Caron, J, Nikfetrat, N, Leung, C, Silver, J, Kamal-Al-Deen, H, Callaghan, P and Chen, R 2013 'Clara. io: full-featured 3D content creation for the web and cloud era', *ACM SIGGRAPH 2013 Studio Talks*, p. 8

Janssen, P, Li, R and Mohanty, A 2016 'MöBIUS: A parametric modeller for the web', *Living Systems and Micro-Utopias: Towards Continuous Designing, Proceedings of the 21st International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2016)*, pp. 157-166

Leitao, A, Lopes, J and Santos, L 2014 'Illustrated Programming', *ACADIA 14: Design Agency, Proceedings of the 34th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, Los Angeles, pp. 291-300

Leitão, A, Castelo Branco, R and Cardoso, C 2017 'Algorithmic-Based Analysis - Design and Analysis in a Multi Back-end Generative Tool', *Protocols, Flows, and Glitches - Proceedings of the 22nd CAADRIA Conference*, Xi'an Jiaotong-Liverpool University, Suzhou, China., pp. 137-146

Leitão, A, Fernandes, R and Santos, L 2014 'Pushing the Envelope: Stretching the Limits of Generative Design', *Blucher Design Proceedings*, pp. 235-238

Leitão, A, Santos, L and Lopes, J 2012 'Programming languages for generative design: A comparative study', *International Journal of Architectural Computing*, pp. 139-162

Lopes, J and Leitão, A 2011 'Portable generative design for CAD applications', *Integration Through Computation - Proceedings of the 31st Annual Conference of the Association for Computer Aided Design in Architecture, ACADIA 2011*, pp. 196-203

Maeda, J 2001, *Design by Numbers*, MIT Press, Cambridge, MA, USA

Marrin, C 2011, 'WebGL specification', *Khronos WebGL Working Group*, ., p. .

Terzidis, K 2003, *Expressive Form: A Conceptual Approach to Computational Design*, Taylor & Francis

[1] https://www.onshape.com

[2] http://www.openjscad.org