
Inside the Matrix: Immersive Live Coding for Architectural Design

International Journal of Architectural Computing

XX(X):1–12

©The Author(s) 2020

Reprints and permission:

sagepub.co.uk/journalsPermissions.nav

DOI: 10.1177/ToBeAssigned

www.sagepub.com/

SAGE

Renata Castelo-Branco¹, Catarina Brás¹ and António Leitão¹

Abstract

Algorithmic Design (AD) uses computer programs to describe architectural models. These models are visual by nature and, thus, greatly benefit from immersive visualization. To allow architects to benefit from the advantages of Virtual Reality (VR) within an AD workflow, we propose a new design approach: Live Coding in Virtual Reality (LCVR). LCVR means that the architect programs the design while immersed in it, receiving immediate feedback on the changes applied to the program. In this paper, we discuss the benefits and impacts of such an approach, as well as the most pressing implementation issues, namely the projection of the programming environment onto VR, and the input mechanisms to change the program or parts of it. For each, we offer a critical analysis and comparison of the various solutions available in the context of two different programming paradigms: visual and textual.

Keywords

Virtual Reality, Algorithmic Design, Live Coding, Programming Environments, Interaction Mechanisms

Algorithmic Design

Nowadays, Algorithmic Design (AD) is becoming increasingly important in the architectural practice, having been adopted by well-known architecture studios such as Foster + Partners, Gehry Partners, and Zaha Hadid Architects. This research follows the definition of AD provided by Caetano et al.¹, which states that an algorithmically-designed project can benefit from parametric flexibility if the entities in the design are logically connected. In that case, changes applied to the parameters are automatically propagated to the rest of the model², allowing the user to explore design variations with ease.

Beyond model flexibility, AD also assists the designer in the creative process by automating repetitive tasks and allowing a finer control over the modelling of complex geometries, which are frequently difficult to produce manually³. Finally, AD facilitates and enhances model visualization (e.g., enabling the automatic generation of renders and animations), analysis (e.g., structural, lighting, and energy-efficiency analysis), optimization, and fabrication.

Considering the advantages AD brings to the Architecture, Engineering, and Construction (AEC) industry⁴, many AD tools have been developed⁵ and two main paradigms stand out: Visual Programming Languages (VPLs), which simplify

the learning process but lack scalability and abstraction^{6,7}, meaning that as programs grow in complexity, they become hard to understand and navigate; and Textual Programming Languages (TPLs), which scale better, allowing for larger programming projects, however, usually requiring more extensive programming knowledge and presenting inadequate representation mechanisms for the innate visual nature of architectural projects⁸.

The two paradigms present a common flaw: the difficulty architects face in understanding AD programs and relating them to the building's concept. This difficulty unfortunately means that AD largely remains a demanding and time-consuming endeavour.

Model Visualization

The typical interaction with architectural 3D models entails a mouse-based manipulation of 3D geometry, where users manipulate their designs through a collection of windows showing the geometry from various perspectives. This

¹INESC-ID, Instituto Superior Técnico, University of Lisbon

Corresponding author:

Renata Castelo-Branco

Email: renata.castelo.branco@tecnico.ulisboa.pt

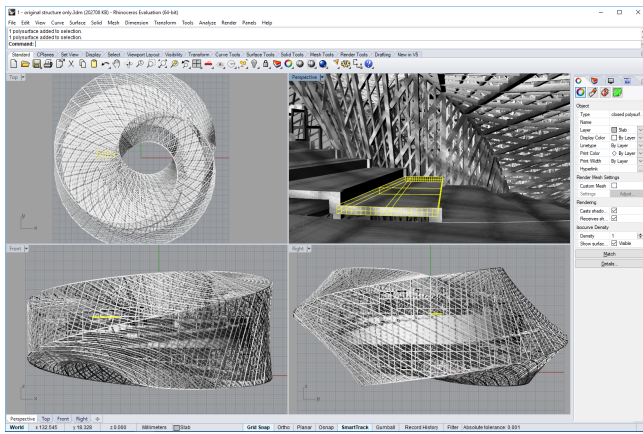


Figure 1. Rendered 3D model of the Astana National Library project in Rhinoceros 3D shown in the 4 default viewports.

system forces architects not only to combine the separate views to form a mental model of the entire scenario⁹, but also to scale it to real size in their imagination. Figure 1 illustrates this scenario for the case of the Astana National Library project. This building, originally designed by Bjarke Ingels Group in 2008, has the shape of a 3D Moebius strip, a rather complex form for one to understand through projections only.

Given that most digital design tools fail to provide the dimensionality required to assess the quality of a design solution^{10,11}, traditional architectural design processes tend to rely on the production of physical models. However, this is expensive and time-consuming.

As we previously established, despite all the advantages, the nature of AD hampers designers' ability to infer a building's concept just from looking at its program; particularly so for the levels of design complexity it promotes. Considering the scenario presented above, better visualization mechanisms are required to bring this representation method closer to the form-creation and manipulation mechanisms architects have for centuries used to design¹⁰.

Virtual Reality

Virtual Reality (VR) is a new interaction paradigm where users experience computer-generated worlds in a deeply immersed manner. In a Virtual Environment (VE), all the elements in the world are artificially created and the user is fully engaged in a virtual experience. Given its potential, VR presents a viable solution to the unnatural visualization problem of digital design tools. Three main strategies¹² were identified for the application of VR in the AEC industry:

- (i) When **designing** in a VE, the 3D medium can shape itself around the author in any scale, facilitating essential perceptions of solid, void, navigation, and function¹³, as well as motivating designers to engage in more creative¹⁴ and exploratory design actions¹⁵.
- (ii) **Communicating** with VR offers new possibilities for customer showcasing, by allowing users to walk inside the constructions as if they were already built. VR technology has also been used for user-centred design and analysis^{16,17}, by having users interact with the models in VR for occupancy or behaviors-related studies and validation^{18,19}.
- (iii) Regarding new **market opportunities**, we highlight the new remote collaboration methods allowed by VR technologies^{20,21}, e.g., multiple collaborators at different location in the globe can now be inside a virtual model of a building simultaneously interacting with the building and with each other.

Typical use of VR in the design process covers design assessment only, meaning the architect wears a Head-Mounted Display (HMD) only to evaluate the design solution, removing it afterwards to get back to the modelling tool in order to change the design. By coupling algorithmic approaches with VR, we can transform the design process in the VE into a real-time and interactive one, with no need for the architect to remove the HMD between design stages. For designer-client interaction, this coupling represents considerable time gains: if architects can make changes to the model in real time, while inside it with the client, they can accelerate the typical client-architect ideation process. In the context of AD, real-time and interactive processes imply the use Live Coding (LC).

Live Coding

LC is a creativity technique centred upon the writing of interactive programs on the fly²². LC is based on a real-time connection between program and result, thus requiring immediate feedback on program changes. Although this requirement was designed to address the timing needs of live performances, such as musical ones²³, it has the added benefit of allowing the programmer to easily understand the impact of program changes.

In the case of AD, immediate feedback requires the model to be recomputed for each change applied to the algorithmic description. This is not an entirely trivial problem to solve, since building designs quickly become arbitrarily complex.

From the panoply of AD tools available in the market, we highlight two that offer LC capabilities²²: Grasshopper, a visual programming environment, and Luna Moth²⁴, a web-based textual programming tool. Both tools offer the liveness aspect that makes the programming task more intuitive. These two tools deal with the immediate feedback vs. complexity issue in very different ways. Luna Moth scales to complex designs by taking advantage of the native execution speed allowed by the compilation of textual programs, while Grasshopper offers users the possibility to deactivate the feedback at large scales.

The creation of a workflow for programming in VR has been attempted by Elliott et al.²⁵ with the RiftSketch project, and by Robert Krahn with CodeChisel3D.* Both tools offer a LC environment built for VR, with text editors floating in the scene for users to code in. Nevertheless, both solutions were only tested with simple graphical models and they were not applied in an architectural context.

Using VPLs, tailored solutions for architecture have also been developed. For instance, Coppens et al.²⁶ and Hawton et al.²⁷ both implemented visual-based AD solutions for VR. However, they only allow users to calibrate parameters in VR using sliders, meaning it is not possible to apply changes to the algorithm itself (the nodes and wires, in this case). This limitation excludes the named solutions from the LC classification.

Immersive Algorithmic Design

Despite being applicable in many different contexts, Live Coding in Virtual Reality (LCVR) becomes particularly useful for architecture, allowing practitioners to live code their models in an immersive AD process. Considering this, we have previously proposed a design workflow that entails the integration of LCVR with AD²⁸. The work here presented extends our previous proposal, firstly, by elaborating on the workflow's building blocks and implementation details and, secondly, by adding additional features to the implementation that make LCVR applicable in a broader range of scenarios.

In the LCVR workflow, the architect uses a HMD to become immersed in the VE, where both the algorithmic description and the generated model are presented. From the VE, the architect can then apply modifications to the AD description, while the resulting model is concurrently updated in accordance to the changes made.

Figure 2 presents one of the possible implementations of LCVR being used in the Astana National Library model. Three design variations, namely in what regards the torsion

of the facade, are presented from different points of view, representing places in the model where the architects can be immersed, evaluating the impact of the changes applied.

For this workflow to take place, a sufficiently performant connection is required between an AD and a VR tool. Game Engines (GEs) provide a solution to this problem.

Integrated Approach

In the past, an integrated approach to AD³ was proposed, which entailed the creation of a single algorithmic description capable of generating equivalent models in various tools. These tools are referred to as backends of the AD tool and they are chosen depending on which paradigm the architect may find most beneficial for the task at hand at any given stage of the design process.

For instance, we could use Computer-Aided Design (CAD) tools for concept and form experimentation; Building Information Modelling (BIM) tools for more detailed stages where construction information is required; analysis tools to evaluate the design's performance; optimization tools to optimize designs based on the performance analysis; and, finally, GEs for fast visualization and interaction with the models in near real-time render quality.

As we saw before, the advantages VR brings to the design process seem to be widespread throughout the various design stages¹². Hence, we believe architects should be free to determine at which stage the LCVR workflow best fits their own design process, and if it should or should not entirely replace programming in (actual) reality.

Figure 3 presents a scheme of the integrated approach extended to accommodate the LCVR workflow. On the top left corner, we notice the typical AD workflow, where the architect interacts directly with the AD tool from the computer's desktop, coding in the Interactive Development Environment (IDE), i.e., the code editor to which the tool is coupled.

Live Coding Workflow

On the bottom left corner of Figure 3, we can see a different mode of interaction with the AD tool, from the VE: LCVR. The LCVR workflow is anchored primarily on GE backends. GEs are optimized visualization tools that allow for faster generation of models and low latency in live model manipulation and navigation. This makes them good

*<http://robert.krahn/past-projects/live-programming-with-three-and-webvr.html>

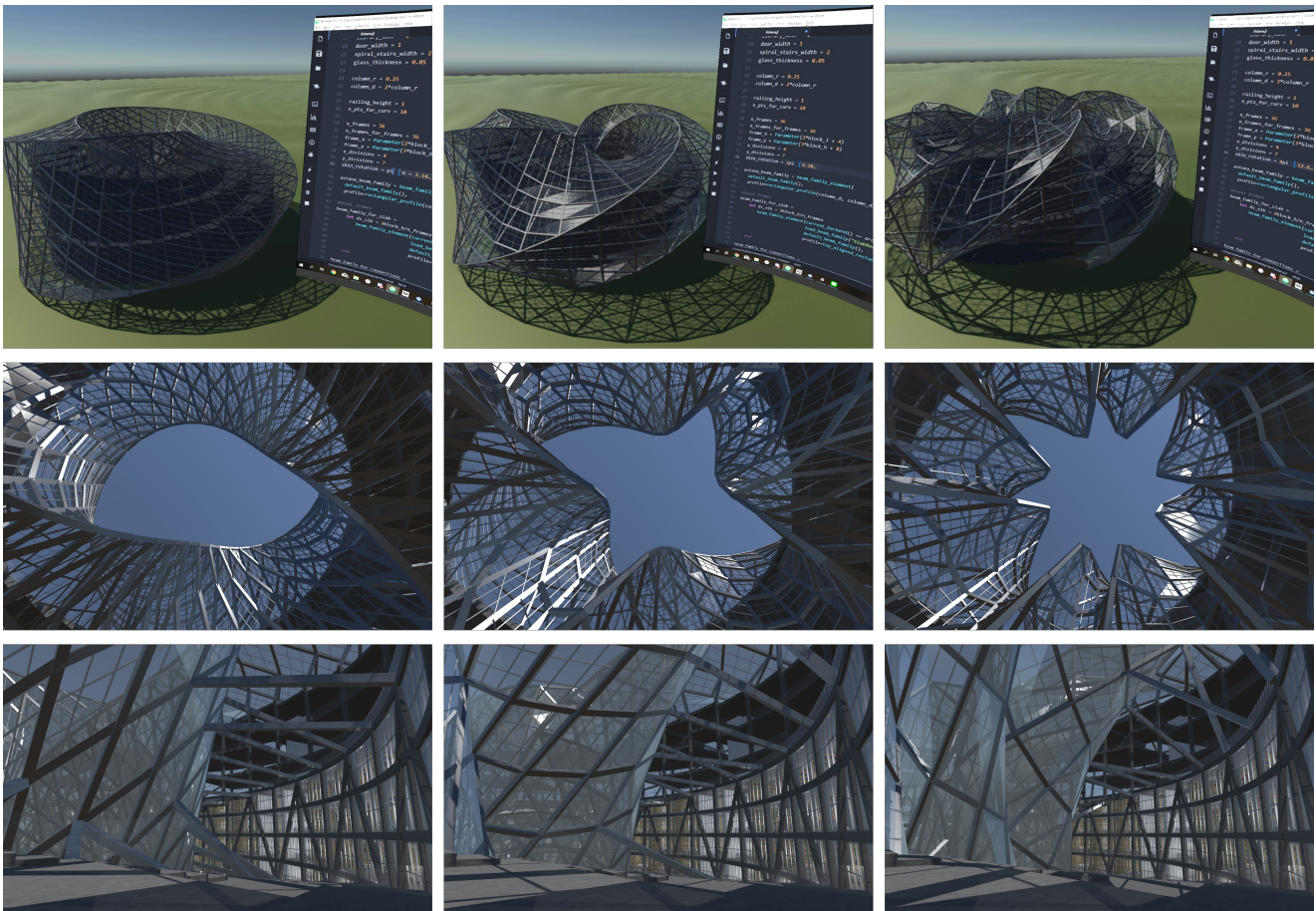


Figure 2. Astana National Library being live coded in VR: 3 variations (left to right) seen from 3 different angles (top to bottom).

ALGORITHMIC DESIGN

INTEGRATED TOOLS

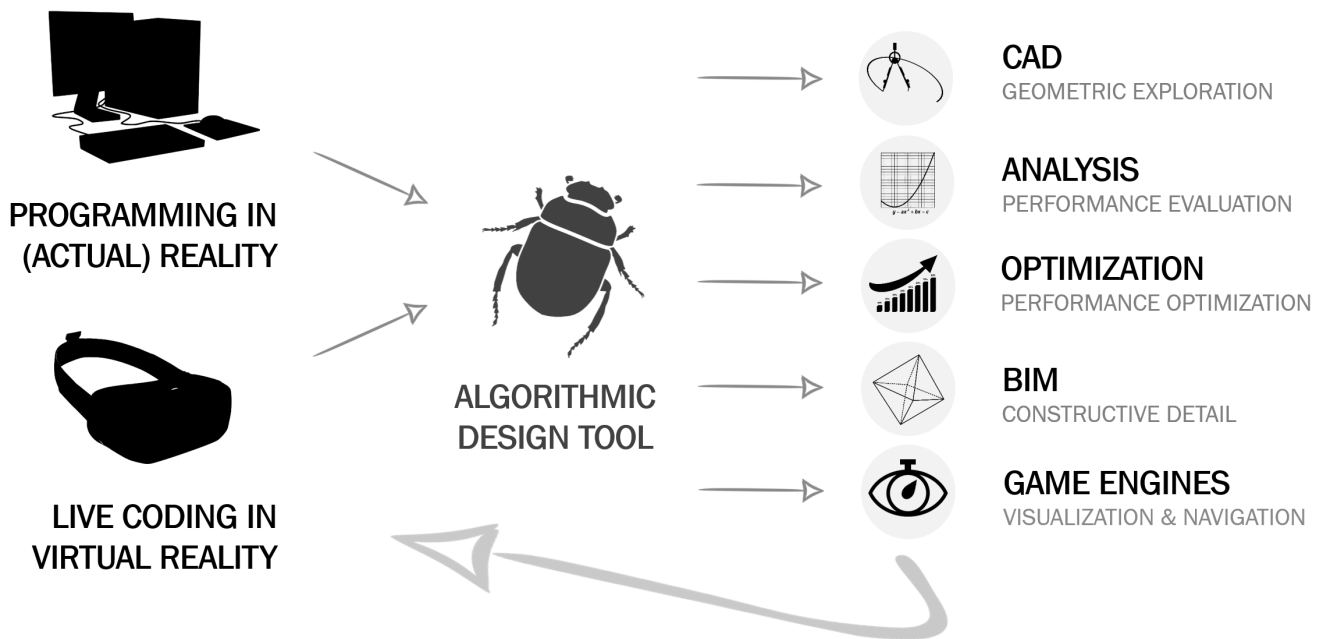


Figure 3. Integrated AD and LCVR workflows.

candidates for a solution that relies on real-time code-model interactivity.

This workflow allows the architect to continue developing the algorithmic description of the design, with all the advantages of typical AD processes. This means the geometry can still be generated in alternative backends, however, when using the GE backend, architects can also control the process inside the VE. Furthermore, LCVR implies that this process is done live, meaning the building changes around architects as they code it on site.

There is, nonetheless, a setback to consider in this scenario. Despite the fast response guaranteed by GEs, the capacity for real-time feedback will always be conditioned by the model's complexity. Architectural 3D models tend to rapidly escalate in complexity, which means large-scale projects will always cause short time lapses between generations of model iterations.

Programming Paradigm

As discussed previously, two competing programming paradigms exist in the field. As both offer very different sets of advantages to AD, we chose to test our proposal with both visual and textual programming. VPLs are generally more appealing to the architectural community, given their user-friendliness and smooth learning curve. On the other hand, TPLs offer more expressive power, flexibility, and efficiency²⁹, which makes them a more appealing option when developing larger AD projects.

Regarding LCVR, it matters not only the expressiveness and scalability of the paradigm, but also the code manipulation mechanisms available in each case. Since users will be coding in a VE, the interaction with the programming environment, or IDE, will necessarily be rather different from the one they have experienced in their computers.

In this regard, we argue that programming in VPLs is likely to render better results when coding in VR, as they require less textual input. While VPLs typically rely on dragging mechanisms for manipulating components, TPLs depend on textual input usually provided via keyboard. LCVR with TPLs will thus require the use of extra equipment for the typing task. We will delve deeper into this topic in the following sections.

Implementation

For the implementation of an integrated solution, we require an AD tool capable of translating the algorithmic description into operations recognized by different backends. There are several tools that allow for this sort of portability. For the

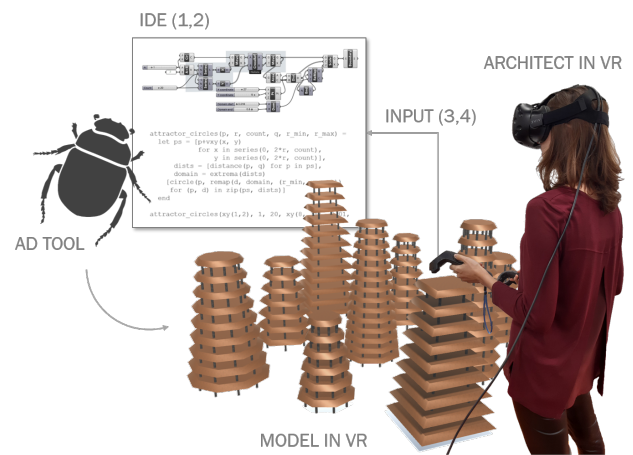








Figure 4. LCVR workflow: the architect in VR interacts with the IDE in order to change the model's algorithmic description. As the changes are being applied to the code, the AD tool regenerates the model in the VE where the architect is.

evaluation of our proposal we will focus on two of them: Grasshopper, representing visual programming, and Khepri, for textual programming. The Khepri AD tool, currently available for use within the Atom IDE and with the Julia programming language, offers a direct connection to a GE backend: Unity³⁰. In order to compare both paradigms in the context of LCVR, we developed a Khepri-based Grasshopper plug-in capable of communicating with Khepri's backends.

This means the Khepri AD tool is, in both cases, responsible for generating the coded geometry. Figure 4 presents the proposed LCVR workflow. While immersed in VR, architects modify their AD programs through the chosen IDE, which is being projected onto the VE. The IDE gives them access to their programs (in this case, visual programs written in Grasshopper or textual programs written in Julia). The Khepri AD tool, in turn, converts the given instructions to operations recognized by the backend, Unity, updating the VR model around the architects in real-time.

In order to implement this workflow, three main features must be considered: (1) the projection of the IDE onto the VE for the architect to access the program; the input mechanisms for the architect to change the program, which can be (2) code manipulation mechanisms, or (3) parameter manipulation mechanisms only, as an alternative for faster and more direct changes to the code. This section contains an overview of currently available solutions for the presented issues, grounded on the experiments we made for each of them.

In order to provide a critical analysis and comparison of the various options, we developed a simpler case study: a random pagoda city exercise, whose limited complexity allowed us, at this stage, to experiment with the various

	TEXTUAL PROGRAMMING	VISUAL PROGRAMMING
FLAT ELEMENTS	 TAILORED TEXTBOXES SHOWCASING ENTIRE PROGRAM	 COMPONENTS PROJECTED ONTO THE VIRTUAL ENVIRONMENT
3D ELEMENTS	 SELECTED TEXT PARTS PROJECTED ONTO THE VIRTUAL ENVIRONMENT	 COMPONENTS PROJECTED ONTO THE VIRTUAL ENVIRONMENT
3D MEDIUM 2D CODE	 MIRRORING THE USER'S DESKTOP	 MIRRORING THE USER'S DESKTOP



 EXPLORED
  IMPLEMENTED

Figure 5. IDE options table for both the explored ideas and implemented solutions.

implementation possibilities as well as test different input mechanisms. An equivalent representation of the pagoda city was programmed in both a VPL - Grasshopper - and a TPL - Julia. However, and since this work is clearly exploratory, some of the solutions we discuss regarding IDE projection onto the VE are presented as mockups only, all of which are properly identified as such.

Integrated Development Environment

In order to live code in VR, the architect requires a platform in which to program from the VE. For this problem, we considered four possible solutions: (1) having textual programs available for manipulation in a tailored textbox, (2) having visual components projected onto the VE, (3) having selected parts of textual programs projected onto the VE as 3D entities, and (4) mirroring the user's desktop in the VE. Figure 5 organizes these options in terms of their applicability to the two programming paradigms, and it identifies the option we chose to implement in the end.

Tailored textboxes is a solution most fit for TPLs and implies having the code showcased in a textbox on screen for the user to edit. In this case, the text moves along with the user's gaze, as it belongs not to the scene but to the user's own viewport. This means users do not need to (re)place the workstation each time they move in the VE (Figure 6-A).

However, there are several disadvantages. Primarily, the code is likely to become visual clutter as it blocks part of the scene. Secondly, IDEs offer debugging, syntax highlight, and other features, which represent great gains in coding efficiency. With a textbox approach, users lose the aid provided by their IDEs of choice during the coding task.

Having VPL components projected onto the VE has two possible development paths: the first option would be, much like the previous solution, to showcase the visual program in front of the scene, moving along with the user's viewport

(Figure 6-B1); the second option would be to have the visual components generated as 3D elements, which could be manipulated like any other object in the scene (Figure 6-B2).

The first option would likely be more intrusive, although the user might choose to turn the programming layer on and off to eliminate the partial scene clutter from time to time. On the other hand, in the second case, the code does not accompany the user's movement automatically, which may limit navigation while coding. Furthermore, having 3D components floating in the scene might lead to confusing situations where code and resulting geometry interfere with each other, thus becoming even harder to understand and manipulate.

Having selected TPL programs projected onto the VE as 3D entities is another option. In this case, users are expected to select an object they wish to modify. The program recognizes the parcel of code that generated this object, or set of objects, and projects it onto the VE, next to the referred objects. Within the VE, the user can then change that parcel of code. Figure 6-C presents the corresponding mockup for this solution.

This approach, as opposed to having the entire program showcased at once, makes better use of the 3D space, since we can distribute the selected parts of the program on the scene along with the corresponding geometry. It is also more succinct, which might help less experienced programmers orient themselves in the code.

Nevertheless, we can also foresee obstruction issues. In this scenario, code would be popping up arbitrarily next to the selected part of the model, unless we devised an algorithm capable of calculating the ideal code position according to both user's sight cone and neighbouring geometry. Even then, readability would not be assured on account of the background.

Mirroring the user's desktop in the VE could work for either programming paradigm. Since we are viewing, inside the VE, what we would view outside it, we can essentially rely on the same IDE we would use in a traditional coding workflow. Besides being the easiest to implement, this solution also offers users entire control over their programs in the VE, meaning they can see and access anything they would if they were coding on their desktops, plus the added advantages of using the complete IDE.

Like its counterparts, this one also presents some level of intrusiveness, in the sense that the mirrored screen constitutes a partial visual blocker to the scene. Nevertheless,

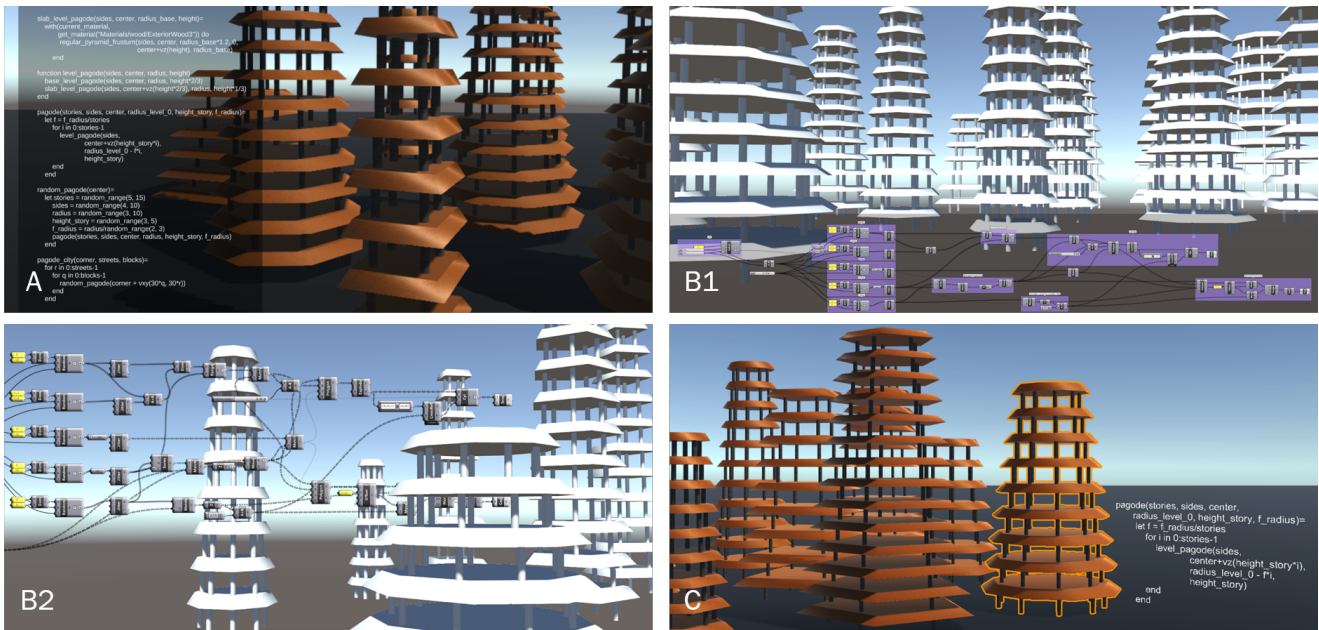


Figure 6. IDE option mockups: A - tailored textboxes; B - visual components as (1) 2D or (2) 3D entities in the VE; C - selected text parts projected onto the VE.

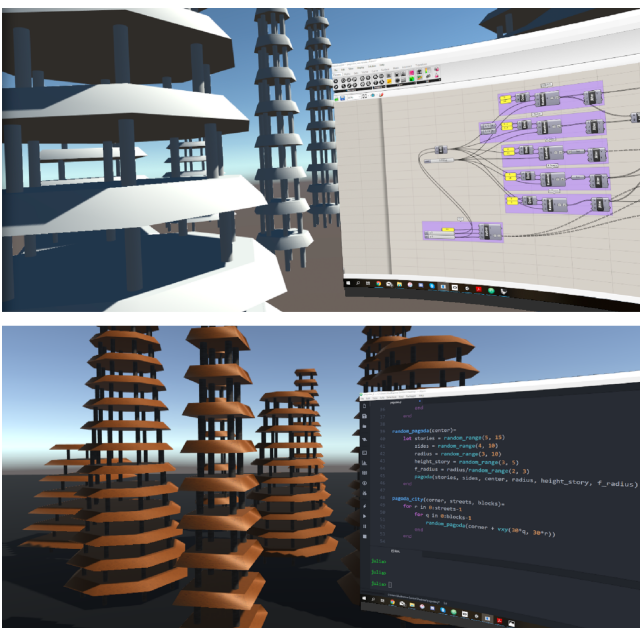


Figure 7. Mirroring the user's desktop IDE option. Grasshopper environment on top and the Atom IDE showcasing a program in the Julia language on the bottom.

the desktop can be moved around or removed entirely with a click of a button.

Considering the discussed advantages, as well as the small amount of obstacles to overcome in its implementation, we have settled on this solution. Figure 7 presents the LCVR scenario for this option in both a visual and textual programming context.

Code Manipulation

In order to program in VR, one must be able to type. This is a more notorious necessity when coding in TPLs, since VPLs mostly rely on dragging and dropping components and wires - a workflow assured by the gripping mechanisms provided by VR technology. However, even using VPLs, we have to input numbers for parameters, variables, etc.

To deal with textual input, we considered four currently available solutions: handwriting, voice input, typing on a virtual keyboard, and typing on a physical keyboard.

Hand-written code recognition presumes the existence of large enough databases of hand-written code and the corresponding typed code for any given tool to be trained with accuracy. For this solution to be implemented, the scale of the written characters in VR must also be considered. Humans are considerably faster when writing symbols in smaller scales (e.g., when writing on paper, as opposed to writing on a whiteboard). Handwriting code in VR, however, as hardware stands today, would have to rely on larger displacements of the writing instruments for the sensors to detect the motion, which might ultimately defeat the performance purpose.

Voice input would probably be the most comfortable option for the user. In the context of VR, vocal programming would allow users to code hands-free, thus requiring no additional equipment other than a microphone, which, in most cases, is already present in VR headsets. Nevertheless, we can foresee a series of obstacles as well. Primarily, current technology



Figure 8. Tested typing options with corresponding visual feedback in VR below: virtual keyboard - wearing the control handles using (1a) laser tags, (1b) index fingers only, or (1f) leap motion technology; and (2) occluded physical keyboard, with a corresponding virtual keyboard with highlighted keys.

has low voice recognition accuracy in loud environments, which means expensive equipment might be required when working collaboratively in VR or showcasing projects to clients. Secondly, and more specific to the problem at hand, dictating coding commands is nothing like dictating text in natural language and, thus, requires training.

Virtual keyboards, along with physical ones, represent the more conservative solution, as they try to mimic the coding workflow in (actual) reality. Currently available solutions for the use of virtual keyboards include using regular VR controllers to (a) point at keys with laser tags, (b) touch the virtual keyboard with the index fingers only, or (c) drum the keys. Our own experimentation (visible in Figure 8) included the use of solutions (a) and (b), which proved to be very slow. Solution (c) is considered a more efficient technique, yet it entails a steeper learning curve.

Still on virtual keyboards, it is also possible to type via (d) gaze input, which is also slow, (e) using wearable finger tracking hardware, and (f) leap motion technology to track the movement of multiple fingers. Option (e) is rather intrusive, on account of the wearables required, which not only take time to mount and dismount, but may also be a burden. From this group we only tested solution (f), leap motion technology, which proved to have good precision levels for gesture recognition, but largely failed in tracking the motion of each finger over a keyboard, since they tend to occlude each other within the range of the sensor. Because of this, we rejected this option prior to implementation in the LCVR workflow. Image (1f) in Figure 8 corresponds to

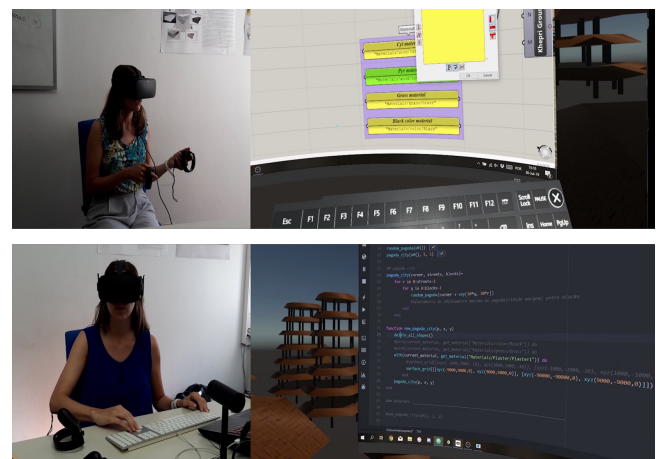


Figure 9. LCVR use case: visual paradigm on top and textual paradigm on the bottom.

a prototype of the ideal solution to be implemented if/when this technology achieves higher levels of accuracy.

The use-case of option (1a) for the pagoda exercise can be found in Figure 9. The top image shows an architect using controllers to point at keys with laser tags in order to modify the version of the pagoda program developed in Grasshopper.

Physical keyboards have already proven to beat the typing performance of virtual keyboards³¹. However, it has been shown that occluded keyboards cause significantly more typing errors³². Our findings concur with the literature review: the occluded physical keyboard yielded good results with experienced typists, but not as good with less trained ones, who consistently mistyped commands and frequently lost their track on the keyboard. We also tried showcasing a

responsive virtual keyboard in the scene, which highlighted the pressed keys to provide the user with key-striking feedback. This slightly improved the performance of the second test group. Surprisingly, though, we found most of them simply preferred to peek the real keyboard through the nose hole of the glasses (a workflow made possible due to the characteristics of one of the headsets used in the experiments).

Finally, it is also possible to provide users with hand-position feedback using suggestions (e) and (f) described above, or using an inlay webcam recording the user's hands on the physical keyboard and projecting the image in VR. The inlay webcam approach has proven to guarantee the least error rates³³, although the time delay on the projected image is substantial in most cases and cannot be overlooked.

In conclusion, either of the presented solutions is far from ideal. Typing on a normal keyboard outside the VE beats the performance of any of these methods by a large margin. This does not mean they are hopeless; we can always count on technological evolution to provide innovative solutions and to improve existing ones. Naturally, this topic is more concerning in the context of TPLs, which require much more typing than VPLs. This leads us to conclude that for LCVR in VPLs there is barely any need for extra equipment, as the typing solutions allowed by the control handles suffice for the task at hand.

Figure 9 shows, on the bottom, the use case of the occluded physical keyboard option for the pagoda exercise. The image shows the architect typing in changes to the textual version of the pagoda program developed in the Julia programming language.

Parameter Manipulation

The mechanisms described above allow architects to inspect and modify, from the VE, the entirety of their AD program. However, for simple and/or localized changes, such as modifying object placement, or object families' dimensions or materials, more elementary interaction mechanisms may suffice. As stated previously, even the most performative typing solution is still no match for the equivalent mechanisms outside VR, hence, an intermediate interface that casts VPL's ever so typical parameter manipulation mechanisms might be useful, specially for the textual case.

The use of parameter manipulation mechanisms like sliders and toggles in the textual paradigm is not new. Luna Moth²⁴, for instance, allows users to manipulate numerical values in the program using mouse dragging actions. Within the visual programming paradigm, there are also solutions with this outline for VR as we have seen previously: Coppens

et al.²⁶ and Hawton et al.²⁷ present intermediate interfaces for parameter manipulation.

For the LCVR workflow we developed a 2D interface that pops up in the VE with sliders and buttons for the user to change the model within the boundaries of the parameters presented. These changes are automatically translated into the corresponding code in the program that generated the model. Figure 10 presents an example of such an interaction: by selecting an object in the scene, the user is presented with a menu containing sliders that control the parameters of the function that generated that object. In this case, the pagoda function was called, and the user changed the number of stories. In this scenario, only controllers are required, since there is no typing involved.

The choices presented in the menu can also be defined by the architects themselves when elaborating the algorithmic description. Choosing what options to present to users may prove helpful in showcasing sessions. Clients or co-workers may, hence, get deeply involved in the modification of the design as well, with no need of prior programming knowledge and without incurring in the risk of introducing bugs in the program or deviating from the architect's original design intention.

For architects themselves working with LCVR, this middle-ground interface allows them to play in a more user-friendly manner with common changes to function parameters or object families, which are subjects of frequent modifications in an AD workflow. The user may desire to maintain the projected IDE in the VE and use this as a complement, or temporarily substitute one for the other.

Conclusion

In this paper we proposed Live Coding in Virtual Reality (LCVR), a design approach that allows architects to benefit from the advantages of Virtual Reality (VR) within an Algorithmic Design (AD) workflow. LCVR offers a different mode of interaction with the AD tool: by having the program displayed alongside the generated model in the Virtual Environment (VE), architects can change the program and, therefore, the model, without leaving the VE.

The work here presented extends our preliminary research on the topic²⁸ by detailing the implementation issues and by addressing one important limitation: the ability to make simple localized changes to the program in VR in a more user-friendly way that does not require typing. Section **Parameter Manipulation** described this feature in detail.

We presented and discussed the implementation challenges of the LCVR approach, as well as the solutions we

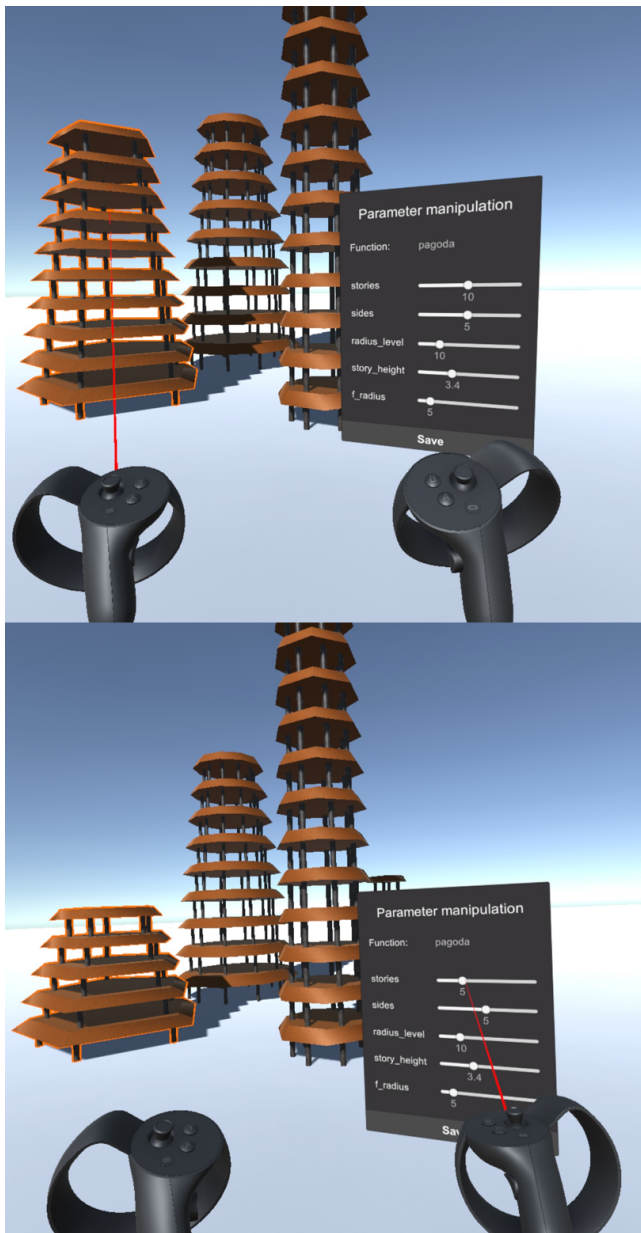


Figure 10. Intermediate interface example: changing the number of stories parameter.

chose to pursue. We explored this approach with both Visual Programming Languages (VPLs) and Textual Programming Languages (TPLs), comparing different ways of projecting a programming environment onto the VE for either case. We settled on the mirrored desktop solution, however, in the future we intend to develop other approaches.

Two different sets of mechanisms for architects to change their programs were discussed: code manipulation and parameter manipulation. The former allows full control over the program while immersed in the resulting model. The requirements of this task in the case of TPLs have proven to be more troublesome. The latter provides a better solution for the case of faster and more direct changes to the program, as is the case of number, size or material choices.

We verified that LCVR tends to render more user-friendly interactions with the use of VPLs. However, this paradigm does not scale well with complexity, which hinders the use of AD for more complex design problems. The use of an intermediate graphical user-interface helps bring TPLs closer to this interaction paradigm as well, in the case of simpler code modifications.

A simple case study was used to explore the proposed implementation. However, and as suggested in section **Immersive Algorithmic Design**, the workflow is meant to be applied precisely to the large-scale or complex architectural projects allowed by AD.

As future work we plan on taking advantage of the multiple backends allowed by the integrated approach. For instance, we can have analysis results, such as radiation colour maps and deformation graphs, being presented to the user in VR as well. This workflow can also occur live, with the analysis tools running simulations in the background, while the user is immersed in VR.

Acknowledgements

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UIDB/50021/2020 and PTDC/ART-DAQ/31061/2017.

References

1. Caetano I, Santos L and Leitão A. Computational design in architecture: Defining parametric, generative, and algorithmic design. *Frontiers of Architectural Research* 2020; 9(2): 287–300. DOI:10.1016/j.foar.2019.12.008.
2. Burry M (ed.) *Scripting Cultures: Architectural Design and Programming*. Architectural Design Primer, John Wiley & Sons, Inc., 2013. ISBN 9781119979289. DOI:10.1002/9781118670538.
3. Castelo-Branco R and Leitão A. Integrated algorithmic design: A single-script approach for multiple design tasks. In Fioravanti A, Cursi S, Elahmar S et al. (eds.) *ShoCK: Proceedings of the 35th Education and research in Computer Aided Architectural Design in Europe (eCAADe) Conference*, volume 1. Faculty of Civil and Industrial Engineering, Sapienza University of Rome, Rome, Italy, pp. 729–738.
4. Peters B. Computation works: The building of algorithmic thought. *Architectural Design* 2013; 83(2): 8–15. DOI: 10.1002/ad.1545.
5. Kelleher C and Pausch R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys* 2005; 37(2): 83–137. DOI:10.1145/1089733.1089734.

6. Celani G and Vaz C. Cad scripting and visual programming languages for implementing computational design concepts: A comparison from a pedagogical point of view. *International Journal of Architectural Computing* 2012; 10(1): 121–137. DOI:10.1260/1478-0771.10.1.121.
7. Janssen P. Visual Dataflow Modelling - Some thoughts on complexity. In Thompson EM (ed.) *Fusion: Proceedings of the 32nd Education and research in Computer Aided Architectural Design in Europe (eCAADe) Conference*, volume 2. Faculty of Engineering and Environment, Newcastle upon Tyne, England, UK, pp. 547–556.
8. Leitão A, Lopes J and Santos L. Programming languages for generative design: A comparative study. *International Journal of Architectural Computing* 2012; 10(1): 139–162. DOI:10.1260/1478-0771.10.1.139.
9. Gobetti E and Scateni R. Virtual reality: past, present and future. *Studies in health technology and informatics* 1998; 58: 3–20. DOI:10.3233/978-1-60750-902-8-3.
10. Stavrić M, Šidanin P and Tepavčević B. *Architectural Scale Models in the Digital Age: design, representation and manufacturing*. Springer, Vienna, 2013. ISBN 9783990435274. DOI:https://doi.org/10.1007/978-3-7091-1448-3.
11. Dunn N. *Architectural Modelmaking*. Laurence King Publishing, 2014. ISBN 9781780671727.
12. Whyte J. Industrial applications of virtual reality in architecture and construction. *Journal of Information Technology in Construction (ITcon)* 2003; 8(4): 43–50.
13. Portman M, Natapov A and Fisher-Gewirtzman D. To go where no man has gone before: Virtual reality in architecture, landscape architecture and environmental planning. *Computers, Environment and Urban Systems* 2015; 54: 376–384.
14. Schnabel MA. The immersive virtual environment design studio. In Wang X and Tsai J (eds.) *Collaborative Design in Virtual Environments*. Springer, Dordrecht, 2011. pp. 177–191.
15. Gu N, Kim MJ and Maher ML. Technological advancements in synchronous collaboration: The effect of 3D virtual worlds and tangible user interfaces on architectural design. *Automation in Construction* 2011; 20(3): 270–278. DOI:https://doi.org/10.1016/j.autcon.2010.10.004.
16. Moloney J, Globa A, Wang R et al. Pre-occupancy evaluation tools (p-oet) for early feasibility design stages using virtual and augmented reality technology. In Rajagopalan P and Andamon MM (eds.) *Engaging architectural science - meeting the challenges of higher density: Proceedings of the 52nd International Conference of the Architectural Science Association (ANZAScA)*. Melbourne, Australia, pp. 717–726.
17. Heydarian A, Pantazis E, Wang A et al. Towards user centered building design: Identifying end-user lighting preferences via immersive virtual environments. *Automation in Construction* 2017; 81: 56–66.
18. Paes D, Arantes E and Irizarry J. Immersive environment for improving the understanding of architectural 3d models: Comparing user spatial perception between immersive and traditional virtual reality systems. *Automation in Construction* 2017; 84: 292–303.
19. Kuliga S, Thrash T, Dalton RC et al. Virtual reality as an empirical research tool - exploring user experience in a real building and a corresponding virtual model. *Computers, Environment and Urban Systems* 2015; 54: 363–375.
20. Koutsabasis P, Vosinakis S, Malisova K et al. On the value of virtual worlds for collaborative design. *Design Studies* 2012; 33(4): 357–390.
21. Dorta T, Lesage A, Pérez E et al. Signs of collaborative ideation and the hybrid ideation space. In Taura T and Nagai Y (eds.) *Design Creativity*. London: Springer, 2011. pp. 199–206. DOI: https://doi.org/10.1007/978-0-85729-224-7_26.
22. Rein P, Ramson S, Lincke J et al. Exploratory and live, programming and coding: A literature study comparing perspectives on liveness. *Programming Journal* 2018; 3(1): 3.
23. Sorensen A and Gardner H. Programming with time: Cyber-physical programming with impromptu. *ACM SIGPLAN Notices* 2010; 45(10): 822–834. DOI:10.1145/1932682.1869526.
24. Alfaiate P, Caetano I and Leitão A. Luna Moth: Supporting creativity in the cloud. In Anzalone P, Signore M and Wit AJ (eds.) *Disciplines & Disruption: Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*. Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, pp. 72–81.
25. Elliott A, Peiris B and Parnin C. Virtual reality in software engineering: Affordances, applications, and challenges. In Kellenberger P (ed.) *Proceedings of the 37th International Conference on Software Engineering*, volume 2. Florence, Italy: IEEE Computer Society, pp. 547–550.
26. Coppens A, Mens T and Gallas MA. Parametric modelling within immersive environments: Building a bridge between existing tools and virtual reality headsets. In Kepczynska-Walczak A and Bialkowski S (eds.) *Computing for a Better Tomorrow: Proceedings of the 36th Education and research in Computer Aided Architectural Design in Europe (eCAADe) Conference*. Faculty of Civil Engineering, Architecture and Environmental Engineering, Lodz, Poland, pp. 711–716.
27. Hawton D, Cooper-Wooley B, Odolphi J et al. Shared immersive environments for parametric model manipulation - evaluating a workflow for parametric model manipulation from within immersive virtual environments. In Fukuda T, Huang W, Janssen P et al. (eds.) *Learning, Prototyping and Adapting: Proceedings of the 23rd International Conference*

- on *Computer-Aided Architectural Design Research in Asia (CAADRIA)*. Beijing, China, pp. 483–492.
28. Castelo-Branco R, Leitão A and Santos G. Immersive algorithmic design: Live coding in virtual reality. In Sousa JP, Henriques GC and Xavier JP (eds.) *Architecture in the Age of the 4th Industrial Revolution: Proceedings of the 37th Education and research in Computer Aided Architectural Design in Europe (eCAADe) Conference*, volume 2. University of Porto, Porto, Portugal, pp. 455–464.
 29. Sammer MJ, Leitão A and Caetano I. From visual input to visual output in textual programming. In Haeusler MH, Schnabel MA and Fukuda T (eds.) *Intelligent & Informed: Proceedings of the 24th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*, volume 1. Victoria University of Wellington, Wellington, New Zealand, pp. 645–654.
 30. Leitão A, Castelo-Branco R and Santos G. Game of renders: The use of game engines for architectural visualization. In Haeusler MH, Schnabel MA and Fukuda T (eds.) *Intelligent & Informed: Proceedings of the 24th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*, volume 1. Victoria University of Wellington, Wellington, New Zealand, pp. 655–664.
 31. Grubert J, Witzani L, Ofek E et al. Text entry in immersive head-mounted display-based virtual reality using standard keyboards. In Kiyokawa K, Steinicke F, Thomas B et al. (eds.) *25th IEEE Conference on Virtual Reality and 3D User Interfaces*. Reutlingen, Germany, pp. 159–166. DOI:<https://doi.org/10.1109/VR.2018.8446059>.
 32. Walker J, Li B, Vertanen K et al. Efficient typing on a visually occluded physical keyboard. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17, New York, NY, USA: Association for Computing Machinery. ISBN 9781450346559, pp. 5457–5461. DOI: 10.1145/3025453.3025783.
 33. Grubert J, Witzani L, Ofek E et al. Effects of hand representations for typing in virtual reality. In Kiyokawa K, Steinicke F, Thomas B et al. (eds.) *25th IEEE Conference on Virtual Reality and 3D User Interfaces*. Reutlingen, Germany, pp. 151–158. DOI:10.1109/VR.2018.8446250.