# Immersive Algorithmic Design

## *Live Coding in Virtual Reality*

*Renata Castelo-Branco[1], António Leitão[2], Guilherme Santos[3]*
*[1,2,3]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa*
*[1,2,3]{renata.castelo.branco|antonio.menezes.leitao|guilherme.j.santos}@tecnico.ulisboa.pt*

*As many other areas of human activity, the architectural design process has been recently shaken by Virtual Reality (VR), as it offers new ways to experience and communicate architectural space. In this paper we propose Live Coding in Virtual Reality (LCVR), a design approach that allows architects to benefit from the advantages of VR within an algorithmic design workflow. LCVR integrates a live coding solution, where the architect programs his design intent and immediately receives feedback on the changes applied to the program; and VR, which means this workflow takes place inside the virtual environment, where the architect is immersed in the model that results from the program he is concurrently updating from inside VR. In this paper we discuss the possible impacts of such an approach, as well as the most pressing implementation issues. 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*

## VIRTUAL REALITY IN ARCHITECTURE

Virtual Reality (VR) has come a long way from the moment Ivan Sutherland, one of the godfathers of computer graphics, created the first Head Mounted Display (HMD) (Steinicke 2016). In 1965, he envisioned the ultimate display (Sutherland 1965), a reality within which the computer controls the existence of matter itself. Although we are still far from achieving this radical vision, VR technologies can already provide enough realism and immersiveness to make it an appealing tool for various professions, namely architecture.

Architects currently use a variety of instruments and tools to represent their designs. The choice of representation used not only has a great influence on the design process itself but is also one of the decisive factors for clients' approval. Furthermore, with the emergence of the Building Information Modeling (BIM) paradigm, the Architecture, Engineering and Construction (AEC) industry tends to integrate more information into the digital models and engage in more collaborative interactions around these models (Wang and Schnabel 2008; Yan et al. 2011). Such workflows have an imperative need for adequate and efficient representation mechanisms for both experts and non-experts. Whyte (2003) identified three main strategies for the application of VR in the AEC industry, which we unroll in the following paragraphs.

The first strategy, **designing** in a Virtual Environment (VE), allows the architect to interact with his creation in a deeply involved manner. The three-dimensional medium can shape itself around the author in any scale, facilitating essential perceptions of solid, void, navigation, and function (Wang and Schnabel 2008; Portman et al. 2015), as well as motivating designers to engage in more creative (Schnabel 2011) and exploratory design actions (Gu et al. 2011).

**Communicating** a design to a client has for long been a challenging step, since both the funding and the general course of the project typically depend on these moments. For that matter, VR offers new possibilities for customer interface, as it allows users to walk inside the constructions as if they were already built. In fact, there are now tools and companies offering this sort of services, such as Sentio VR, Enscape, REinVR, and IrisVR. Studies have also been made on how to use VR technology for user-centered design and analysis, (Moloney et al. 2018; Heydarian et al. 2017). However, most of this research involves showcasing final or mockup models with which users can interact for occupancy or behavior-related studies and validation (Paes et al. 2017; Kuliga et al. 2015), which means VR is being used as a communication medium only.

Regarding new **market opportunities**, we highlight the new collaboration methods allowed by VR technologies (Koutsabasis et al. 2012). More engaging experiences are now possible for remote participants in the same VE (Dorta et al. 2010), e.g., multiple collaborators can be inside a virtual model of a building simultaneously interacting with the building and with each other.

## ALGORITHMIC DESIGN

The issue with the approaches presented so far is the fact that most revolve around navigating static building models and allow only small, localized, manual changes to those models (de Klerk et al. 2019).

To overcome these limitations, we propose the integration of VR with Algorithmic Design (AD) strategies. AD entails the creation of algorithmic descriptions of the architects' design intent. These descriptions, shaped in the form of a computer program, instruct the machine to perform specific modeling operations, which will culminate in a digital model of the design.

This approach to design allows architects to relegate the modeling task to the computer, thus saving time and money through the automation of repetitive, time-consuming, and error-prone tasks. For the exact same reasons, AD also enables the creation of more diverse and complex design solutions.

Given these advantages, the goal, then, is to support AD in the VE. In VR, the use of algorithmic approaches means we can transform the design process in the VE into a real-time and interactive one, instead of a concept assessment period only, after which the architect must remove the HMD and get back to the modeling tool in order to change the model in due time. Furthermore, for designer-client interaction, it represents considerable time gains: if the architect can make the required changes to the model in real time, while inside it with the client, he can accelerate the typical client/architect ideation process.

## LIVE PROGRAMMING & LIVE CODING

From the panoply of AD tools available in the market, we highlight two that offer Live Programming (LP) capabilities (Rein et al. 2018): Grasshopper, a visual programming environment, and Luna Moth (Alfaiate et al. 2017), a web-based textual programming tool. LP requires real-time interaction between program and model, which means that changes to the algorithmic description must have immediate repercussions in the generated geometry. Both tools offer this liveliness aspect that intends to make the programming task easier to understand. In fact, the intuitiveness of LP is at the core of Grasshopper's success in the field.

However, besides the real-time interactiveness and immediate feedback of LP, in order to change the algorithmic description of the model we are immersed in, we require a Live Coding (LC) solution. Unlike LP, whose focus is the very activity of programming in an attempt to make it more compre-

hensible, LC is a creativity technique centered upon the writing of interactive programs on the fly. It is commonly used to create and showcase sound and image-based digital media (Rein et al. 2018). The Fluxus project [1] is a good example: a programming environment that allows users to code 3D animations that react to external input in real time.

LC can, nevertheless, be used as a design method for architects as well. In fact, the idea is already in place in some AD tools. Dynamo, for instance, presents the algorithmic components right alongside the generated model. This means users can modify their program in the same environment where the model is concurrently being updated, which is the particular LC strategy we are interested in as well.

## PROGRAMMING IN VIRTUAL REALITY

The creation of a workflow for programming in VR has been attempted by Elliot et al. (2015) with the RiftSketch project, and by Robert Krahn with CodeChisel3D [2]. 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.

In the architectural context, having the code displayed alongside the model in the VE allows professionals to change their models at will without taking their HMD off or leaving the VE. By transporting the programming environment to the VE we can have architects, and clients for that matter, inside the virtual representation of the projects, developing the algorithmic descriptions of the models, applying changes and visualizing them, in real time.

## LIVE CODING IN VIRTUAL REALITY

Live Coding in Virtual Reality (LCVR) entails the integration of LC in VR and its application to AD. AD allows the architect to code his design intent in a flexible and parametric way, which means the resulting algorithmic description represents not one, but multiple variants of the same model according to the chosen parameters. With LCVR, we take advantage of AD's flexibility in order to live code our models.

Using a HMD, the architect is transported to the VE, where both the algorithmic description and the generated model are present. From the VE, the architect can then apply modifications to the AD description, while the resulting model is concurrently updated in accordance with the changes made. For this workflow to take place, however, we require a sufficiently performant connection between both an AD and a VR tool.

### *Integrated Approach*

In the past, an integrated approach to AD (Castelo-Branco and Leitão 2017) was proposed, which entailed the creation of a single algorithmic description capable of generating equivalent models in various tools, depending on which paradigm the architect may find most beneficial at any given stage of the process.

LCVR in one such paradigm, whose place in the design process is yet to be figured out. As we saw before, the advantages VR brings to the design process seem to be widespread throughout the various design stages (Whyte 2003). Hence, we believe architects should be free to determine at which stage either paradigm best fits his design process.

Figure 1 presents a scheme of the integrated approach extended to accommodate the LCVR workflow. On the left, we notice the typical AD workflow, with the architect interacting directly with the AD tool from the desktop of his PC, coding in the Integrated Development Environment (IDE), the code editor, to which the tool is coupled.

For the implementation of an integrated solution, we require an AD tool capable of translating the given instructions into operations recognized by a series of different backends, whose use may vary along the development process, depending on the more pressing needs at that stage. We can have, for instance, Computer-Aided Design (CAD) tools for concept and form experimentation; BIM tools for more detailed stages where construction information is re-

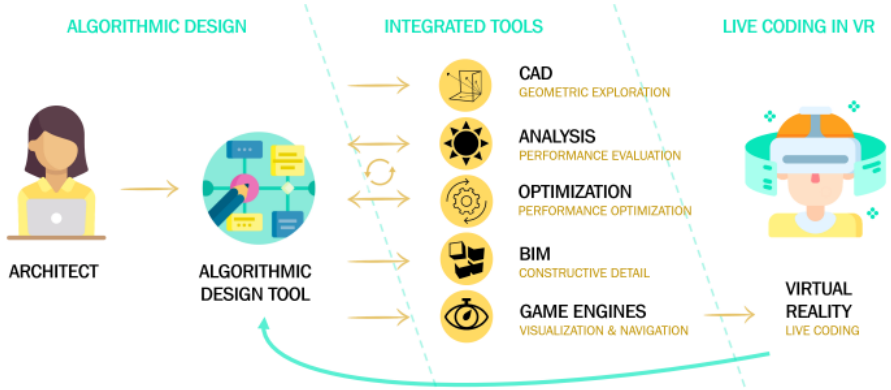


Figure 1
Integrated AD and LCVR workflow

quired; analysis tools to evaluate the design's performance; optimization tool to optimize designs based on the performance analysis; and, finally, Game Engines (GE) for fast visualization of the models in near real-time render quality, interaction with the models, and VR integration.

### *Live Coding Workflow*

GEs, in particular, are optimized visualization tools that allow for faster generation of models and low latency in live model manipulation and navigation. This makes them good candidates for a solution that relies on real-time code-model interactiveness.

Thus, on the right of figure 1, we can see a different mode of interacting with the AD tool, from the VE: LCVR. Anchored primarily on GE backends, this workflow allows the architect to continue developing the algorithmic description of his design, with all the advantages of typical AD processes, namely the generation of the geometry in alternative backends, however, whilst inside his creation, or at least the version of it offered by the GE backend. Furthermore, LCVR implies a live process, meaning the building changes around the user as he codes it on site.

There is, nonetheless, a setback to consider in this scenario. Despite the fast response guaranteed by GE tools, 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.

## IMPLEMENTATION

There are several tools that allow for the sort of portability required in an integrated approach, within both the visual and textual programming paradigms. For the 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 language, offers a direct connection to a GE backend: Unity (Leitão et al. 2019). In order to compare both paradigms, visual and textual in the context of LCVR, we developed a Khepri-based Grasshopper plug-in capable of communicating with any of Khepri's backends as well, but most importantly, Unity.

In order to implement the proposed workflow, three main features must first be considered: (1) the programming paradigm; (2) the projection of the programming environment onto the VE; and (3) the code manipulation mechanism. This section contains an overview of currently available solutions for the presented issues, grounded on the experimentations we made for each of them.

We offer a critical analysis and comparison of the various options in the light of a case study, a random pagoda city. An equivalent representation of the city was programmed in both a Visual Programming Language (VPL) - Grasshopper, and a Textual Programming Language (TPL) - Julia. However, and since a lot of this work is exploratory, many of the futuristic solutions we discuss regarding IDE projection onto the VE are presented as mockups only, all of which are properly identified as such.

### *Programming Language*

VPLs are generally more appealing to the architectural community, given their user-friendliness and smooth learning curve. However, they also lack scalability: as programs grow in complexity, they become harder to understand and change (Leitão et al. 2012). TPLs, on the other hand, offer more expressive power, flexibility, and efficiency, when compared to VPLs (Sammer et al. 2019). Model size and complexity are a non-issue for TPLs, which makes them a more appealing option when developing larger AD projects. However, TPLs usually require more extensive programming knowledge, and not all professionals find the required learning investment worthwhile.

Regarding LCVR, it matters not only the expressive and scalability power of the paradigm but also the code manipulation mechanisms available in each case. With LCVR, users will be coding in a VE, where the interaction with the programming environment will necessarily be different from the one they have coding in their PCs.

In this aspect, we argue that programming in VPLs is likely to render better results in live VR coding, as they require less textual input. While VPLs typically rely on dragging mechanisms for manipulating components, TPLs depend on textual input typically 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.

### *Integrated Development Environment*

In order to LC in VR, the architect requires a platform in which to program from the VE. For this problem, we considered four possible solutions: (1) mirroring the user's desktop in the VE, (2) having textual programs available for manipulation in a tailored textbox (3) having visual components projected onto the VE, and (4) having selected parts of textual programs projected onto the VE as 3D entities.

The first approach (1) **mirroring the user's desktop** in the VE, could work for any of the programming paradigms discussed (visual or textual). Since we are viewing, inside the VE, what we would outside it, we can essentially rely on the same IDE's we would use in a normal coding workflow. Besides being the easiest to implement, this solution also offers the user entire control over his program in the VE, meaning he can see and access anything he would if he were coding on his desktop. However, it presents a high level of intrusiveness, in the sense that the mirrored screen constitutes a partial visual blocker to the scene. Figure 2 presents the LCVR scenario for case (1) in both a visual and textual programming context.

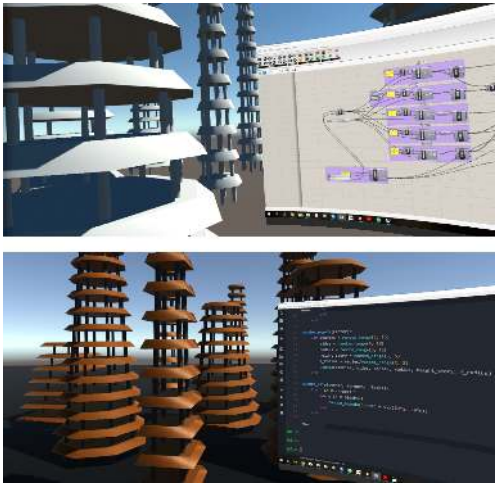

Figure 2
IDE option (1): mirroring the user's desktop. Grasshopper environment on top and the Atom IDE showcasing a program in Julia on the bottom

This solution also brings about concerns regarding the limited screen resolution offered by most of the current HMD, which greatly affects the sharpness of the text displayed in VR. This issue is a more immediate concern for TPLs, as these must display larger amounts of characters when compared to VPLs. The immediate solution implies increasing font sizes. However, this means less text can be shown at a time. The alternative would be to use higher resolution HMDs, which are becoming available as we speak.

Another possible solution entails taking advantage of the (2) **tailored textboxes** offered by Unity's Application Program Interface (API). This solution is most fit for TPLs and would imply having the code showcased in the textbox on screen. The user could then edit the code directly in the textbox. In this case, unlike the previous solution where the mirrored desktop is a 3D entity placed statically in the scene, 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 the user does not need to allocate the workstation each time he moves in the VE (figure3).

There are several disadvantages, however. With the first approach, the architect could use an IDE of his choosing to code live in VR, which represents great gains in coding efficiency since current IDEs offers debugging, syntax highlight, and other features that considerably help the coding task. Having the code transcribed onto a textbox, while less intrusive, offers even less aid to non-expert programmers.

Solution (3), having **VPL components projected onto the VE**, has two possible development paths: the first option would be, much like the previous solutions, to showcase the visual program in front of the scene, moving along with the user's viewport (figure 4 up); 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 4 down).

The first option would most likely be more intrusive, although the user might choose to turn the programming layer on and off. On the other hand, the second one suffers from similar issues to the mirrored desktop solution: the code does not accompany the user automatically as he moves around the VE. Furthermore, having 3D components floating in the scene along with the generated geometry might lead to confusing situations where code and resulting geometry intercept, thus becoming even harder to understand and manipulate.

Figure 3
IDE option (2) mockup: Unity's scripting API

Figure 4
IDE option (3) mockup: Visual components as 2D or 3D entities in the VE

The last approach, (4) having selected parts of **TPL programs projected onto the VE** as 3D entities, expects the user to select an object he wishes to modify. The program must then recognize the parcel of code that generated this object, or set of objects, and project it onto the VE, next to the referred objects.

Within the VE, the user can hence perform modifications to that parcel of code. Figure 5 presents the corresponding Mockup for this solution.

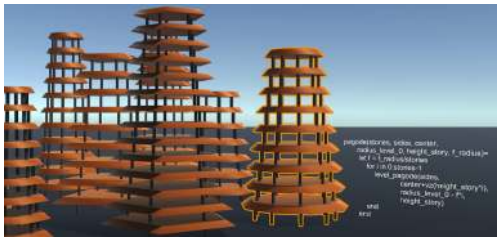

Figure 5
IDE option (4) mockup: selected text parts projected onto the VE

This approach, as opposed to having the entire program showcased in the scene, is considerably more interesting in terms of exploration 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 synthetic, which might help less experienced programmers orient themselves in the code.

Nevertheless, this solution requires substantial investment in finding intelligent ways of managing program traceability and showcasing the control flow of the program. The main problem lies in the fact that the features the user wants to change might be located deeper in the control flow of the program, and not specifically in the top-level function that generates the geometry. This means that a functional manner in which to navigate the various program parts in VR must be thought of. It also implies good traceability mechanisms on the part of the editor itself to allow users to jump from function to function as they climb down the abstraction ladder.

Finally, although the selected parts of the program would ideally appear and disappear automatically as the user selects or deselects geometry, we can also foresee geometry intersection issues in this case. In this scenario, the code would be popping up arbitrarily next to the selected part of the model, unless we can devise an algorithm capable of calculating the ideal code position according to both user's sight cone and neighboring geometry.

### *Code Manipulation*

In order to code in VR, one must be able to type in commands. 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., not to mention the search for components, which also requires text input.

To deal with textual input, we considered four currently available market solutions: (1) handwriting, (2) voice input, (3) typing on a virtual keyboard, and (4) typing on a physical keyboard.

(1) **Hand-written code** recognition presumes the existence of large enough databases of handwritten 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, for instance 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. Hence, this would likely become a big scale writing experience, which might ultimately defeat the performance purpose.

(2) **Voice input** is a bold approach, but probably the most comfortable for the user. Since Tavis Rudd presented his system to dictate code to his laptop in 2013 [3], a wave of vocal programming has been flooding the market with better voice coding solutions. 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 built-in the HMD. Nevertheless, we can foresee a series of obstacles in this solution as well. Primarily, current technology has low voice recognition accuracy in loud environments, which means expensive equipment might be required when working collaboratively in VR or show-

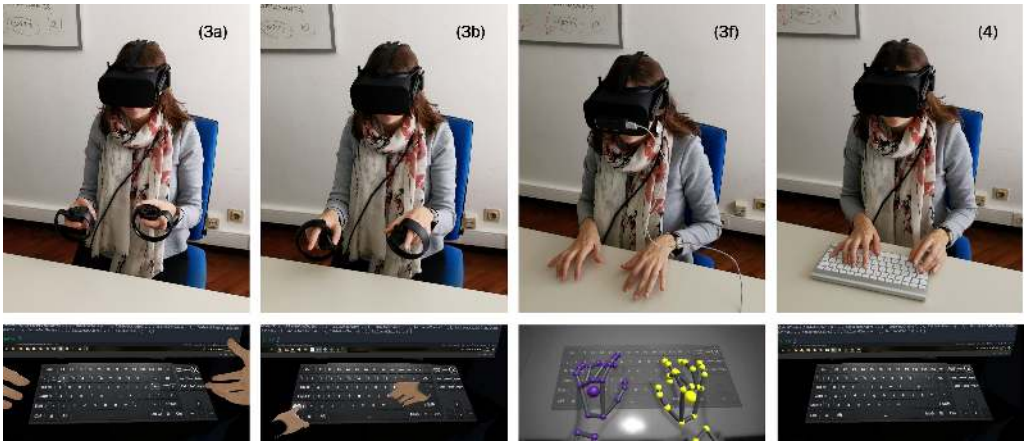


Figure 6
Tested typing options, with corresponding visual feedback in VR below: (3) virtual keyboard - wearing the VR controllers using (a) laser tags, (b) index fingers only, or (f) leap motion technology; and (4) using an occluded physical keyboard to type in, with a corresponding virtual keyboard with highlighted keys

casing projects to clients. Finally, and more specific to the problem at hand, dictating coding commands is not a trivial task and, thus, requires training[3].

The last two approaches are the more conservative ones, trying to mimic the user's workflow when coding AD traditionally, that is, using keyboards. Currently available solutions for the use of (3) **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 6) included the use of solutions (a) and (b), which proved to be very slow typing approaches. Solution (c) is considered a more efficient technique, yet it entails a steeper learning curve. For lack of available software, we did not include it in the experiment.

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 (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 (3f) in figure 6 corresponds to a prototype of the ideal solution, to be implemented if/when this technology achieves higher levels of accuracy.

Regarding (4) **physical keyboards**, which have already proven to beat the typing performance of virtual keyboards (Grubert et al. 2018a), it has been shown that occluded keyboards cause significantly more typing errors and speed reduction (Walker et al. 2017). Our findings concur with the literature review: the occluded physical keyboard yielded good results with experienced typographers, but not as good with less trained typographers, 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 keystriking 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 HMD, a workflow made possible by the characteristics of the one used in our experiments.

Finally, it is also possible to provide users with hand-position feedback using suggestions (e) and (f) described above, or 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 rates (Grubert et al. 2018b), 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 of no hope; we can always count on technological evolution to provide innovative solutions and constant improvement to existing ones. Naturally, this topic is more concerning in the context of TPLs. Although typing is also required in VPLs, the amount of it is not even comparable to that required to code in TPLs. 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 VR controllers suffice for the task at hand.

## 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 code displayed alongside the generated model in the Virtual Environment (VE), the architects can change the model at will without leaving the VE. The proposal is anchored primarily on Game Engine's (GE) ability to efficiently generate, navigate, and process geometry, which allows for the live coding performance to guarantee a near real-time update of the model.

We performed several informal user studies using the first IDE option presented: mirroring the user's desktop onto the VE, with both a Textual Programming Language (TPL) and Visual Programming Language (VPL). Regarding TPLs, we concluded that experienced typographers found no immediate setback to LCVR: using a physical keyboard to input text, they were able to program with the same ease they would outside the VE, with the added benefit of seeing the building change around them as they code it on site. Less experienced typographers, on the other hand, which constituted the majority of our sample group, found it hard to type with the same efficiency they would have outside the VE. Nevertheless, the experience proved to be very rewarding in terms of spatial awareness and live coding feedback.

Our VPL experiences were more limited. However, we could already conclude that the interaction mechanisms required, which are essentially limited to dragging and dropping components, facilitated the coding task in VR. This suggests they could be a stronger candidate for LCVR. Nevertheless, we also verified an aggravated scalability issue in the context. The pagoda exercise used as case study to test the LCVR approach was simple, but with a scalability parameter that proved to kill the VPL workflow, long before the TPL one started lagging, even outside VR.

VR technology alone is heavy on the hardware. Having an AD tool running that exhausts the machine as well, will lead to increased lags and struggles in this workflow. Furthermore, the LCVR approach best fits complex models, whose experience inside VR is most rewarding.

In the future, we plan on developing possible implementations for the mockup solutions presented, as well as conducting more formal user tests with studied metrics to infer the true potential and the adversities brought about by LCVR.

## ACKNOWLEDGMENTS

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UID/CEC/50021/2019 and PTDC/ART-DAQ/31061/2017.

## REFERENCES

Alfaiate, P, I, Caetano and Leitão, A 2017 'Luna Moth: Supporting Creativity in the Cloud', *Proceedings of*

*the 37th ACADIA*, Cambridge, pp. 72-81
Castelo-Branco, R and Leitão, A 2017 'Integrated Algorithmic Design - A single-script approach for multiple design tasks', *Proceedings of the 35th eCAADe*, Rome, pp. 729-738
Dorta, T, Lesage, A, Pérez, E and Bastien, J 2010, 'Signs of Collaborative Ideation and the Hybrid Ideation Space', in Taura, T and Nagai, Y (eds) 2010, *Design Creativity*, Springer, London, pp. 199-206
Elliott, A, Peiris, B and Parnin, C 2015 'Virtual Reality in Software Engineering: Affordances, Applications, and Challenges', *Proceedings of the 37th ICSE*, Florence, pp. 547-550
Grubert, J, Witzani, L, Ofek, E, Pahud, M, Kranz, M and Kristensson, PO 2018a 'Text entry in immersive head-mounted display-based virtual reality using standard keyboards', *Proceedings of the 25th IEEE*, Reutlingen, p. 159–166
Grubert, J, Witzani, L, Ofek, E, Pahud, M, Kranz, M and Kristensson, PO 2018b 'Effects of hand representations for typing in virtual reality', *In Proceedings of the 25th IEEE*, Reutlingen, p. 151–158
Gu, N, Kim, MJ and Maher, ML 2011, 'Technological advancements in synchronous collaboration: The effect of 3D virtual worlds and tangible user interfaces on architectural design', *Automation in Construction*, 20(3), pp. 270-278
Heydarian, A, Pantazis, E, Wang, A, Gerber, D and Becerik-Gerber, B 2017, 'Towards user centered building design: Identifying end-user lighting preferences via immersive virtual environments', *Automation in Construction*, 81, pp. 56-66
de Klerk, R, Duarte, A, Medeiros, D, Duarte, JP, Jorge, J and Lopes, D 2019, 'Usability studies on building early stage architectural models in virtual reality', *Automation in Construction*, 103, pp. 104-116
Koutsabasis, P, Vosinakis, S, Malisova, K and Paparounas, N 2012, 'On the value of Virtual Worlds for collaborative design', *Design Studies*, 33(4), pp. 357-390
Kuliga, S, Thrash, T, Daltonc, RC and Hölscher, C 2015, 'Virtual reality as an empirical research tool — Exploring user experience in a real building and a corresponding virtual model', *Computers, Environment and Urban Systems*, 54, pp. 363-375
Leitão, A, Castelo-Branco, R and Santos, G 2019 'Game of Renders: The Use of Game Engines for Architectural Visualization', *Proceedings of the 24th CAADRIA*, Wellington, pp. 655-664
Leitão, A, Santos, L and Lopes, J 2012, 'Programming Languages For Generative Design: A Comparative Study', *International Journal of Architectural Computing*, 10(1), pp. 139-162
Moloney, J, Globa, A, Wang, R and Khoo, CK 2018 'Pre-Occupancy Evaluation Tools (P-OET) for early feasibility design stages using virtual and augmented reality technology', *Proceedings of the 52th ASA*, Melbourne, pp. 717-725
Paes, D, Arantes, E and Irizarry, J 2017, 'Immersive environment for improving the understanding of architectural 3D models: Comparing user spatial perception between immersive and traditional virtual reality systems', *Automation in Construction*, 84, pp. 292-303
Portman, ME, Natapov, A and Fisher-Gewirtzman, D 2015, 'To go where no man has gone before: Virtual reality in architecture, landscape architecture and environmental planning', *Computers, Environment and Urban Systems*, 54, pp. 376-384
Rein, P, Ramson, S, Lincke, J, Hirschfeld, R and Pape, T 2018, 'Exploratory and Live, Programming and Coding - A Literature Study Comparing Perspectives on Liveness', *Programming Journal*, 3(1), p. 3
Sammer, MJ, Leitão, A and Caetano, I 2019 'From Visual Input to Visual Output in Textual Programming', *Proceedings of the 24th CAADRIA*, Welligton, pp. 645-654
Schnabel, MA 2011, 'The Immersive Virtual Environment Design Studio', in Wang, X and Tsai, J (eds) 2011, *Collaborative Design in Virtual Environments*, Springer Science + Business Media B.V., p. 177–191
Steinicke, F 2016, *Being Really Virtual: Immersive Natives and the Future of Virtual Reality*, Springer, Switzerland
Sutherland, IE 1965 'The Ultimate Display', *Proceedings of the IFIP Congress*
Walker, J, Li, B, Vertanen, K and Kuhl, S 2017 'Efficient Typing on a Visually Occluded Physical Keyboard', *Proceedings of CHI*, Denver, pp. 5457-5461
Wang, X and Schnabel, MA 2008, *Mixed Reality In Architecture, Design, And Construction*, Springer, Netherlands
Whyte, J 2003, 'Industrial Applications of Virtual Reality in Architecture and Construction', *ITcon Special issue Virtual Reality Technology in Architecture and Construction*, 8, pp. 43-50
Yan, W, Culp, C and Graf, R 2011, 'Integrating BIM and gaming for real-time interactive architectural visualization', *Automation in Construction*, 20(4), pp. 446-458
[1] http://www.pawfal.org/fluxus/
[2] https://github.com/cdglabs/CodeChiselJ3D
[3] https://www.youtube.com/watch?v=8SkdfdXWYaI