UMONS
Université de Mons

Université de Mons
Faculté des Sciences
Département d'Informatique

Faculté
des Sciences

Integrating Immersive Technologies for Algorithmic Design in Architecture

Adrien Coppens

A dissertation submitted in fulfilment of the requirements of
the degree of *Docteur en Sciences*

Advisors
Dr. Tom Mens   Université de Mons, Belgium
Dr. Mohamed-Anis Gallas   Université de Mons, Belgium

Jury
Dr. Bruno Quoitin   Université de Mons, Belgium
Dr. Gilles Halin   Ecole d'Architecture de Nantes, Université de Lorraine, France
Dr. Annie Guerriero   Luxembourg Institute of Science and Technology, Luxembourg

February 2022

Acknowledgements

As an important chapter of my life is about to end, I would like to express my gratitude towards those who directly or indirectly have been part of this four-year journey.

First and foremost, I would like to thank Tom, my advisor and head of the lab I have been part of this whole time. Thank you for giving me the opportunity to work with all these technologies I wanted to explore, even though they are outside of your main research interests, but also for your incredible availability (how many dozens of days do you have in a week?) as well as the swiftness, quantity and quality of your invaluable feedback and guidance along the way; and thank you for letting me borrow your most excellent (immersive?) research kit during the CAMPaM workshop; so, for all these reasons and many more, bedankt![1]

Anis, my second adviser, thank you for providing a context in which I could experiment with immersive technologies. Thank you for helping me benefit from your network of contacts in a field I knew nothing about, but also for turning your students into "guinea pigs" to test our prototypes and for accompanying me to these conferences and seminars.

Thank you as well to all jury and "comité d'accompagnement" members, for accepting to review my work and provide valuable feedback on it. Your comments and suggestions helped make this dissertation better. More specifically, thank you Annie for inviting me at LIST during these 3 months by combating the sanitary situation. Thank you Gilles for the discussions we had at various events, and for showing me that hybrid specimens in between computer scientists and architects do exist. ACK to you Bruno, for providing the external view on my work and for giving me an excuse to talk about technical details.

"Merci" (in Luxembourgish) to Nico, Elie, Sylvain, Calin and the people at

---

[1]Sentence intentionally left long and complex

LIST, for welcoming me in your office in order to collaborate on that prototype.

Thank you to all the colleagues I met during this time, and to the multiple co-workers I had the pleasure to share an office with. I would likely forget some of you so I would rather keep it generic but you all helped make our work environment such a nice and pleasant place.

Thank you to Robin, for choosing to work on my student project proposal, and for carrying it out nicely so that it contributed to the present dissertation. Thank you to all the "guinea pigs" that tried various versions of our prototypes and provided feedback on them, and to all survey respondents for their valuable time.

Thank you to Ivan Sutherland, for inspiring and pioneering both head-mounted displays and computer-aided design. As a nod to your research achievements and how their combination matches the subject of this dissertation, each chapter starts with an epigraph with quotes from you that felt appropriate.

On a more personal level, thank you to all my tennis and padel partners, hitting the ball with you allowed me to exert myself and helped clear my mind from all those 0's and 1's. Thank you to my friends, for providing all those relaxing and enjoyable times. Thank you to my family for your unwavering support during these four years but also since I was born.

Finally, thank you to all the "et al." that I may have forgotten.

Abstract

Architectural design practice has radically evolved over the course of its history, due to technological improvements that gave rise to advanced automated tools for many design tasks. Traditional paper drawings and scale models are now accompanied by 2D and 3D Computer-Aided Architectural Design (CAAD) software.

While such tools improved in many ways, including performance and accuracy improvements, the modalities of user interaction have mostly remained the same, with 2D interfaces displayed on 2D screens. The maturation of Augmented Reality (AR) and Virtual Reality (VR) technology has led to some level of integration of these immersive technologies into architectural practice, but mostly limited to visualisation purposes, e.g. to show a finished project to a potential client.

We posit that there is potential to employ such technologies earlier in the architectural design process and therefore explore that possibility with a focus on Algorithmic Design (AD), a CAAD paradigm that relies on (often visual) algorithms to generate geometries. The main goal of this dissertation is to demonstrate that AR and VR can be adopted for AD activities.

To verify that claim, we follow an iterative prototype-based methodology to develop research prototype software tools and evaluate them. The three developed prototypes provide evidence that integrating immersive technologies into the AD toolset provides opportunities for architects to improve their workflow and to better present their creations to clients. Based on our contributions and the feedback we gathered from architectural students and other researchers that evaluated the developed prototypes, we additionally provide insights as to future perspectives in the field.

Contents

Introduction

"It's not an idea until you write it down."

Ivan Sutherland

During my computer science studies, I had the opportunity to work with gestural interaction trackers and immersive technologies on several projects. I have to say I was hooked to these innovative interaction devices as soon as I got a hold of them. When the opportunity to pursue a PhD on these technologies arose, it was obvious to me I should go for it.

After a few meetings with both Prof. Tom Mens and Dr. Mohamed-Anis Gallas, we converged to an interesting area that combined my interests for the aforementioned technologies with an actual research problem: integrating immersive technologies into architectural design and Parametric Modelling in particular or, as I prefer to call it, Algorithmic Design (AD).

Due to its interdisciplinary nature, the current thesis was co-directed by Prof. Tom Mens from the Faculty of Sciences and Dr. Mohamed-Anis Gallas from the Faculty of Architecture and Urban Planning.

The present document presents my work on the subject, that was carried out during the last four years, and aims to help the reader understand why and how I proceeded as well as what perspective my work offers.

1.1. Context

Figure 1.1: A brief overview of the evolution of architectural tooling. Created based on pictures from online sources[1].

The Architecture, Engineering and Construction (AEC) industry has radically evolved due to technological improvements that gave rise to advanced automated tools for many design tasks. Traditional paper drawings and scale models are now accompanied by 2D and 3D **Computer-Aided Design (CAD)** software.

These tools have consistently improved in terms of performance and accuracy, and new features are constantly added to them, such as real-time (lighting, structural) simulations and photo-realistic renderings.

In fact, the inclusion of immersive technologies in architectural practice is currently mostly restricted to enhancing the visualisation of final models (Blach et al., 1998), built using traditional (non-immersive) desktop software. At that point, the design process is essentially over, and changes to the model based on the immersive feedback are unlikely.

---

[1]This figure was created using three CC0 pictures from www.pexels.com and one (the last one) from uploadvr.com/marui-plugins-bring-vr-support-to-3d-tools-maya-and-blender

We believe that there is potential to employ immersive technologies before the process is finished, in order to take advantage of the visualisation characteristics during design activities. These include the sense of scale, the immersion, the spatial perception as well as the interaction opportunities provided to the user (Delgado et al., 2020; Jayaram et al., 2001; Louis et al., 2020).

1.2. Algorithmic Design in architecture

Immersive technologies provide opportunities to visualise a geometry in a three-dimensional (virtual) environment. While this can be beneficial for architecture in general, this is particularly attractive to design processes that would benefit from rapid iteration cycles, where the designer would like to quickly transition from the editing software to an updated view of the rendered model, and vice versa.

So as to limit the scope of our work to a reasonable area, we chose to focus on **Algorithmic Design (AD)** (Woodbury et al., 2010; Monedero, 2000). We made that choice because that design paradigm inherently favours parametrised adjustment of designed models and therefore is a suitable candidate for the rapid iterations we suggest.

Furthermore, we identified a clear lack of support for immersive technologies with the most popular software tools supporting AD, leading to more opportunities for us to explore the field. We will consequently mostly discuss the integration of immersive technologies with AD in particular. But what is AD exactly?

AD is an architectural design paradigm that involves generating geometries using algorithms which are often driven by parameters that can be changed, allowing designers to explore different solutions by tweaking the values of these parameters. Algorithms can be represented in textual or visual forms, sometimes interchangeably, and may correspond to different programming paradigms (see section 2.3.3 for more information on the paradigms covered by AD languages and tools).

This enables the designer to explore different solutions by modifying the values of parameters from these algorithms, and facilitates the design process of complex structures, such as the control tower from the Managua Airport, represented in Figure 1.2.

Figure 1.2: Example of an architectural project that relies on AD (through Grasshopper and additional plug-ins) to generate a tower. Reproduced from an example project by Mario Alberto Espinoza[2].

In architectural practice, the most common representation form is flow-based programming (Morrison, 1994) through a visual interface. Tools such as Grasshopper[3], GenerativeComponents[4] and Dynamo Studio[5], stand out as the most popular software solutions (Cichocka et al., 2017).

Figure 1.3 shows an abstract representation of such a visual algorithm that generates a simple cube, with the corresponding rendered geometry on the right.

In flow-based programming, the final output is constructed by connecting processes that have an internal behaviour and return an output value. The idea is that in order to produce a result, one does not need to know the details about the inner workings of each process. The processes can be thought of as "black boxes", whose output values can simply be reused by other processes. Here, the output value of each process is often a geometry that is passed from one component to the next in order to construct the final output.

We chose Grasshopper as our target AD system for the real prototypes we aim to develop, because it is one of the most popular AD systems used in both research and industrial contexts (Cichocka et al., 2017; Stals et al.,

---

[2]http://www.iaacblog.com/programs/managua-tower/

[3]http://grasshopper3d.com

[4]http://bentley.com/products/product-line/modeling-and-visualization-software/generativecomponents

[5]http://autodesk.com/products/dynamo-studio

Figure 1.3: A basic example of an AD visual algorithm with its associated geometry.

2018b). Additionally, its software development toolkit supports C#, which is convenient considering my familiarity with Unity, that also supports C#. Choosing Grasshopper also means we have an easy access to a license and informal evaluators since the usage of that software is taught at the University of Mons, as part of one of Dr. Gallas' courses.

1.2.1 Terminology

On a terminological note, AD is often referred to as *computational design* or *parametric modelling*. Both of these terms appear too generic to us since they could apply to non-algorithmic design as well: the former simply informs that a computer was used, while the latter signals that the design is driven by parameters; parametric modelling (or parametric design) is in fact regularly confused with Building Information Modelling (BIM) (see section 2.3.2). A related term whose definition and relation with AD is sometimes unclear is Generative Design (GD). Some consider it to be a superset of AD, while others define it as a subtype of AD (Caetano et al., 2020). Most publications mentioning GD are related to (multi-criteria) optimisation (Villaggi et al., 2018a), machine learning techniques (Nagy et al., 2017a) and Domain Space Exploration (Calixto & Celani, 2015), all of which necessarily involve algorithms that generate designs. Hence, we would be tempted to agree with the latter group and consider GD as a subset to AD.

Since the most popular software tools for AD offer visual representations of the algorithms (similar to what is shown in Figure 1.3), more specific terms could be used to emphasise on that aspect. Based on computer science terminology, we could indeed consider these visual algorithms as graph structures (and therefore come up with a term such as "graph-based design") or even highlight the fact that we rely on flow-based programming (which would lead to a

term such as "flow-based design" or "dataflow-based design" to disambiguate with other types of flows such as control flow). However, these terms are not standard in the architectural field and appear to be too technical to be broadly used. Another issue that distances us from relying on terms that highlight the visual nature of these representations is that AD software generally also support textual programming. We will therefore simply refer to the previously defined modelling paradigm as AD from now on. This choice of term is consistent with a recent paper that results from a literature review (Caetano et al., 2020).

Additionally, we note that the "model" term in the context of AD can be ambiguous: it could be used to designate a visual AD program or the generated (or even rendered) geometry. For that reason, in the remaining of this document we will try to avoid mentioning "model" and instead use "algorithm" and "geometry" (or geometrical representation) to circumvent this ambiguity whenever possible.

1.3. Thesis statement

Since we consider the integration of immersive technologies to be particularly lacking in Algorithmic Design editors in architecture, our aim is to create and evaluate prototypical tools that show the potential of using these technologies as part of that design paradigm's toolkit.

The main question that this thesis dissertation tries to answer is: "How can immersive technologies be adopted for Algorithmic Design activities in architecture?". This leads to multiple sub-questions: Which immersive technologies are the most appropriate? How are these technologies perceived by practitioners? How can these technologies be integrated with existing tools, both technically and in terms of user interaction, so that they are adapted to AD practice ?

> **Thesis statement**
>
> Integrating Augmented Reality (AR) and Virtual Reality (VR) technologies into the Algorithmic Design toolset provides opportunities for architects to improve their workflow and to better present their creations to clients.

1.4. Research methodology

To provide evidence for the thesis statement, we demonstrate that both **Augmented Reality (AR)** and **Virtual Reality (VR)** can be integrated into AD practice, through the implementation of several prototypes. They show that such technologies are useful additions to the toolset at the architects' disposal for AD activities. We substantiate that usefulness through surveys and interviews.

To structure our course of action, we generally follow an iterative and prototype-based process. Figure 1.4 depicts that process, which closely resembles the action research spiral (Kemmis et al., 2014) (plan, act, observe, reflect, re-plan, etc.).

Figure 1.4: The iterative research process we followed to support the thesis statement.

First, we identify a gap in existing tooling, that we attempt to fill with a prototype. This allows us to organise evaluations of that prototype with students and academics, through workshops and visits in universities. Based on the feedback we thereby gather, we identify further needs and therefore start another iterative cycle with a new prototype.

During these four years, we repeat that cycle three times, with three different prototypes being developed. The first prototype enables geometry streaming and parameter adjustment in VR, through a bridge between Grasshopper and a virtual environment. The second one lets users have greater control over

their design, with actual algorithm editing capabilities. That control necessitates novel interaction mechanisms and we therefore explore a few options via different techniques and devices. The third and last prototype goes back to parameter adjustment but operates through more accomplished visualisation and interaction systems.

1.5. Thesis structure

The thesis is split into 5 chapters. In between the current introduction (chapter 1) and the conclusions (chapter 5), the reader will find a background literature review followed by two chapters (3 and 4) describing and discussing the developed prototypes.

Specifically, chapter 2 presents and discusses the state of the art and the state of the practice in several domains that are heavily tied with our work. This comprises an introduction to the terminology in use for immersive technologies, a general overview on computer-aided (architectural) design, and a section on how to interact within three-dimensional environments. Aggregated together, these knowledge pieces form the basis on top of which we caved out an iterative research process.

Chapter 3 and 4 describe the prototypes developed during each iteration cycle, grouped by interaction level i.e., how much control they offer to the user over their design. Chapter 3 focuses on the two prototypes that cover parameter adjustment, while chapter 4 introduces a third prototype that enables control over the visual algorithm itself. In both chapters, we present the building blocks the prototypes are made of, discuss how they were received by evaluators, and explain the rationale behind the decisions we made to move on from each prototype to a new one.

Finally, chapter 5 summarises our contributions towards the thesis statement, reflects on the developed prototypes, their use cases and usage scenarios, and opens perspectives on how our research could be continued. We additionally provide recommendations and projections of where the field should head towards.

It should however be noted that this thesis structure does not exactly reflect the methodology or chronology of our research developments. It is rather organised based on the relation between the features and use cases that our iterations aim to cover.

2

State of the art

> "I just need to figure out how things work."
>
> Ivan Sutherland

Before diving into our work, section 2.1 introduces the display and interaction technologies commonly used for creating immersive experiences, as well as the corresponding terminology.

Section 2.2 presents three-dimensional visual modelling techniques that have been proposed for various application domains, so that we may draw inspiration from such techniques and apply them in the Algorithmic Design context.

Section 2.3 goes through the history of Computer-Aided Design and its architectural counterpart, before focusing on architectural design through programming and Algorithmic Design itself.

Then, Section 2.4 presents an overview of three-dimensional interaction techniques that are adapted to immersive manipulations, so that we may choose the techniques that are most appropriate to the Algorithmic Design context.

Finally, Section 2.5 addresses new ways of interacting with architectural designs, including optimisation techniques, exploratory approaches, and the use of immersive technologies, to help us better picture the landscape of existing integrations of these technologies and techniques in the architectural design field.

2.1. Immersive technologies

Terms such as Augmented Reality and Virtual Reality are now familiar to a lot of people, thanks to well-known applications such as Pokémon Go and popular devices such as HTC Vive[1] and Oculus Rift[2]. In the last few years, these immersive technologies have been subject to an increasing attention in research, business and society in general (Suh & Prophet, 2018). This section clarifies the terminology and how immersive experiences relate to one another, then presents the underlying technologies and provides examples of use cases.

2.1.1   Terminology

A very frequently used taxonomy is the Reality Virtuality (RV) continuum (Milgram & Kishino, 1994) represented on Figure 2.1. Experiences that mix real and virtual elements can be placed on that continuum, whose reach goes from the real world to an entirely virtual one.

Figure 2.1:   The original RV continuum (Milgram & Kishino, 1994)

This includes AR, that describes the superimposition of virtual elements onto the real world. It can basically be seen as "adding virtual things on top of the real world's perception". A typical example of an AR device is Google Glass[3]. The game Pokémon Go[4] helped popularise AR with its integration of Pokémon creatures on top of the live camera feed as if these creatures were there in the real world (as shown on Figure 2.2a).

On the other end of that continuum, we find VR, that fully immerses users into a three-dimensional virtual world. While the paternity of the concept is unclear, VR as a term is generally attributed to Lanier who worked actively (Lanier, 1988; Conn et al., 1989) in the domain in the late 1980's. The usual equipment used for VR experiences involves a visualisation system, typically

---

[1]`www.vive.com`

[2]`www.oculus.com/rift`

[3]`https://google.com/glass/start/`

[4]`https://pokemongolive.com/`

(a) Superimposing a Pokémon on top of the camera feed (AR feature).

(b) A person wearing a VR headset, occluding that person's field of view.

Figure 2.2: Example images[5]on the two most commonly mentioned immersive technologies: AR and VR.

either a Head-Mounted Display (HMD) or multiple wall-sized displays, as well as some interaction device, such as a controller, a glove or a tracking system.

On the RV continuum, two other terms are mentioned: Augmented Virtuality (AV) and Mixed Reality (MR). AV can be seen as the "opposite" of AR, since it describes experiences that augment the virtual world with real world elements. We will not talk much about it, not only because it is much less commonly encountered, but also because its frontier with AR and VR can be blurry, especially with the ever-improving technologies that will make virtual elements harder to distinguish from real ones.

The original definition of MR encompasses everything that falls in between the extrema of the aforementioned RV continuum: a MR environment is "one in which real world and virtual world objects are presented together within a single display" (Milgram & Kishino, 1994). However, the term MR is often used nowadays to describe AR devices and experiences that show an advanced degree of spatial understanding, generally thanks to an environment scanning system. Figure 2.3 depicts such an experience, with a virtual building properly anchored to a real table. This would be described by some as a MR experience. While that statement is not fundamentally false (since AR is a subset of MR according to the original definition), it leads to confusion and may prompt unfamiliar individuals to reject the classification of that experience as AR.

We believe the communication and the media coverage around the Mi-

---

[5]The figure was created using images from www.phonandroid.com/pokemon-go-guide-astuces-progresser-facilement.html and www.pexels.com, respectively.

crosoft Hololens (an AR headset capable of scanning its surroundings) played a big role in that deviation from the original definition, since it is mostly referred to as a MR device. The misuse of that term, with regards to its original meaning, became so common that a new term was coined to describe exactly what MR initially was: eXtended Reality (XR).

Figure 2.3: Commercial visual of an AR experience with proper spatial mapping of virtual elements, that would be classified as MR by some[6].

However, we admit that the ability to properly understand the surroundings as part of an AR experience enables a whole new range of applications. We therefore suggest to introduce a new term as a subset of AR, to recognise that particularity without borrowing Milgram & Kishino's original MR term. Three options we suggested in (Coppens, 2017) were "Spatial-Aware AR", "Surroundings-Aware AR" and "Spatial-mapped AR".

Many other taxonomies have been proposed; there is even another one suggested in the paper that also describes the RV continuum (Milgram & Kishino, 1994): a three-dimensional hyperspace that places MR systems according to 3 axes. These 3 axes are: Extent of World Knowledge (EWK: how much we know about the - real or virtual - world in which the experience happens), Reproduction Fidelity (RF: how realistic the augmented content is) and Extent of Presence Metaphor (EPM: how immersed the user is). Other examples of classifications are based on location and temporality (Fuchs, 2006), on intended purpose (Dubois et al., 2000), on the type of entity that is being augmented

---

[6]Picture from Microsoft's Mixed Reality documentation portal, found on `https://docs.microsoft.com/en-us/windows/mixed-reality/design/color-light-and-materials`.

(Hugues et al., 2011; Mackay, 1996), or on who is in control of the experience (either the system or the user) (Renevier, 2004).

2.1.2 Technological building blocks for immersive experiences

In order to create such experiences, there are a few technological needs, mainly in terms of display and tracking devices. This section will cover the most common techniques currently in use.

Displays for VR

Since their purpose is to immerse the user into an entirely synthetic world, VR-enabling displays have to occlude a large part of the user's field of view. For that reason, the typical representation of a VR user as of today is a person wearing a HMD, as pictured on Figure 2.2b. These HMDs are generally binocular, meaning that each eye can be presented with a (slightly) different image. This results in the ability to produce stereoscopic imaging, that creates a three-dimensional effect, fooling the wearer's perception into feeling immersed in a three-dimensional virtual space.

Although more expensive, wall displays can also be used to create VR experiences. These projection-based systems are usually referred to as CAVE-like setups, since the first occurrence of such a system was the Cave Automatic Virtual Environment (CAVE) (Cruz-Neira et al., 1992), which dates back to the early 1990's. A more recent example of a CAVE-like arrangement is shown in Figure 2.4, where we can see the projections on the walls of a cubic room as well as on the floor. These types of setups provide a better sense of presence (Juan & Pérez, 2009) but are typically a lot more costly. Furthermore, considering we work in the context of architectural design, it should be noted that the spatial understanding and distance perception is not necessarily better than those experienced with HMDs (Ghinea et al., 2018).

Displays for AR

From the user's point of view, the simplest form of AR displays are monitor-based ones, where the augmentation happens on a distant screen such as a television or a mobile device (a more precise term is then available: hand-held AR). The display is treated as a window to the augmented world, hence the alternative name "Window-on-the-World" from (Milgram et al., 1995). Examples of experiences relying on such displays are analysis tools for sport broadcasting (with virtual elements such as names and arrows being superimposed to the

Figure 2.4: A CAVE-like setup used to create a virtual grocery store. Reproduced from (Borrego et al., 2016).

camera feed), or the previously mentioned AR feature in Pokémon Go (Figure 2.2a).

Another non-intrusive alternative is Spatial AR, that is sometimes referred to as projective AR since it is about augmenting reality by projecting images directly onto real objects. It was first introduced in (Raskar et al., 1998) and has the advantage that it naturally provides multi-user experiences. Combined with the non-intrusive aspect, this technology becomes a good candidate for cultural contexts e.g., for museum exhibitions and monumental projections.

While it is possible to create an AR experience by using a VR headset mounted with a camera (to project the camera's input to the displays), it typically creates latency, since there is a delay between the recording of an image and the moment it is being displayed to the wearer. These displays are classified as video see-through but the display delay can create cybersickness (a term used to cover various symptoms generally including nausea that resemble motion sickness). In fact, cybersickness is believed to be heavily tied (LaViola, 2017, p. 49) with mismatches between visual and vestibular (body balance and movement) information, and solutions that limit such mismatches are usually preferred to mitigate the issue.

Another option is to use optical see-through displays, that present the user with a direct view of the real world, in the sense that the real world is not occluded, obviating the need to project the world on the display. In that case, the augmentation happens on a transparent surface placed in front of the observer, and virtual elements are integrated into the observer's view of the real world. There are multiple ways to place the augmentation surface relative to the observer. Figure 2.5 helps in picturing them, while also serving as a visual summary of the terms introduced in the present section.

Figure 2.5: Main options to produce AR experiences. Reproduced from (Bimber & Raskar, 2006).

An additional term is introduced in Figure 2.5: retinal display. Sometimes named Retinal Projection Display (RPD) or Virtual Retinal Display, such a device directly projects AR content on the user's retina, using low-power laser beams. The technology has not matured yet but several prototypes are available (Schowengerdt et al., 2003; Takahashi & Hirooka, 2008). Theoretically, when the technology develops into a small form-factor, wide field of view, high fidelity display, it should supersede most other AR technologies and could even lead to devices capable of switching back and forth between AR and VR modes.

It should also be noted that we only talked here about visual AR as it is the most common type of AR. Other kinds of augmentations exist, targeting other senses: smell (olfactory AR), touch (haptic AR), taste (gustatory AR) and hearing (audio AR). These technologies fall out of scope of the current dissertation since we did not rely on them for the experiments we conducted.

Tracking needs

Both AR and VR experiences typically require some form of motion and positional tracking. Common techniques to enable suck tracking are hereby presented.

The bare minimum to enable HMD-based experiences is rotational tracking for the wearer's head. High quality headsets (such as the HTC Vive or the Oculus Rift) also include positional tracking capabilities. This allows the wearer's head to be fully-tracked in 3D space i.e., we know its location along the three (x,y,z) positional axes as well as its orientation along the three usual (yaw, pitch, roll) rotational axes. Since this gives six separate and independent ways to the head to modify its situation, a device with these tracking capabilities is usually labelled as a Six Degrees of Freedom (6-DoF) tracking system. Figure 2.6 clarifies the concept, with each color representing one degree of freedom.

Figure 2.6: The six degrees of freedom we have access to in 3D space. Reproduced from Wikimedia Commons[7].

The ideal tracking system should be accurate, precise (no jitter), fast (low latency and high refresh rate), robust (immune to environmental factors), and provide great mobility (lightweight, wireless or even autonomous, small form factor) in a wide area. This is obviously ambitious, but the next section describes major tracking technologies that attempt to reach that goal.

Tracking technologies

The simplest tracking method is likely mechanical tracking, where the tracked object is directly attached to the tracking system so that an object's position and rotation can be determined from sensors placed on the joints of the tracking system. This technique can lead to bulky and cumbersome systems with mechanical arms made up of articulated pieces, such as the Sword of Damocles, pictured in Figure 2.7.

---

[7]https://commons.wikimedia.org/wiki/File:6DOF.svg

Figure 2.7: The sword of Damocles. A mechanical tracking system designed for an early HMD. Reproduced from (Sutherland, 1968).

Another tracking option that is (one of) the most popular on the market is inertial tracking, driven by accelerometers, gyroscopes and, sometimes, magnetometers.

An accelerometer measures an object's acceleration along one axis, taking gravity into account, whereas gyroscopes measure rotation around a single axis. Combining three sensors of each type therefore theoretically suffices to obtain a 6-DoF tracker that provides relative position and rotation data, reporting how much the position and rotation has changed since the tracking process started. Magnetometers can additionally provide a reference heading to stabilise other sensors' measurements, since they measure magnetic fields and can indicate their direction (e.g., the Earth's magnetic field's direction for a compass). The combination of these types of sensors in a tracking system is often referred to as Intertial Measurement Unit (IMU) since they are based on the principle of inertia ($F = ma = m\frac{dv}{dt}$), used to derive a relative position and rotation.

The inertial tracking solution is widely used since the necessary sensors are relatively cheap and small, and are often already integrated in most smartphones. In practice though, an IMU has to be coupled to another tracking technology to provide positional information, since the drifting of accelerometers very quickly leads to unusable data. This is due to the relative nature of the measurements that necessarily accumulate errors, coupled with the fact that an accelerometer measures an acceleration that is then used to calculate a movement, meaning that the accumulated error on the acceleration is reported quadratically on the relative position output. The inability of smart-

phone IMUs to provide reliable positional tracking on their own is the reason smartphone-based VR headsets can only produce limited experience where real world movement cannot be taken into account (the virtual point of view therefore is fixed or controlled externally) and it can also cause additional cybersickness compared to higher-end systems, since a user's head generally slightly moves when rotating.

Another popular technique is optical tracking, where some kind of sensor (e.g., a camera) tracks known patterns, features, or markers in the surrounding environment. The sensor can be external to the tracked object (outside-in tracking) or attached to it (inside-out tracking).

Marker-based systems rely on markers placed on the tracked object. These markers can be active (light-emitting sources) or passive (easily identifiable items that do not emit light themselves) and are typically not visible to the human eye (e.g., using infrared lights), except when relying on paper markers, such as the ones presented on Figure 2.8. The latter option provides a cost-efficient way to produce AR experiences with printed papers and a simple camera. A very common marker-based library amongst AR developers is ARToolKit, that was initially developed as part of an AR-based conferencing system (Kato & Billinghurst, 1999).

Pattern-based solutions (such as Microsoft's Kinect sensor[8]) project a known pattern on the environment and observe the distortion of that pattern to derive spatial data, while feature-based solutions identify (and track) features to construct a 3D representation of the surroundings that ultimately allows the system to build a 3D plan.

Figure 2.8: Some well-known AR marker types, sharing similarities with QR codes. Reproduced from (Kan et al., 2011).

---

[8] https://developer.microsoft.com/windows/kinect

That last option is strongly related to the simultaneous localisation and mapping (SLAM) problem (Durrant-Whyte & Bailey, 2006), that describes the mapping of an unknown environment by a mobile robot. Since it has no previous knowledge of the surroundings, the robot must solve two related and interdependent problems at once: localising itself and mapping the environment. The problem is typically solved using feature descriptors and feature-tracking algorithms to derive the position of landmarks in the environment, combined with a variety of mapping methods. Recently, neural networks have been extensively used to solve SLAM instances (Chaplot et al., 2020; Zhang et al., 2017). Relying on SLAM-based techniques can allow for good tracking quality while suppressing the need for markers, but is harder to develop than most tracking solutions and is typically sensitive to environmental changes (e.g., moving objects). Some VR headsets such as the Oculus Quest[9] rely on SLAM techniques to provide tracking capabilities. A more complete description of the SLAM problem and the associated state of the art is out of scope of the current thesis, but it remains an important component of spatial-aware AR HMDs.

Figure 2.9: Valve's Lighthouse tracking system, using optical (infrared) tracking and active markers. Reproduced from a YouTube video[10]that explains how the system works.

Every tracking technique has its drawbacks and combining the strengths of different technologies produces the best results (Welch & Foxlin, 2002). For that reason, lots of trackers rely on hybrid solutions. A relevant example would be Lighthouse, Valve's tracking system for the HTC Vive, that relies on an IMU for rapid updates while the more accurate position is obtained

---

[9]`www.oculus.com/quest`
[10]`https://www.youtube.com/watch?v=J54dotTt7k0`

from an optical tracking system (using infrared and active markers) to limit the IMU's drifting issue to the few IMU updates between the optical system's updates. Figure 2.9 pictures a red laser beam about to hit the HMD to provide a one-dimensional update (the system uses two base stations that each emit two beams per update).

Note that the technologies described in this section can be used for head tracking but also for interaction devices e.g., controllers or gloves, and even to provide hand or body tracking.

2.1.3   Application domains

In general, AR and VR are most useful when they can provide a virtual equivalent to applications where a real implementation would be too difficult (or even impossible), too costly, or too dangerous to realise. This section covers such scenarios along with other AR or VR-based application domains, ignoring architectural design for now since that specific field will be discussed in further details in Section 2.5.2.

Entertainment

The entertainment sector is a major player in the popularisation of immersive technologies, with a plethora of movies and games being created. In addition to the previously mentioned smartphone game Pokémon Go, other popular titles have been extended to AR, with games such as ARQuake (Thomas et al., 2002) that lets students shoot virtual monsters in their university campus. Some of these AR games are based on real sports, like tennis (Henrysson et al., 2005), where players facing each other control virtual racquets through their phones. Many VR games have been developed in the last few years, some of them by researchers themselves, e.g., a geocaching game (Brade et al., 2017) to compare presence and usability in VR with the real world equivalent, but most studies rely on commercial games with VR support, such as Team Fortress 2 (Martel et al., 2015) or Half-Life 2 (Tan et al., 2015).

Outside of video games, AR has been used to enhance book experiences (Billinghurst et al., 2001) or card and board games (Lam et al., 2006; Lee et al., 2005; Molla & Lepetit, 2010). Researchers have also investigated VR movies and the consequences the medium has for movie makers (Serrano et al., 2017), or how it impacts viewer engagement (Gruenewald & Witteborn, 2020).

The sport industry also makes use of AR, especially for broadcasting, where overlays are often added on top of camera images to show additional information to the viewer. Such augmentations are common to point out specific

athletes, what they did or should have done, or to draw virtual lines in order to indicate distances. Figure 2.10 pictures a similar experience but for a spectator that is actually inside the stadium.

That being said, both AR and VR raise interest in a wide range other domains, some of which are covered in the remaining of this section.

Figure 2.10: An AR feature to superimpose virtual line marks on a rugby field, through a Hololens headset. Reproduced from (Zollmann et al., 2019).

Healthcare

Healthcare is a very large field on its own, with many opportunities for immersive technologies to shine. Thanks to the sense of presence induced by VR and the actual presence inherent to see-through AR, many psychiatric treatments relying on these technologies have been proposed. A plethora of anxiety and phobias are covered by such treatments, since they can expose the patient to their fears in a controlled environment, meaning the practitioner can choose the degree of exposure and even abort the experience at any time. As for VR-based treatments, a term has even been coined to encompass such therapies: Virtual Reality Exposure Therapy (VRET). VRETs have been proposed for flying phobia (Botella et al., 2004), fear of heights (Krijn et al., 2004), animal phobias (Carlin et al., 1997) (also in AR (Botella et al., 2005; Juan et al., 2005)) and Post-Traumatic Stress Disorders (Rizzo et al., 2009).

Other healthcare applications include the assessment and rehabilitation of

disabilities, following brain injuries (Rose et al., 2005), strokes (with both VR (Jack et al., 2001) and AR (Mousavi Hondori et al., 2013)), or amputations (with phantom limbs AR treatment (Carrino et al., 2014))

These technologies can additionally find usages during interventions, with VR being used for pain distraction (e.g., during heavy interventions), while AR can help surgeons with augmented overlays (Fuchs et al., 1998; Sato et al., 1998).

Education and training

Both AR and VR have been used in educative contexts, e.g., for surgical education (Basdogan et al., 2007), as part of anatomy courses (with VR (Nicholson et al., 2006) and AR (Blum et al., 2012)), or to teach astronomy (Fleck & Simon, 2013). These examples back up our claim that these technologies show their potential when a physical counterpart to the virtual experience would be harder or impossible to implement.

Thanks to the ability of immersive environments to replicate the real world, potentially in a realistic way and including sensible physics simulations, immersive technologies provide opportunities for training applications. Examples include firefighters (Xu et al., 2014) and astronauts (Aoki et al., 2007) training in VR, as well as individuals learning assembly tasks (Reiners et al., 1999) or military operations (Brown et al., 2006). These types of experiences have proven to be successful in allowing their users to transfer virtually acquired skills to the real-world counterpart of the target activity. As an example, researchers have demonstrated (Michalski et al., 2019) that real table tennis skills can be improved through VR training.

Culture and tourism

Many touristic locations offer binoculars to visitors, so that they can better observe the surroundings in exchange for a bit of money. Sometimes, these devices are augmented with information on or pointers to specific points of interest, thereby creating an AR experience (Fritz et al., 2005).

More advanced usages of immersive technologies in a similar context are also common, with cultural heritage experiences allowing users to visualise monuments (Gaitatzes et al., 2001) or inhabitants (Noh et al., 2009; Vlahakis et al., 2001) that have since disappeared. Similar experiences have also been created for places that still exist but cannot (easily) be visited, while other are simply made for advertising purposes (Kim & Hall, 2019; Loureiro et al., 2020).

Industrial maintenance and complex tasks

As discussed in section 2.1.3, both AR and VR are used to train workers, including for maintenance tasks (Gavish et al., 2015), but AR can also act as a virtual assistant when actually performing these tasks. Specific pieces that the worker has to manipulate can be superimposed with information and related 3D models can even be displayed (Schwald & De Laval, 2003) to help with the task at hand. That principle has been followed to support various applications such as welding (Echtler et al., 2004) and pump maintenance operations (Garza et al., 2013).

The augmented information does not necessarily have to be set in stone, as collaborative solutions also have been developed, with remote engineers able to place indicators when needed (Bottecchia et al., 2010; Benbelkacem et al., 2011). AR-based solutions could therefore, in some instances, replace lengthy manuals with dynamic, world-anchored and collaborative digital equivalents.

2.2. 3D visual modelling

Architectural design is not the only field employing visual modelling, as similar approaches have been applied to many domains, and three-dimensional versions of some of the corresponding tools were developed.

In fact, numerous 3D programming languages for virtual 3D environments have been designed, most of which rely on three-dimensional dataflow diagrams to define programs. Examples from the early 90's include CUBE/CUBE-II (Najork & Kaplan, 1991; Najork, 1996), a functional language based on a dataflow metaphor, and Lingua Graphica (Stiles & Pontecorvo, 1992), that translates from/to C++ code. Figure 2.11 presents a CUBE definition that returns the factorial of a given number. The upper plane is only executed when the input value n is equal to 0, and outputs 1 in that case. The lower plane is only executed when the input value n is greater than 0 and outputs the product of the input value with the output of the "!" function for the value $n - 1$. This indeed returns $n!$ since $n! = n * (n - 1)!$ for $n > 0$. We can additionally note that if a negative value were to be provided as input, nothing would be returned since none of the planes' conditions would be satisfied.

The previously mentioned languages target general programming, but the main motivation to create 3D languages and editors is often to match the dimensionality of the program with its output, so as to integrate both of them in a single virtual environment. As an example, SAM (Solid Agents in Motion) (Geiger et al., 1998) is another early 3D visual language that enables "parallel

Figure 2.11: The CUBE programming language: a recursive definition named "!", that calculates the factorial of a given number, with a planar projection of the lower plane on the right. Reproduced from (Najork, 1996).

systems specification and animation", targeting animated 3D presentations.

2.2.1 Immersive authoring of visual models

While the previous examples were indeed designed with virtual environments in mind since the authors all mention such environments in their respective papers, they were never adapted to immersive displays (e.g., VR), despite the alledged benefit of immersing the programmer in the same environment as the program and its output, an approach sometimes referred to as embodied spatial programming.

On the other hand, Steed & Slater implemented an immersive system that allows users to define object behaviours whilst being immersed (Steed & Slater, 1996), once again through dataflow graphs. The system could be used to design animations or interactive applications, that conveniently also took place within the virtual environment. With similar goals in mind, (Lee et al., 2004) presented an AR system to define the behaviour of scene objects for AR applications. A more recent example of a more accomplished VR authoring system is FlowMatic (Zhang & Oney, 2020), that not only lets users create, destroy, or define basic animations for 3D models but also allow them to define an object's behaviour depending on discrete events such as timers or collisions.

Another application domain is the Internet of Things (IoT), with prototypes such as Ivy (Ens et al., 2017), a VR-based programming tool that al-

lows its users to define the behaviour of IoT systems that depend on sensor data, through yet another visual dataflow-based representation. Figure 2.12 shows the tool's interface, with coloured particles that represent data flowing through the links. Aiming towards a similar goal but using AR, the Reality Editor (Heun et al., 2013) allows (re)programming of smart objects and their relations to others, so as to define a system's behaviour. A more recent prototype is CAPturAR (Wang et al., 2020), that serves a similar purpose but offers an activity-recording feature using a body-tracking system, to help in defining scenarios.

Figure 2.12: The Ivy programming tool that helps with designing sensor-driven systems. The depicted environment places the user in an industrial fabrication workshop, and the displayed program cuts the workshop's power if excessive vibrations are detected on a machine. Reproduced from (Ens et al., 2017).

2.3. Computer-Aided Design

2.3.1 Evolution of Computer-Aided Design

The starting point for CAD likely dates back to the 1950's. During the first half of that decade, computers started to be produced for commercial purposes and people naturally began to imagine and reflect on how they could be used for design activities. Those reflections lead to conceptual developments in

the following years; a good example being the artistic impression of a design workstation, depicted in Figure 2.13 and published in the Fortune magazine in 1956.

Figure 2.13: Artistic impression of a design workstation, from the Fortune magazine in November 1956. Reproduced from (Mitchell, 1989).

Academics also picked up interest in the potential of using computers for design activities, with a notable milestone in 1959 when a meeting took place at MIT (Coons, 1963) between the "Computer Applications Group" and the "Mechanical Engineering Department". That meeting concluded that computers had a significant role to play for (engineering) design and notably lead, a year later, to a report (Ross, 1960) whose author is often credited with coining the Computer-Aided Design term; a term that was used in the report's title.

Ever since these early concepts, CAD tools were indeed developed and have matured over the years: they reached new domains, expanded their functionality and their adoption progressed dramatically. In (Horváth & Vroom, 2015), the authors identified 5 periods of evolution for CAD; the remaining of this section is inspired by that paper's structure, although it should be noted that such a periodisation (dividing history into periods) always involves some degree of subjectivity and arbitrariness.

Early research developments (1960's)

A seminal work in the history of CAD is Sutherland's PhD dissertation (Sutherland, 1964) in 1964. He developed Sketchpad, a computer program that is generally considered as the first CAD system. It pioneered the use of a graphical user interface, controlled by a light pen (an old version of a stylus pen that works with CRT displays) and allowed users to create lines and curves, as seen in Figure 2.14. In addition to drawing these simple shapes, the user could also create and apply constraints to the drawing and define certain drawings as master objects, that could then be instantiated (any change to the master drawing automatically updates the clones).

Figure 2.14: Sutherland drawing simple shapes and applying constraints using Sketchpad. Reproduced from (Fusté Lleixà, 2018).

The system was first presented at the 1963 AFIPS conference (Johnson, 1963) and a video demonstrating its functionality at the time can be found online[11]. The conference covered information science in general but the 1963 edition had a track dedicated to CAD, in which other CAD systems were presented, together with more conceptual contributions (e.g., (Coons, 1963) outlining the requirements for CAD systems).

The first conference fully dedicated to CAD was created shortly after, in 1964 (Horváth & Vroom, 2015). Contemporaneously to those research-driven explorations, some large industrial firms also noticed the potential of the technology and developed in-house applications for their own needs (e.g., the DAC-1 system at General Motors).

Another important milestone for CAD history during that period was the

---

[11]https://www.youtube.com/watch?v=6orsmFndx_o

development of Bézier curves (with their mathematical representation) that are still in use nowadays. That family of curves was invented independently by two mathematicians working for French car manufacturing companies: Bézier, employed by Renault, and de Casteljau, working for Citroën (Shah & Mäntylä, 1995). They intended to create a mathematical form for a curve that would be easy and intuitive to modify (thanks to control points that define the curvature), so as to allow for experimentations by a user on a graphical system. Because de Casteljau was not allowed to publish his work that was kept secret by his employer (Farin, 2002), the curves are named after Bézier, who published extensively on the matter and even wrote a PhD dissertation (Bezier, 1977) based on his work at Renault.

Those developments mostly focused on two-dimensional sketching, even though there were some exceptions that also integrated some 3D capabilities (notably in Sketchpad itself).

Industrial adoption (1970's)

During the seventies, commercial 3D modellers grew into a more tangible reality, with systems running on workstations. CAD therefore became subject to a broader industrial adoption, although mostly limited to large companies.

More design applications were also considered in that period, including architecture but also Computer-Generated Imagery (CGI) for the entertainment (film) industry, with programs such as MAGI's SynthaVision, that allowed users to make 3D animations. A promotional video of the software's capabilities from 1974 can be found online[12] but the system was notably used, a few years later, for the original (1982) Tron movie, including most of its action sequences.

Overall, the CAD field matured in that decade, to a point where an overview of the state-of-the-art (Eaglesham, 1979) and the impact of the technology in the industry was published in a 1978 edition of the Computer-Aided Design journal.

In terms of important research developments during that era, one of the most prominent was the concept of Boundary Representation (B-rep) (Braid, 1975), a technique that allows for solid objects to be described (and therefore stored on a computer) using primitive solids that are moved, scaled or rotated, and then combined with or subtracted from one another, as shown with simple examples in Figure 2.15.

Similar ideas were developed in the same period, with Voelcker's paper

---

[12]https://www.youtube.com/watch?v=jwOwRH4JpXc

(i) Q_1, Q_2 positive and overlapping; result Q_p.

ii) Q_1, Q_2 positive and touching; result Q_p.

(iii) Q_1 positive, Q_2 negative and within or touching Q_1; result Q_p.

(iv) Q_1 positive, Q_2 negative and overlapping; result Q_p and Q_n.

Figure 2.15: Braid's addition (and substraction Q_2 is negative) of objects. Reproduced from (Braid, 1975).

(Voelcker & Requicha, 1977) often cited as the original reference for Constructive Solid Geometry (CSG). Both B-rep and CSG represent complex objects by combining primitive ones (with what is often referred to as boolean operations) but, whereas B-rep internally stores faces, edges and vertices, CSG directly stores the primitive solids.

Both systems are used today, sometimes jointly to take advantage of their respective benefits (Hoffmann, 1989), such as CSG's insurance that an object is always closed (and defines an interior volume) and B-rep's efficiency for graphical rendering.

Mass adoption and networking (1980-1995)

Based on earlier developments, the first version of AutoCAD was released in 1982. It was one of the first CAD systems to run as an application on microcomputers (as opposed to mainframes and minicomputers that had to be shared and accessed through terminals) and became very popular. The

availability of such software combined with the advent of (more affordable) personal computers lead to a much larger adoption of CAD tools during the eighties and the first half of the nineties.

Another key factor was the development of computer networks, which enabled better cooperation and data exchange. Many researchers were therefore developing specifications for exchange formats at the time (Wilson, 1987), with IGES (Initial Graphics Exchange Specification) (Smith, 1983; Smith & Wellington, 1986) being the most widely used. Those research advances led to a more concerted effort to develop a single standard specification, named STEP (Standard for the Exchange of Product Model Data) (Pratt, 2001, 2005). That specification eventually became an ISO standard, thereby reaching a wider industrial adoption.

In addition to these technical computer-to-computer communication advances, the early nineties also pushed CAD forward thanks to the invention of an alternative technique to draw curves using control points: the Non-uniform rational B-spline (NURBS). The use of that technique in CAD is hard to attribute to one particular individual since it results from the work of several separate researchers iterating over a concept that then lead to a later industrial adoption, but a PhD thesis in particular (Versprille, 1975) can be considered as seminal work on the subject.

While Bézier curves start and end at control points (respectively the first and the last ones) defined by the user, NURBS curves do not reach their first/last control points (they bend from/towards them). Although both approaches are rather intuitive to use, this can make Bézier curves slightly easier to handle, even though NURBS curves are more efficient for complex curves (in terms of computing power to calculate them). They can also exactly represent circles, whereas Bézier curves can only approximate them (on 3dbiology.com, 2018).

Even though Sutherland's Sketchpad could already apply certain constraints to drawings, the eighties saw **constraint-based design** arise as a paradigm. As is often the case with innovations, constraint-based design was further developed in a research context, with theoretical advances on how to enable variations of a (constrained) geometry (Hillyard & Braid, 1978; Light & Gossard, 1982).

Another paradigm also appeared in that period: **feature-based modelling**. It was first proposed by Pratt (Pratt, 1984) and was subsequently integrated into research prototypes (Cunningham & Dixon, 1988; Shah & Rogers, 1988). The idea of feature-based modelling is that the design is composed of features (such as a hole) that carry a semantic meaning instead of simply being

geometric shapes (e.g., a circle). These features can also impose constraints or define properties.

Both constraint-based design and feature-based modelling were adopted by the industry and became part of most commercial CAD systems and suites by the mid 1990s (Shah, 1998).

Maturation and collaboration (1995-2005)

Even though 3D modelling tools existed before, they truly started to shine and be used in the industry starting from the mid nineties (Baba & Nobeoka, 1998). The sophistication of these tools, with more features and improved user interfaces, combined with a larger availability of personal computers and better compatibility (of both the hardware and the software) led to a rise in usage. In (Asanowicz, 1999), the author considers that we may only talk about mature CAD starting from that period, which coincides with the first mass adoption of CAD tools (Horváth & Vroom, 2015).

Constraint-based design was integrated into commercial CAD tools such as Pro/Engineer, and enabled automatic solving of geometric constraints. Even though the tool started as yet another B-rep system, it received an update with sketch-based constraint definition during the nineties. This allowed designers to easily add constraints and annotate them to specify dimensional values. The sketches were then instantiated using a constraint solver (Hoffmann, 2005). Figure 2.16 shows the Sketcher interface in Pro Engineer Wildfire 2.0.

Together with feature-based modelling, constraint-based design led the way to a new paradigm: **history-based modelling** that is sometimes also referred to as **parametric modelling**. History-based tools remember the designer's actions on the features and allow for later modifications to the parameters of the modification, before replaying the steps that previously followed that action, to rebuild and update the geometrical representation accordingly.

Another paradigm was placed in opposition to history-based design: **direct modelling**. Since creating parametric relations requires specific training, direct modelling was presented as an easier-to-grasp alternative. The approach is also called history-free because the designer directly works on the geometry with no knowledge of previous actions and no relations to define. It is therefore easier and faster to produce results in early stages of design but is not necessarily the better option in the long-term, especially for "families of designs" (that differ only by a set of parameter values) or highly configurable models (Tornincasa et al., 2010).

The research on collaborative design that was carried out during the second half of the nineties and beyond was also pushed and helped by the tremendous

Figure 2.16: Sketcher constraints in Pro Engineer Wildfire 2.0 (2004). Reproduced from brighthubengineering.com[13].

growth of the internet. CAD systems therefore started to include better support for online collaboration.

CAD sub-specialisation and input/output enhancements (2005-now)

While each of the previous periods saw evolutions and breakthroughs that were crucial to the evolution of CAD systems, the beginning of the 21st century was mostly about consolidation and enhancements, in terms of Product Data Management (allowing better integration between different systems and therefore facilitating cooperation (Liu & Xu, 2001)), but also with regards to portability (with the advent of tablets and smartphones).

In the meantime, a plethora of specialised CAD tools were created, therefore extending the reach of computer-aided design to new domains. Specialised CAD tools were already available for fields such as mechanical engineering, architecture or electronics design, but CAD started to spread to other areas like life sciences. Examples include dentistry (Davidowitz & Kotick, 2011) and medicine in general (Bibb et al., 2014) but also specialisations that need to work at a miniature scale (often molecular or even atomic-level design), such as pharmacology (drug design (Macalino et al., 2015)) or chemical biology (e.g.,

---

[13]https://www.brighthubengineering.com/cad-autocad-reviews-tips/
22421-why-3d-cad-modeling-software-is-parametric/

Figure 2.17: A version of the CEREC software, used to design dental restorations and implants. By watching carefully, one can see the overlay of 2 models on the picture: the blue one is the scan result and the yellow one represents the model being worked on. Reproduced from (Davidowitz & Kotick, 2011).

protein design (Mandell & Kortemme, 2009)).

Another aspect that greatly evolved in the recent years is the integration with external data and tooling. As for the input side, 3D scanning and photogrammetry are increasingly employed, either to create a basis to work on (rough shape of an existing similar structure or even something the designer wants to replicate or reverse engineer) (Kuş, 2009), or simply in order to provide context to the design (e.g., surroundings) (Wolf et al., 2014). Figure 2.17 shows an example of a specialised CAD tool for dentistry, that also illustrates the use of scanned data to help the designer.

The popularisation of 3D printing technology led to an increase in their use for Computer-Aided Manufacturing (CAM), the next logical step in the production process for many CAD-enabled domains. For example, manufacturers have used the technology to make dental implants (Dawood et al., 2015) or even bone tissues (Bose et al., 2013). CAD tools have also evolved with regards to visualisation features, with photo-realistic renderings becoming more commonplace, for instance to better picture violations of requirements in autonomous vehicle safety assessment (O'Kelly et al., 2017) or to simulate coating appearance depending on lightning (Jhamb et al., 2020).

2.3.2   Computer-Aided Architectural Design

The evolution of **Computer-Aided Architectural Design (CAAD)** is mostly consistent with the history of CAD in general (Mitchell, 1989; Moubile, 2018), as expected since both fields can benefit from the same technological advances.

The concept of CAAD as a research area started to emerge during the 1970's, with overviews of the state of the art and the practice in that period, but is only truly recognised as a separate field in the 1980's (Koutamanis, 2005).

Generations of CAAD

A common way to structure the evolution of CAAD (as well as CAD in general) is to divide its history in generations of tools (Kale & Arditi, 2005) but many different divisions have been proposed and the suggested timelines often contradict one another. The three generations presented in this section are therefore a subjective summary based on our literature review.

The first generation would be computer-aided drafting, where the computer essentially serves as an alternative and potentially more precise tool to draw lines and shapes. Drawings are then stored in an electronic format and can therefore be easily modified and shared. Three-dimensional wireframe versions of such tools were also developed but the underlying design process did not fundamentally change.

A second generation of CAAD technology has, in a sense, raised the level of abstraction, by allowing designers to work with more complex objects and transformations. Regardless of which technology is used to store these objects, i.e., B-rep or CSG (see section 2.3.1), or even a combination of both, the geometrical representation is created using parametrised primitive (3D) shapes directly, that are then combined to construct a composite object.

A third generation comprises Computational Design methods, that will be further described in section 2.3.3. The general idea is that these methods allow architects to act on rules, constraints, or instructions that will generate or act on the designed geometry. The level of abstraction of the artefacts manipulated by the architect for that generation of tools is therefore even further raised, since the designer no longer has to act on the geometrical representation directly to produce changes.

It should be noted that each new generation of CAAD technology never entirely replaces the previous one. This is particularly true for the last two generations described, that are both prominent in today's architectural practice

and their respective benefits are often combined in the same (suite of) tools.

Building Information Modelling

An important milestone in CAAD history is the release of Autodesk Revit in 2000. The software helped popularise BIM in the mid-2000s (Moubile, 2018; Azhar et al., 2012), although the conceptual basis dates back to the 1980's (Aish & Bredella, 2017) with the Building Description System (Eastman, 1975). BIM describes both the software and the process that encourages a better integration of stakeholders on a project in the AEC sector (Azhar et al., 2012). These stakeholders include owners, designers, constructors, engineers, contractors, suppliers and facility managers. The core concept is that a project consists of 3D models that are interlinked, shared with all stakeholders, and connected to the information from these stakeholders and the different phases of the project's development. That concept is depicted in Figure 2.18.

Figure 2.18: The BIM concept. Reproduced from (Kymmell, 2008).

Using the technology tends to lower the risk of inconsistencies and errors, but also helps provide semantic meaning to building elements. In a way, the concept of BIM relates to the Enterprise Resource Planning (ERP) technology in use in many businesses, where the company would be the construction project, and the departments and divisions would correspond to the project's stakeholders. They all share a common database where everyone is able to access and modify his part.

2.3.3 Programming architectural models

Parametric definition of geometries and mathematical expressions to describe volumes clearly predate the computer era. The work from Antoni Gaudí at the turn of the 20th century, in particular the Colònia Güell, is also often cited as an early occurrence of algorithmic design thinking. This is likely due to the innovative way the architect realised the form-finding step of this architectural masterpiece: through chains hung from a ceiling of other chains and burdened with small weights, depicted in Figure 2.19, so as to let the laws of physics curve the chains to distribute the load evenly. Once flipped upside-down, the resulting shapes define a structure of arches superimposed on each other. If a chain anchor's point or load changes, the entire structure is changed by the natural optimisation process. This resembles the way AD parameters impact the whole geometry and may explain why Gaudí's work is sometimes associated with AD, although it would likely be more accurate to talk about "analog computing" when mentioning that sort of natural optimisation.

Figure 2.19: Gaudí's hanging model and resulting rendered form. Reproduced from (Maher & Burry, 2003).

Even way before these constructions were designed, during antiquity, civilisations were using mathematical patterns and even rule-based (i.e., algorithmic even though no computer was involved) methods for architecture and art, with Islamic patterns (Agirbas, 2020) coming to mind as a primary application of these concepts.

The CAAD tools we will talk about in this section are more heavily tied to the concept of programming, with the explicit existence of an algorithm

```
!Domino House demo
!For... Next Loops
!3D Script

PEN 1: MATERIAL matl
depth=bdep*bays+clsz
width=bwid*bays+clsz

!Do all the floors
FOR m=1 TO snum
 ADDz ftfh*(m-1)

!Do one floor
BLOCK width,depth,clsz

!Do all the columns
FOR n=1 TO bdep+1
  ADDy bays*(n-1)

!Do 1 row of columns
FOR k=1 TO bwid+1
  ADDx bays*(k-1)
BLOCK clsz,clsz,ftfh
```

```
  DEL 1
 NEXT k
   DEL 1
  NEXT n
    DEL 1
   NEXT m

!Do the Roof
  ADDz ftfh*snum
BLOCK width,depth,clsz
  DEL 1

!Glass
MATERIAL "Glass"
  ADD clsz/2,clsz/2,0
BLOCK width-clsz,
   depth-clsz,ftfh*snum
  DEL 1
```


Figure 2.20: A Geometric Description Language (GDL) script to generate a domino house. Reproduced from (Nicholson-Cole, 1998).

that can be edited. This comes as a profound paradigm change that may not be natural for conventionally-educated architects, since they have to learn to "think algorithmically" i.e., to decompose a design idea into a set of instructions that are simple enough for a computer to execute. These instructions can be processed with variable parameter values or even ignored depending on specific conditions declared by the designer-programmer through what is called flow-control mechanisms. This is one of the main benefits of using such algorithm-based design tools: a plethora of possible (geometric) solutions can be generated by these algorithms, allowing the designer to explore a variety of different outcomes. Depending on the particular project on which it is used, designing with algorithms can reduce human errors (Burry, 2011) and costs (Woodbury et al., 2010).

Different representations: textual vs visual

Programming is typically carried out through (advanced) text editors, at least for professional developers, since it seems to be the appropriate choice for productivity and scalability (Myers, 1990). That being said, designing with algorithms does not come naturally for traditionally-trained architects. While an increasing number of university curricula focus on, or at least include, courses about AD, architects are not usually programmers. An architect starting to learn AD through a textual language therefore first has to get familiar with programming concepts such as variables, functions and scopes, in addition to the language's syntax.

Figure 2.20 depicts an example in GDL, a script language for ArchiCAD,

```
(defun C:ro ()
      (command "erase" "all" "")
      (setq  ang 0
             n 0)
      (repeat 12
            (setq  p1 (list 0 0 n)
                   p2 (list 2 4 (+ 1 n))
                   thing (solbox p1 p2 "")
                   n (+ n 1))
            (command "rotate" thing "" p1 ang)
            (setq  ang (+ ang 30))
      )
)
```

Figure 2.21: An AutoLISP script to generate a spiral staircase. Reproduced from (Coates & Thum, 1995).

while Figure 2.21 shows a staircase model in AutoLISP (for AutoCAD). Both languages were released in the 1980's.

Because of the difficulty to learn textual programming for non-programmers, **Visual Programming Languages (VPLs)** have been developed for architectural design and are more popular than the textual alternatives in that context (Sammer et al., 2019).

The most popular ones as of today include GenerativeComponents, Dynamo Studio and Grasshopper. An example created with the latter is depicted in Figure 2.22.

All these tools aim to make AD more accessible by providing an intuitive representation of the algorithm that better satisfies the visual nature of architects. This makes the learning curve more gentle, although it comes at the cost of a reduced scalability. In fact, contrary to the common adage that "a picture is worth a thousand words", VPL models tend to become hard to understand and manipulate (Leitão et al., 2012) when they grow in size and complexity. This is partly due to the lack of advanced abstraction mechanisms that tends to induce redundancy in the visual code (copying-and-pasting parts of an algorithm).

Furthermore, many of these visual languages do not natively support recursivity[14]. The choice is understandable since they usually target non-programmers that would need to get familiar with the concept first and could easily end up

---

[14]Such functionality is however usually available within AD solutions, using plug-ins or through a textual programming language supported by the solution.

Figure 2.22: Example of a Grasshopper definition for a conical spiral. The spiral (output by the PLine component) is created from a list of "vertices" (input parameter V) that are effectively points (Pt), whose coordinates are defined by evaluating (Eval components) two formulas on a list of values, themselves generated from a Range component. Reproduced from (Leitão et al., 2012).

with infinite loops in the meantime, but may further increase the scalability issue in certain instances.

In an attempt to benefit from the best of both worlds, some editors (in fact, most of the popular AD software tools) enable hybrid programming approaches, that rely on a combination of visual and textual representations. In most cases, the visual language is used to define the outline of the algorithm (i.e., the overall logic on how to construct the target geometrical representation), using some nodes or blocks that themselves may contain code in textual form.

Programming paradigms

Another way to classify the plethora of AD tools that have been created is by grouping them by the paradigms of their underlying programming languages. Such a taxonomy is presented in (Appleby, 1991) and further depicted in (Dave et al., 2013), where it is annotated with examples of programming languages and AD tools made for or at least used by architects. Figure 2.23 presents a reworked and updated diagram of that taxonomy, with a few examples of currently popular or historically-relevant AD tools and programming languages.

On the left-hand side, the red circle represents the set of **imperative languages** i.e., languages that focus on how a program has to operate, through

Figure 2.23: Algorithmic Design tools and programming languages (adapted from (Davis, 2013) and based on (Appleby, 1991))[15].

ordered statements that alter the state of a program. Within that set, procedural programming is a subtype of imperative programming that relies on functions (or procedures) to split a program into separate pieces that can be called from different parts of the overall program. Examples of AD languages that fall into this category are the previously mentioned GDL for ArchiCAD as well as MEL (Maya Embedded Language), that both use control flow statements (e.g., IF/ELSE) to conditionally decide which code blocks to execute and loops (e.g., FOR) to execute the same code block a specified number of times (typically until a certain condition is met).

Another subset of imperative languages is the set of object-oriented programming languages, that rely on the notion of objects that contain data and behaviour, typically through object-specific procedures that explain why object-oriented languages are very often also procedural languages. These objects help package specific functionalities with the relevant data states, and isolate parts of the code that are unrelated. An example of an object-oriented AD language is Maxscript for 3DS Max, but popular general-purpose programming languages (broadly used outside of AD) such as C# and Python can also be used with many CAAD tools. In fact, these languages are supported by

---

[15]The image is an intentionally oversimplified view of the relations between programming paradigms. Many hybrid languages have been created and integrate features from paradigms that do not overlap in the figure, including AutoLISP that is in fact mentioned twice, in both the procedural and the functional sets.

major solutions such as Rhino, AutoCAD, Maya, Houdini, Dynamo Studio or Revit. Some AD software tools, such as Luna Moth (Alfaiate et al., 2017), Möbius (Janssen et al., 2016) and PIM (Maleki & Woodbury, 2013), also rely on ad hoc programming languages that support that paradigm.

On the right-hand side of Figure 2.23, the blue circle is for **declarative languages** i.e., languages that focus on the logic of the program without controlling the flow of execution. A common simplification of the principal difference between declarative and imperative languages is that the declarative paradigm focuses on *what* to accomplish instead of *how* to do so, as would be the case for the imperative paradigm approach. Within the declarative circle, we find **functional languages**, where programs are created by composing functions. That paradigm has its roots in lambda calculus (Church, 2016), a formal system that only uses functions for computation.

Often linked to functional programming (because it shares some common characteristics) is **dataflow programming** (Johnston et al., 2004), where a program can be modelled as a directed graph that lets data flow along the edges after being processed by the graph's nodes. Although there are textual dataflow programming languages, their inherent capability to be represented as graphs has lead to the creation of many visual languages based on that paradigm. As for AD, the main software tools in use today do rely on visual languages that integrate elements from both functional and dataflow programming. These include Grasshopper, depicted in Figure 2.22, but also GenerativeComponents and Dynamo Studio.

Very often, modern programming languages integrate features from multiple paradigms. One such example as far as AD is concerned is AutoLISP and its enhanced version Vital LISP (later renamed Visual LISP), the AutoCAD scripting language that is a LISP dialect. Figure 2.21 shows an example of AutoLISP code that produces a spiral staircase.

It should be noted that most popular software tools integrate multiple programming languages. Most of the time, the core skeleton of an AD model is a visual dataflow-based program, but sometimes specific components of that program can be written in another programming language supported by the software. This gives the designer access to multiple paradigms and representations, and he is free to choose the tool that best fit his design intent and/or abilities.

2.4. Three-dimensional Human-Computer Interaction

As made clear by its name, **Human-Computer Interaction (HCI)** is inherently an interdisciplinary domain, that combines knowledge from many disciplines. Our focus here will be on the technical (computer) side of HCI. The relevant literature in psychology, human factors and ergonomics therefore will be considered out of scope, although the interested reader may find good references on these subjects in (LaViola, 2017, chap. 3), (Salvendy, 2006), (MacKenzie, 1992), and (MacKenzie, 2012, chap. 2).

Interaction techniques are the necessary bridge between the user and the application's interface, they are how that user communicates intent to the system. In the world of two-dimensional applications, most interfaces rely on the Windows, Icons, Menus and Pointers (WIMP) metaphor and users typically interact with the system through a mouse and a keyboard. Since there is no such well-established standard for three-dimensional environments, interaction techniques and user interfaces adapted to that context can take many forms and rely on various input devices.

This section covers such techniques, focused on the most basic manipulation tasks we need: selection, positioning and rotation. The section's structure is dictated by a common taxonomy (LaViola, 2017) that classify techniques depending on whether they offer direct interaction with the target entity.

2.4.1 Direct manipulation

Amongst direct manipulation techniques, one way of categorising them is based on isomorphism: **isomorphic** approaches preserve a natural one-to-one mapping between input actions and their resulting effect, whereas **non-isomorphic** techniques afford non-realistic interactions and can even be based on "magical" or "virtual" tools.

That being said, **direct** manipulation techniques very often either rely on (A) a *touching* (grasping) metaphor or (B) a *pointing* metaphor. The organisation of this section is therefore rather based on the metaphor being used.

(A) Grasping metaphor

As for techniques based on touching (or grasping), the user must reach the target object's position to interact with it. This is the most natural and intuitive way of interacting within three-dimensional environments since this is how human naturally interact in the real world.

Typically, grasping-based techniques rely on a virtual hand (or assimilated) that is mapped to a tracked object in the real world, that we will refer to as "the real hand" in the remaining of this section. A simplistic isomorphic example of such a setup would be to track a physical controller with six degrees of freedom and map its position and orientation to the virtual hand. If the physical controller has actionable buttons, they could be used to trigger the grasping of objects that collide with the virtual hand, so that they can be manipulated and then released (e.g., when releasing the button). Figure 2.24a depicts a similar example, with the user's real hand (in blue) overlapping the virtual one (in green) to touch a yellow ball.

When the tracking zone is limited in space compared to the "interactable" area, non-isomorphic mappings are typically used to mitigate these limitations. Figure 2.24b presents that idea, with the real hand's movement being amplified to change the virtual hand's position, so that it can reach the yellow ball even though the distance from the user would not allow such interaction in the real world (should the yellow ball have a physical counterpart in the same position). The Go-Go (Poupyrev et al., 1996) technique is an example that only amplifies the virtual hand's motion when the real hand's distance to the user's body exceeds a predefined threshold. On the other hand, different techniques such as PRISM (Frees & Kessler, 2005) also rely on a non-linear mapping between the virtual hand and the real one's motion, but rather choose to scale down the virtual hand' movement when the real hand's motion is close to the user's body, to allow for more precise manipulations. Note that, in the case of that technique, an offset recovery is also present when the real hand's motion exceeds a given threshold, so as to allow for the virtual hand to catch up on the real hand. There is additionally a buffer zone in between those two threshold so that movements whose speed falls in that range can provide a direct (one-to-one) mapping between real and virtual motions.

(B) Pointing metaphor

The other metaphor commonly encountered is the pointing metaphor, that naturally mitigates space-related limitations, since aiming at an object becomes sufficient to start interacting with it. Similarly, the user is able to reposition that object by pointing towards a target position. Typically, the position and orientation of a tracked controller, that we will refer to as "the wand" in the remaining of this section, defines a "laser beam" that selects and manipulates objects.

Techniques based on that metaphor differ in how the position and orientation of the wand affect the laser beam, and which object(s) are selected based

(a) Grasping metaphor (isomorphic)   (a) Grasping metaphor (non-isomorphic)

(c) Pointing metaphor (curved)   (d) The *spotlight* technique (fixed α)

Figure 2.24:   Examples of direct interaction techniques, relying on different metaphors to select a yellow ball.

on that beam. The most common version is often called ray-casting (Poupyrev et al., 1998), where the wand defines a simple line segment and that segment's intersection with the environment defines the target object or position. More complex techniques allow users to bend the line to mitigate limitations related to occlusion, as shown on Figure 2.24c, where selecting the yellow ball occluded by an object becomes possible thanks to the curve. There are many ways of bending the beam based on the wand's position and orientation, but most techniques rely on Bézier curves (e.g., the flexible pointer (Feiner, 2003)).

To select distant objects more easily or in order to enable multi-objects selection, volumes can be used instead of simple rays. The selection volume size can be static (e.g., the spotlight technique (Liang & Green, 1994) that relies on a selection cone, as pictured in Figure 2.24d) or dynamic e.g., the aperture technique (Forsberg et al., 1996), that expands on spotlight by allowing the user to control the spread angle of the cone by changing the distance from the eye to the wand (the angle gets bigger as the wand gets closer, as if the wand served as an open cylinder that would select everything the user can see through it).

More complex techniques that rely on two rays also exist, such as iSith (Wyss et al., 2006), depicted in Figure 2.25. That technique uses a "projected intersection point" that is placed at the middle of the shortest line segment between both rays, that is then used as the interaction point ("cursor") to select and move objects when pressing a button.

Figure 2.25: The idea behind the iSith bimanual interaction technique, with an example where a yellow ball is moved to a new location by rotating both wands.

Other comparable techniques that add support for controlling more degrees of freedom have been proposed, "Spindle + Wheel" (Cho & Wartell, 2015) that allows users to control with 7 degrees of freedom (3 positional axes, 3 rotational axes, and a global object scaling control) and is based on previous work (Mapes & Moshell, 1995) that proposed a similar approach that did not offer pitch control (one of the three rotational axes).

2.4.2 Indirect manipulation

Many other interaction techniques allow users to manipulate objects without directly interacting with them in the virtual environment. These may not be as natural as direct manipulation but provide alternative solutions to mitigate spatial and occlusion constraints so as to enable distant interaction.

One option is to rely on proxies, i.e., miniature representations of remote objects often located close to the user. Actions on a proxy are mapped to the proxy's full scale counterpart. This typically enables direct manipulation of the proxy, e.g., based on grasping techniques, but can produce an effect on a distant object. A well-known technique that relies on this principle is WIM (World In Miniature) (Stoakley et al., 1995), that provides the user with

a hand-held scale model of the virtual environment, similar to the maps or radars that are part of many game interfaces, but in three dimensions.

A more advanced proxy-based technique is Voodoo Dolls (Pierce et al., 1999) that first asks the user to select one or multiple objects to interact with, through a pointing-based technique. Clones (dolls) of the selected objects are created and scaled down, then placed into the user's non-dominant hand. When the user grabs a doll with its dominant hand, the manipulation of the doll (and its full scale counterpart) can begin, with motions relative to dolls in the non-dominant hand. Figure 2.26 shows that technique being used to interact with a nutcracker and a pin.

Figure 2.26: The Voodoo Dolls proxy-based indirect interaction technique. The blue pin can be moved relative to the nutcracker, since they were both selected and the pin was then grabbed by the user's dominant hand. Reproduced from (Pierce et al., 1999).

Another alternative to enable indirect manipulation is to make use of widgets. Such widgets are extremely common in desktop-based design tools for many domains (e.g., architectural design software or game engines), where the user can generally manipulate an object's position, rotation, or scale, based on handles placed in the target object's close environment. One handle typically only controls one degree of freedom but three-dimensional interfaces can also allow a widget to control two degrees of freedom at the same time. While controlling a third degree of freedom would be possible with 3D-tracked con-

trollers, the widget is then only limited to a starting or anchoring role, and the technique therefore becomes more of a grasping-based one (potentially with an offset).

2.4.3   Hybrid techniques

As with tracking technologies, hybrid interaction approaches have been proposed to combine the advantages of multiple types of techniques and circumvent their individual shortcomings. There are two ways to hybridise such techniques: either the user (or the system itself) can select the appropriate manipulation technique at any point in time, or specific techniques are assigned to specific tasks (or stages in the interaction).

An example of the latter is to perform the selection of an object through a raycast-based technique that teleports a virtual hand to the target location when the object is selected, before manipulating the target object using a grasping-based technique with scaled motion, similar to the previously mentioned Go-Go technique. (Bowman & Hodges, 1997) proposed such a combination of techniques with HOMER (Hand-centered Object Manipulation Extending Ray-casting). That method has been adapted to handheld mobile AR with HOMER-S (Mossel et al., 2013), whose concept is depicted in Figure 2.27.

Figure 2.27:   The HOMER-s hybrid manipulation technique, that combines both raycast and grasp-based approaches for handheld-AR interaction. Reproduced from (Mossel et al., 2013).

2.5. New ways of interacting with architectural designs

As discussed in Section 2.3, recent advances in various technologies have provided CAD software with new opportunities to interact with models. Architectural design is no exception and the advent of innovations such as 3D printing technologies have led to academic and industrial adoptions. Researchers have explored different uses of the technology, ranging from small-scale educational prototyping (Greenhalgh, 2016) to large-scale industrial constructions (Gardiner, 2011; Wu et al., 2016).

Another recent technology that is being used extensively in architectural practice is photogrammetry, which can be used to document specific buildings for cultural heritage purposes (Hanan et al., 2015) or to provide a basis on which construction planning activities can be performed (Liu & Xu, 2001).

While the previous examples described potential enhancements to the design and construction processes that were "external" to the modelling tools themselves, work has been done to increase interactivity with existing software tools, as predicted in (Monedero, 2000).

This is the case for various stages of the design process. As for the conceptual design phase, where designers typically rely on sketching tools, the integration of generative design techniques has lead to systems such as DreamSketch (Kazi et al., 2017) that generates three-dimensional geometries based on free-from sketches. When sketching with that system, the designer creates objects that can be labelled as *interface*, if they are part of the geometry, or *obstacle*, if they are not to be included in the generated solution. The user can then define *variables* by moving an object along the edges of the polygon that encloses that object's possible positions, and the system uses that information to generate potential solutions. Similar automated generation techniques have also been used for more advanced modelling activities, often through add-ons or plug-ins extending the functionality of popular modelling tools.

Section 2.5.1 discusses how exploratory and optimisation-based methods have been applied to architectural design, while Section 2.5.2 overviews the current use of immersive technologies in the field.

2.5.1   Design Space Exploration and Optimisation of parameter values

The general concept of the AD paradigm, consisting of generating geometry through an algorithm, already leads to a fundamental change in the way ar-

chitects interact with their design, since they mentally switch from "modelling an object" to "modelling the logic that generates an object". The parametric nature of AD definitions allows the designer to explore different solutions by adjusting these parameters. By integrating performance assessment tools into AD software, the architect becomes able to iterate over parameter values until satisfied by both the generated design itself and its computed performance (e.g., via thermal or airflow analyses). Figure 2.28 represents a typical instance of such a performance-based design exploration process, likely for an airflow analysis on a building, created with Grasshopper.

Figure 2.28:   A typical performance-based design exploration process where parameters (a) drive an AD definition (b), that generates a geometrical representation (c) on which performance assessment tools can be run (d), so as to adjust parameter values (e) and iterate over the design. Reproduced from (Harding & Shepherd, 2017).

It is even possible to go one step further and rely on artificial intelligence and optimisation techniques to close the loop, i.e., to let such techniques fix or improve at least part of the algorithmic (logic) or parametric (values) definitions. The process of using such techniques to explore solutions and find the best designs is called **Design Space Exploration (DSE)**.

The output of such techniques are typically optimised according to one or more target metrics, and it would consequently be easy to assume that the geometries that have been generated that way would be used as the final result. However, while this may indeed be the case for certain design teams, others only use these "optimal" models as starting points over which they can then iterate (Bradner et al., 2014).

It is important to realise that, even when only parameter values are being adjusted by such techniques, the design space very quickly becomes immense due to its combinatorial nature, since n parameters with m possible values each leads to m^n possibilities. Even if m is finite, n leads to an exponential growth

and it therefore rapidly becomes impossible to explore the whole design space in a reasonable time, when many parameters are involved.

For this reason, DSE and optimisation-based approaches often rely on approximation algorithms, such as metaheuristics (Talbi, 2009) (general procedures attempting to find a good solution without exploring the whole solution space). While a large body of work on DSE is linked to microprocessors and integrated circuits in general (Xie et al., 2006), similar approaches have been used for architectural design activities, and can help focus on a given criterion or a combination of criteria, using multi-objective optimisation techniques.

Space layout

A common application of these techniques is in space layout, where they produce automated facility, office, or housing arrangements. Methods based on single-parameter optimisation techniques (e.g., to minimise the cost of product flow between departments (Buffa, 1964)) or graph theory (e.g., constructing a planar graph with a hexagonal structure to generate a rectangular block plan (Goetschalckx, 1992), as shown on Figure 2.29) produce satisfying results on some instances of the space layout problem, but more complex techniques are necessary when multiple constraints or criteria are involved (Liggett, 2000).

Figure 2.29: Generation of a block layout based on a technique using planar graph. Reproduced from (Liggett, 2000) as per (Goetschalckx, 1992).

More recently, evolutionary algorithms have gained a lot of popularity for space layout problems, especially **Genetic Algorithms (GAs)** (Calixto & Celani, 2015). This is likely due to two main factors working in their favour: their ability to handle multiple objectives and their inherent capacity to generate multiple solutions that the designer can choose from. In fact, an evolutionary algorithm naturally generates a new population of solutions at each iteration. That population results from crossovers and mutations on the best

solutions (according to a fitness function that can accommodate for many parameters) from the previous generation.

Many examples of uses for GAs solving space layout problems can be found in the previously cited (Calixto & Celani, 2015) as well as in (Turrin et al., 2011) or even in commercial solutions such as Spacemaker[16]. Such tools can accommodate a plethora of criteria, such as energy efficiency (Wright & Farmani, 2001), thermal comfort (Chen et al., 2008), or adjacency preference (Nagy et al., 2017b). An extension to the latter has incorporated survey results to the generative process (Villaggi et al., 2018b). In fact, they integrated user satisfaction based on a questionnaire to that process, in order to enable certain occupant-level goals, that are usually disregarded, to be used.

Other techniques based on machine learning, and neural networks in particular, have been used for similar purposes, often based on **Generative Adversarial Networks (GANs)**, a technique first proposed in (Goodfellow et al., 2014) that involves two competing neural networks: one that generates candidate solutions while the other evaluates them. GANs have been used to generate bedroom configurations (Radford et al., 2015) as well as floor plans, based on coloured drawings that represent areas with different functions (Huang & Zheng, 2018), or simply based on a given parcel used as input for a three-stage pipeline (Chaillou, 2019) (building footprint outlining, room splitting, furnishing) represented in Figure 2.30.

Figure 2.30: Generation of a floor plan using a three-stage pipeline. Reproduced from (Chaillou, 2019).

Morphogenesis

Another modelling application for DSE techniques is morphogenesis, i.e., the generation of form. For instance, (Caldas & Norford, 2003) proposed to use

---

[16]www.spacemakerai.com

GAs to produce building envelopes, based on factors such as ventilation, lighting and construction costs. Similar work includes envelope generation that searches for the optimal shape of a tower, to maximise solar radiation while satisfying geometric constraints (Besserud & Cotten, 2008), or to provide the best possible views (Doraiswamy et al., 2015).
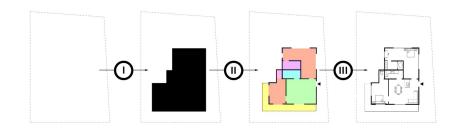
Morphogenic approaches based on deep learning have also been proposed, with (As et al., 2018) exploring the use of different types of neural networks to generate houses.

Smaller parts of a design are also covered by GA-based approaches, with (Turrin et al., 2010) addressing the generation of roofs that offer good daylight and thermal comfort, (Choi et al., 2014) designing louvres based on thermal performance, or (Ercan & Elias-Ozkan, 2015) exploring shading devices to be placed on facades to optimise daylight while avoiding excessive solar heat gain.

Although generally rather linked to engineering than architectural design, many researchers have integrated these techniques to structural design (i.e., the creation of stable structures capable of resisting loads and various forces being applied on their parts). For example, (Kaveh et al., 2008) relied on an ant colony algorithm (a metaheuristic that is inspired by the behaviour of ants using pheromones to converge towards better solutions) to generate stiff structures given a certain quantity of materials.

A more interdisciplinary work was proposed in (Miles et al., 2001), whose authors created a system to be used by both engineers and architects. That system uses a GA to produce floor plans and determine the layout of columns based on criteria such as lighting or ventilation, but also structural performance. Similarly, (Mueller & Ochsendorf, 2015) combined structural performance with designer preferences using an evolutionary algorithm.

The latter example integrates an interactive aspect, since the designer selects the best solutions for each iteration of the evolutionary algorithm, thereby "piloting" the exploration of the design space. This is also the case for (Marin et al., 2012) who have used so-called Interactive GAs, to produce buildings that both rely on energetic performance optimisation and designer preferences. A similar approach has also been proposed for urban landscape design (Koma et al., 2017). Commercial solutions integrating such concepts are also available, such as One Click LCA[17] that can allow Grasshopper designers to minimise the carbon footprint of a building (Apellániz et al., 2021).

---

[17]`www.oneclicklca.com/fr/parametric-and-generative-carbon-optimisation`

2.5.2 Use of immersive technologies

In addition to these automated approaches to generate designs, innovative "input" interaction devices have also increasingly been integrated to CAD software tools. This includes hand or body tracking devices that enable gestural interfaces for conceptual design (Khan & Tunçer, 2019) or even more advanced modelling activities (Dave et al., 2013), but tangible user interfaces have also been created to facilitate collaboration as part of architectural design activities (Gu et al., 2011), for various purposes such as daylight simulation control (Nasman & Cutler, 2013), or simply to move a geometry (Abdelmohsen & Do, 2007).

This section will rather focus on advanced "output" devices and immersive displays in particular.

As mentioned in Chapter 1, the current use of AR and VR technology in the AEC sector is mostly limited to visualisation purposes (Blach et al., 1998), often with marketing goals in mind, i.e., to help convince and sell a product to a client (Juan et al., 2018). This ability of immersive displays to facilitate the understanding of three-dimensional volumes for laymen is indeed one of the main benefits that has been pointed out.

It has in fact been demonstrated that such technologies improve the understanding of objects (Louis et al., 2020) and space (Schnabel & Kvan, 2003). Additionally, several studies (Johnson, 1963; Paes et al., 2017) have shown that the spatial perception is even enhanced as the immersion deepens. Therefore, as envisioned and described in (Brooks Jr, 1987), immersive walkthroughs using these technologies favour the dialogue between stakeholders (Coroado et al., 2015; Lo et al., 2019) (e.g., involving designers, constructors and clients), leading to better decision-making and easier detection of issues (Roupé et al., 2016), and helping to move towards user-centred design, leading to better (accepted) designs (Mobach, 2008).

Design software usually comes with features to export the geometry being worked on. Most major CA(A)D tools in fact support exporting to the OBJ and/or FBX formats, as they are amongst the most common options found on the market. Both formats can be used as a basis to create an immersive experience quite easily with popular game engines such as Unity[18] and Unreal Engine[19]. With the growth of the demand for immersive productions, a number of tools were developed to facilitate and (semi-)automate the creation of such experiences, especially for VR. Some of these tools allow users to have control over environmental factors (sun position, daylight intensity) or even provide

---

[18] `www.unity.com`
[19] `www.unrealengine.com`

access to extrinsic properties of the rendered design, such as texturing, scaling, or positioning options. However, most of the currently available tools do not enable changes on the form itself and should rather be seen as inspection or showcasing tools.

Design activities tend to follow a well-known sequence of processes. Based on a literature review in both engineering design and cognitive psychology, (Howard et al., 2008) describes the usual design process as a 4-stage endeavour: (1) analysis of the task at hand, (2) conceptual design, (3) embodiment design, and (4) detailed design. That process, illustrated in Figure 2.31, also allows designers to go back to previous stages until they reach stage (4). The remaining of this section will discuss existing work on modifications to the model itself, at different stages of the design process.

Figure 2.31: The stages of the design process, according to (Howard et al., 2008).

Sketching and sculpting

During the conceptual phase, a designer explores different ideas and traditionally relies on hand drawings and scale models. Their digital and three-dimensional equivalents are free-form sketching and what is sometimes called sculpting, often based on voxels, that allow users to spawn or remove material at the pointer's location. The corresponding tools are not sufficient to produce actual CAD models, but provide the opportunity to explore design options in 3D environments during the early phases of the process.

Early examples include (Kameyama, 1997) that is based on a virtual clay metaphor allowing the designer to add or remove some of the clay, and the more advanced (Wesche & Seidel, 2001), based on line drawings but with support for creating surfaces based on these lines. Both of these systems rely on 3D-tracked controllers and stereoscopic HMDs used at the time.

Contemporary commercial applications following similar approaches but

---

[20]www.gravitysketch.com
[21]www.tiltbrush.com

involving modern HMD and input devices have been released, including Gravity Sketch[20] and Google Tilt Brush[21], that was selected in (Arora et al., 2017) to evaluate sketching in VR with a surface to support the drawing.

As for research contributions more focused on architectural design, (Donath & Regenbrecht, 1995) presented voxDesign (Donath & Regenbrecht, 1995), a VR sketching tool for early phases of the design process relying on a 3D-tracked controller, while (Dorta et al., 2016) more recently proposed Hyve-3D, another conceptual design system that suggests using a tablet to control a 3D cursor that moves around and enables sketching, as depicted on Figure 2.32.

Figure 2.32: The Hyve-3D VR sketching system. Reproduced from (Dorta et al., 2016).

Direct modelling

Slightly more advanced modelling capabilities were added to solutions such as (Butterworth, 1992), (Billinghurst et al., 1997) and (Chu et al., 1997), that all enable basic shapes to be created from a VR application. Other systems were proposed more recently with architectural modelling in mind during development, including (Mine et al., 2014) and (Innes et al., 2017), illustrated in Figure 2.33.

Commercial applications replicating such an approach are Microsoft Maquette[22] and Google Blocks[23], that both offer the essential functionality to create low-complexity geometries from a VR-based environment.

However, the aforementioned tools lack support for some usual CAD fea-

---

[22]www.maquette.ms
[23]arvr.google.com/blocks/

tures and cannot truly be considered as the VR equivalents or at least the companions to modern desktop-based tools, and are therefore confined to a conceptual design role.

Figure 2.33: A VR modelling system that allows the user to create basic shapes and manipulate them with a selection of tools. Reproduced from (Innes et al., 2017).

More advanced VR modelling systems that integrate with popular CAD software have been developed, including the MARUI[24] plug-in for Maya[25], and Mindesk[26], compatible with Rhinoceros[27]. Besides enabling interoperability between a VR interface and a desktop CAD application, these solutions offer more complex design features than the systems we previously mentioned, e.g., to manipulate NURBS curves.

Algorithmic Design

Algorithmic Design is also covered by immersive solutions, to a certain extent. Generating a VR (or even AR) experience can be done quite easily using a game engine but, in order to do so, one must typically first export the design to a specific format to allow the game engine to load the corresponding file and import it into a virtual world. Some tools facilitate that process, such as Twinmotion[28] and IrisVR Prospect[29] by providing "one-click" exports from well known 3D modellers, such as Rhinoceros and Revit[30]), to a VR environment that includes the up-to-date geometrical representation. They therefore do smooth out the burden of creating such experiences, but still provide limited control over the design artefact itself (what we called extrinsic properties

---

[24]`www.marui-plugin.com/marui4`
[25]`www.autodesk.com/products/maya`
[26]`https://mindeskvr.com/`
[27]`www.rhino3d.com`
[28]`www.unrealengine.com/twinmotion`
[29]`www.irisvr.com/prospect`
[30]`www.autodesk.com/products/revit`

Figure 2.34: Immersive AD landscape in 2017: collaboration scenarios and levels of control (defined in details in Section 3.1).

in Section 2.5.2). Such tools can be used with some AD editors, since they generally rely on — or easily connect to — general modelling systems that these tools support. But their usage remains limited to visualisation purposes.

These types of tools only provide extrinsic control over the designed model, but deeper control over AD definitions is possible and will be discussed in the remaining of the present section, then further described in Section 3.1. Figure 2.34 clarifies what the state of the practice was at the start of our research work (in 2017), with regards to immersive control over AD models, and immersive collaboration scenarios.

Some proposals however enable manipulation of key control points employed by an AD algorithm, e.g., (Kwieciński et al., 2017) that uses tangible items as part of an AR experience. Other systems allow VR users to remove elements from a structural analysis calculation, from a VR environment that highlights geometry parts that do not meet pre-specified requirements (Moubile, 2018). These types of interactions can be subsumed as input control, since they do not interact with the internals of the algorithm but allow for changes to the entities it uses.

Another set of tools that fall into the same category lets users adjust an AD algorithm's parameter values. Such immersive adjustments are made possible

in a prototype presented in (Hawton et al., 2018), depicted in Figure 2.35, that relies on the Oculus Rift[31] VR HMD. Such a feature is also covered by (Moubile, 2018) that works with the same headset. In both cases, the user is presented with a panel, attached to a standard VR controller (tracked in 3D). The panel contains a list of parameters, coming from a Grasshopper definition, whose values can be changed. They both support number parameters, tweaked through the manipulation of sliders. Changes made to the parameter values are sent back to Grasshopper so as to modify the generated geometry whose updated version is, in turn, relayed back into the VR environment.

A commercial solution, Fologram[32], includes the same functionality but targets AR displays such as Microsoft's Hololens[33] and can even work with simple smartphones. The user can therefore visualise a geometrical representation while manipulating the parameters driving the (re)generation of that geometry, with no need to leave the augmented experience to do so. Note that these tools are directly related to the prototype we will describe in Section 3.2; they were developed simultaneously yet result from entirely independent works.

Figure 2.35: A VR application that allows for parameter adjustments to an AD definition. Reproduced from (Hawton et al., 2018).

---

[31]`www.oculus.com/rift`

[32]`www.fologram.com`

[33]`www.microsoft.com/hololens`

The aforementioned solutions cannot be used to add or remove components, nor do they allow to interact with the links between these components. To overcome that limitation, researchers have recently started working on adding control over the model itself. A VR proposal described in (Castelo-Branco et al., 2020) relies on desktop mirroring i.e., the VR user has access to a "window" that mirrors the view of the computer running Grasshopper. In order to interact with that window, one of the VR controllers "simulates" a standard desktop mouse, with a "point and click" approach. Figure 2.36 shows that mirror view integrated into the VR environment. Mindesk, a commercial software that we mentioned previously, was updated to provide the same functionality following the same approach of mirroring the computer screen.

Providing a mirror view of the desktop interface coupled with simulated mouse and keyboard input gives the user access to the same feature set as within the desktop tool itself. Nevertheless, interacting that way does not take full advantage of the 3D-tracked controllers and leads to difficulties when small items have to be selected or more generally when precision is required. Despite predating the aforementioned work, the prototype we describe in Chapter 4 is an attempt at tackling that interaction problem.

Figure 2.36: A mirror view of the desktop's display, incorporated into a VR-based environment. Reproduced from (Castelo-Branco et al., 2020).

2.5.3 Survey on the potential of Virtual Reality for architectural design

Based on the state of the art we discussed in this chapter, we posit that immersive technologies should be integrated as part of the CAAD process, and for AD in particular. To verify these claims, we conducted an online survey on the potential of using VR for architectural design.

The survey was shared in January 2020 to practitioners on the Rhinoceros fora[34], researchers on the official eCAADe LinkedIn and Facebook pages, as well as architecture students from three French-speaking universities, and was made available in two languages (French and English) in order to target and solicit a broader audience.

This section discusses the results we got out of that survey, even though one should keep in mind that only 36 complete responses were received. This is probably good enough to provide anecdotal evidence but is not sufficient to capture statistically significant evidence. A copy of the complete questionnaire and the results we gathered can be found online[35]. The code we used to analyse that data and produce figures is also available[36]. We also published a paper that expands on the results we obtained (Coppens et al., 2021).

Population

By targeting diverse venues, we managed to obtain a population of respondents with different profiles. Figure 2.37a shows the age distribution amongst the respondents, highlighting that 25 out of 36 participants (69%) were between 20 and 29 years inclusive (median: 25, interquartile range: 9.25). Figure 2.37b presents the distribution of experience amongst respondents through a population pyramid reporting the number of years of experience in architecture or related fields (median: 3, interquartile range: 8.5). These two figures seem to suggest that a significant part of the surveyed population is comprised of students and recently graduated practitioners.

Based on their reported architectural experience and diplomas, we classified respondents into four distinct architectural profiles: 2 *uninitiated* respondents, 11 *novices*, 11 *competent* participants, and 12 *experts*. We observed a balanced grouping of respondents in the three categories with the most qualified profiles, as well as a good gender balance across these categories.

---

[34]https://discourse.mcneel.com/t/your-opinion-on-vr-for-architectural-design/95111

[35]zenodo.org/record/4696074)

[36]zenodo.org/record/4696071)

(a) Respondents' age pyramid.

(b) Distribution of experience amongst respondents.

Figure 2.37: Profile of survey respondents.

We also wanted to assess the respondents' familiarity with VR as well as their experience with the technology, by asking what devices they had used. This allowed us to separate between three types of respondents: (1) those without VR experience; (2) those with prior exposure to "low-quality" VR devices (rotational tracking only); and (3) those that had been able to try fully tracked (6-DoF) VR experiences. Figure 2.38 depicts the corresponding distribution, with 11 respondents stating they were not familiar with VR at all (1), 9 belonging to type (2) since they were familiar with VR but had not tried a 6-DoF experience, and 16 with 6-DoF VR experience (3).

Figure 2.38: Familiarity of respondents with different quality levels of VR.

Potential of Virtual Reality for architecture

The core focus of the questionnaire was to assess the current use and the potential of using VR for architectural design practices. Respondents that had indicated prior exposure to VR were asked whether they had tried to use such devices for architecture-related activities. Among respondents that indicated having tried VR, more than half (11 out of 21) had also done so for architecture-related activities. For them, we queried which tools they had used and what limitations they encountered when using them.

Only 6 participants answered the question about the perceived limitations of architecture-specific VR tools, but they mentioned tooling complexity (R18: "work-intensive transition from regular CAD model to VR"; R17: "difficult to set a proper scale for the imagery"), issues with user interfaces or interactions (R27: "lack of easy-to-use interface"; R56: "limited interactions"), as well as collaboration needs (R27: "it gets kind of lonely in VR [...] on projects with multiple stakeholders, it takes a long time to present, because everybody wants to go in") and hardware costs. That being said, more than half (6 out of 11) of these respondents mentioned game engines in the list of tools they used for architectural activities in VR, which leads us to think they may not reasonably be aware of more ad-hoc and easy-to-use software solutions.

We also asked all respondents about the potential of VR for architecture-related activities, regardless of their previous experience with such tools and devices. Figure 2.39 shows the results we obtained for that question, contrasted with the respondents' prior experience with VR. One can observe that as much as 75% (25 out of 36) respondents indicated that they considered the technology could be at least very useful to the field, while the remaining 25% still consider the technology to be moderately useful.

In addition to these general results, we drilled down into specific stages and actors of the architectural design process that are perceived as the most suitable targets for embracing VR. We proposed four uses of the technology as possible answers for a multiple-choice question: (1) after the design process; (2) during the design process, for informing stakeholders other than the designers; (3) during the design process, to be used by the designer himself; and (4) right from the start and all along the design process.

Stage 1 covers what we identified as the most common contemporary use of VR technology in the architectural context, i.e., to show a finished design to a client. Stage 2 suggests the potential to show stakeholders a work-in-progress in order to gather early feedback that can be taken into account for subsequent iterations. Stage 3 encompasses a workflow where the designer sporadically checks on a design in VR, e.g., in order to evaluate the design

Figure 2.39: Perceived potential of VR for architecture, depending on whether respondents had prior exposure to 6-DoF VR.

at full scale and from a human point-of-view. Finally, Stage 4 represents the extreme case where VR is fully integrated into tools that support all steps of the architectural design process, potentially replacing non-VR solutions.

Figure 2.40 presents the answers received for that question, compared against prior VR exposure. A large proportion of respondents (29 out of 36) consider VR technology suitable for presenting a finished project, which supports the assumptions about the current use of the technology we derived from our literature review. More surprisingly, there are equally many respondents that indicate it could be used to involve stakeholders during the design process. Slightly more than half of the respondents (19 out of 36) believe designers themselves could use the technology during their architecture-related activities, while 9 respondents indicate they believe VR tools could be used right from the start and all along the design process.

Potential of Virtual Reality for Algorithmic Design

The questionnaire then focused on how VR could be used to support AD. For this part of the survey, we only considered respondents that were aware of algorithmic design or parametric modelling tools. The vast majority of them (34 out of 36) signalled an awareness of such tools. Given the vague and confusing term "parametric", and in order to better appreciate the level

Figure 2.40: Stages of the Architectural Design process suitable for VR integration.

of understanding of what the paradigm entails, we asked respondents "How would you define parametric modelling/design?".

In a similar way to (Stals et al., 2018a), we classified respondents based on their answer into three distinct categories: "wrong definition", "correct definition", and "unclear definition". While the first two categories are rather straightforward, the latter contains respondents that did mention the parametric aspect but whose proposed definition did not contain any reference to algorithms, programming or interlinked components, and did not mention that variations of parameter values produce chain reactions. Since such an incomplete definition means the presence of parametric objects (e.g., parametrised primitive shapes in traditional CAD) is enough to fill the bill, it would also apply to non-algorithmic software.

Out of 34 respondents that answered this question, only 12 gave a definition that we considered to be correct, while 18 persons provided a definition that we classified as being unclear.

By combining the respondents' answer with their architectural profile, we classified them into four categories based on their AD profile: *uninitiated*, *novice*, *competent* and *expert*. Figure 2.41 shows the distribution of these categories across respondents, in relation to their age.

Next, we queried respondents on the usefulness and desirability of integrating AD software and VR hardware. We first showed them an online video of the prototype we will describe in Chapter 4. This prototype demonstrates the feasibility of editing a Grasshopper algorithm in VR with an interface adapted

Figure 2.41: Algorithmic Design profile of the respondents.

to the three-dimensional immersive context. It should however be noted the responses are therefore based on the respondents' subjective perception derived from a video that demonstrates a prototype that does not intend to provide a consumer-grade experience. Exacerbated by the fact that the video has to be watched on a 2D screen that does not generate the same immersion as an actual VR-based test, some of the responses may have been negatively impacted.

Nevertheless, after viewing the video, each respondent was presented with a question on the usefulness of similar VR-enabled functionality if it were combined with a 3D visualisation of the geometry being worked on. We identified an interest in having VR tooling adapted to the AD paradigm, but not nearly as pronounced as the one for CAAD in general. In fact, as illustrated on Figure 2.42, almost half of the respondents (16 out of 34) consider VR functionality as moderately or very useful for AD. The two sub-figures compare these answers with prior exposure of respondents to 6-DoF VR on the one hand (2.42a) and to VR tools for architecture on the other hand (2.42b). Note that the latter only takes 20 answers into account, since we did not include respondents that indicated they were not familiar with VR or had never experienced the technology. Figure 14 does take those respondents into account.

The proposed prototype was slightly better received amongst respondents with prior exposure to 6-DoF VR and VR tools for architecture, respectively with 54.5% (6 out of 11) and 53.3% (8 out of 15) of the answers that consider it to be at least "moderately useful" within these subgroups of the population. Although we notice that slight difference, the number of respondents is too low

(a) Usefulness of VR for AD, broken down by respondents' prior exposure to 6-DoF VR devices and experiences.

(b) Usefulness of VR for AD, broken down by respondents' prior exposure to VR tools for architecture (if they were familiar with VR).

Figure 2.42: Usefulness of VR for Architectural Design.

to draw any statistically significant conclusions.

General outcome

As a whole, the survey results confirm our assumption that the VR technology should be used earlier in the design process. A very large number of respondents see potential for a more user-centric approach that would involve stakeholders during the design process. Although the demand for VR editing tools is not as high (for architecture in general and for AD more specifically), it still is there.

Despite the threats to validity we have to take into account (especially the population size that prevents us from drawing statistically significant conclusions), we consider the findings we derived from this survey to provide sufficient anecdotal evidence of the need for better immersive tooling in architectural practice that goes beyond the current usage. Architects need solutions to be able to edit and evaluate designs from an immersive context, with adapted user interfaces and interactions.

Immersive parameter adjustment for algorithmic co-design

> "A display connected to a digital computer gives us a chance to gain familiarity with concepts not realizable in the physical world. It is a looking glass into a mathematical wonderland."
>
> Ivan Sutherland

As stated in Chapter 1 and as supported by the survey discussed in Section 2.5.3, we posit that immersive technologies should be integrated into the architectural design process and to AD in particular. To address this need, we follow a prototype-based process described in Section 1.4 and therefore develop research prototype tools that show the potential of using immersive technologies for AD activities.

Based on our literature review and the state of the art of the various domains our research covers, we start working on how to bring immersive technologies to AD in architecture. In this chapter, we explore different display options (VR and AR) and interaction modalities (based on desktop-like interfaces adapted for Six Degrees of Freedom (6-DoF), but also using **Tangible User Interfaces (TUIs)**) and focus on immersive geometry visualisation and immersive value adjustment of parameters shared with an AD definition.

3.1. Requirements for creating immersive Algorithmic Design tooling

The most essential aspect to take advantage of immersive technologies is to provide a way for an AD project to be visualised in an immersive manner, preferably with a navigation system so as to be able to examine the rendered architectural geometry from different angles.

Since we do not aim to replace desktop-based tools such as Grasshopper entirely, our goal is to interact with existing AD definitions created with such tools. Two options are available to do so.

The first one is implementing a way to read from and write to a specific AD format (e.g., Grasshopper's file format) from an immersive application, to process a given definition. We can then develop a module that generates the geometrical representation based on that definition.

Another option is to create a "bridge", between an immersive application and the desktop-based tool, through a custom component for that desktop tool. The component would simply share geometry and parameter data to a companion immersive application. Generating the geometrical representation based on the definition remains the desktop tool's responsibility in that case; the custom component simply sends the result to the immersive application.

We opted for this latter option, since it does not force us to dive into the logic of how to generate geometries based on an algorithm. Moreover, it allows the resulting immersive application to be tool-agnostic, in the sense that any AD software would be able to work with the immersive application, as long as a bridge plug-in is created for that specific software.

Ideally, the process of transferring the geometry from a CAAD desktop-tool to the immersive application should satisfy two main properties: it should be fast so that the user benefits from near-instant updates on a modified geometrical representation, but also automatic, so that the designer does not have to go through a cumbersome process of exporting a geometry each time a new version is to be evaluated. One might also be tempted to aim for the update process to be incremental (working with modifications to the current state of the geometry instead of resending the whole geometry). Unfortunately, doing so would require a geometry-adapted diff tool (Doboš & Steed, 2012) that would inherently be imprecise (because the rendering-adapted representation of the geometry is itself imprecise) and would likely be computationally expensive. In the specific context of AD, a particular change (e.g., a value change on a parameter) may impact the whole geometry, rendering such diff tool even less appropriate.

To go further than visualisation only, it is needed to add different kinds of interactions with the design, that we classify in 4 groups corresponding to different levels of control, as shown in Table 3.1.

We first differentiate between **extrinsic** and **intrinsic** control. *Extrinsic* modifications to a project only impact the rendered geometry or its surrounding environment (e.g., lighting or weather). This correspond to a first level of control (*Level 1*) and includes modifications to textures, or changes in zoom level.

On the other hand, *intrinsic* control actually allows for modifications to the AD model itself. We further split intrinsic control into three more levels of control: input control (*Level 2*), high-level algorithm (*Level 3*) and low-level algorithm (*Level 4*). These three levels allow designers to have control over the actual AD definition, impacting the way the geometrical representation is generated.

Input control is limited to the data used by the algorithm (e.g., parameter values and key or anchor points for certain components), while both algorithm control levels allow for modifications to the algorithm itself.

High-level algorithm control provides the designer with the ability to manipulate the (often visual) algorithm that drives the geometry generation. This includes adding or removing components in a visual algorithm (as in Grasshopper) and making changes to the links between them.

Low-level algorithm control allows designers and potentially software developers to modify the (often textual) code inside the components used by the high-level algorithm. This includes implementing custom components to perform project or company-specific geometric operations, or connecting to an external service to retrieve or share data.

A designer clearly needs at least *Level 3* control to create an architectural project, but can be confined with *Level 2* control when tweaking an existing project (therefore on a temporary basis). *Level 4* is only really needed when the built-in components in the AD tool do not suffice for specific needs, and when no custom component was created or shared by a third-party to cover these needs.

We already discussed in Section 2.5.2 that *Level 1* can be achieved with many existing tools. The current chapter presents our work towards enabling *Level 2* control over an AD project from within an immersive experience. Chapter 4 will present how we manage *Level 3* control.

Level 4 is however outside of the scope of this dissertation. This is due to two reasons: (1) it typically involves textual code and typing and immersion do not currently mix very well (Grubert et al., 2018), and (2) custom compo-

	Type of control	Target users	Description	Examples
Level 1	extrinsic	designer and others	Making modifications to the rendered geometry only	textures, zooming, geometry positioning, lighting
Level 2	input	designer only	Making changes to data used by the algorithm generating the geometry	adjustment of parameter values, key point positioning
Level 3	high-level algorithm	designer only	Editing the (generally visual) algorithm used to generate the geometry	adding links (edges), components (nodes), removing or grouping them
Level 4	low-level algorithm	designer and developer	Developing (generally textual) code for custom components to be used for Level 3 control	performing geometric operations or connecting to external services

Table 3.1: Types of control over Algorithmic Designs.

nent development is usually done through specialised development software, outside of Grasshopper. (2) means that integrating immersive technologies for *Level 4* control would heavily complexify the development of a solution, since it would have to communicate with these specialised development tools and have them process changes to produce new versions of the custom component being modified. Even then, Grasshopper would need to be restarted to reload all components and thereby take into account changes made to the specific component that was just modified. (1) further implies that overcoming these obstacles are not even guaranteed to provide a truly useful interface.

To clarify some usage scenarios where *Level 2* control is likely to be beneficial to the design process, we provide the following two hypothetical use cases, that we will use as an inspiration to develop our research prototypes.

Use Case 1. An architectural firm is in charge of the construction of a modern-looking clubhouse for a golf club. Based on the club's demands, the architects have designed a first version of the building that will welcome the 300 registered members. The design process in quite advanced and the building is expected to be close to its final form. The architects would like to get input from the clients (the club's board members), so they invite them to a VR session where both the lead architect and board members visualise the virtual building in its current state. The VR visualisation helps the clients understand what the actual clubhouse will look like. The architect guides the clients around the virtual environment and adjusts parameter values along the way to accommodate the remarks from the clients.

Use Case 2. The municipal council of a city wants to create a public space in a disused area of 1,000 square meters, nestled amid a few apartments and shopping buildings. They appointed an architectural firm that works with Algorithmic Design to do so and asked them to include a big sculpture to be placed somewhere in the centre of that space. The project is already well advanced in Grasshopper and the architects now want the opinion of the council to adjust parameter values. They therefore invite council members for a meeting at their office, that themselves invite representatives from commercial and residential buildings in the project's vicinity with them. Using the immersive and interactive system at the architects' office, all these stakeholders are able to visualise the project from different angles and can take an active part in the remaining (parameter adjustment) design decisions. The AR and VR visualisations help non-architects to better appreciate the dimensions of the sculpture and how it integrates with the surroundings.

3.2. Prototype for adjusting models in Virtual Reality

In order to enable *Level 2* control over an AD project, we developed a proof-of-concept research prototype, that serves as a bridge between a Grasshopper definition and a VR headset. We therefore decided to call it **GHVRBridge**. We chose Grasshopper because it is very popular in both industry and research (Cichocka et al., 2017), and supports the creation of custom components through a C# Software Development Kit (SDK). As for the VR headset, we chose the HTC Vive HMD since it was, at the time, one of the very few options to offer good quality (including 6-DoF tracking) at an affordable price. To ensure generalisability of the approach, we rely on a cross-headset toolkit that should work with other devices as well, so as to handle different settings should the need arise.

A video demonstrating GHVRBridge is available online[1] and our codebase is hosted as open-source software on a GitHub repository[2], available under the MIT[3] license. The prototype was also described in a paper we published (Coppens et al., 2018).

Figure 3.1 presents the overall concept and structure of GHVRBridge, involving three separate software components in addition to Grasshopper itself: a Grasshopper component that shares geometries and parameters, a VR application for the HTC Vive, and a relay server to forward the information that

---

[1]`http://informatique.umons.ac.be/staff/Coppens.Adrien/?video=eCAADe2018`
[2]`https://github.com/qdrien/GHVRBridge`
[3]`https://opensource.org/licenses/MIT`

Figure 3.1: GHVRBridge: a proof-of-concept research prototype that enables VR-based visualisation and adjustments of Grasshopper definitions.

needs to be exchanged. The next two sections focus on the first two components only, since the relay server's functionality is rather self-explanatory. It allows for VR users to be in a different location than the computer running Grasshopper, and will facilitate the later extension to a collaborative setting.

3.2.1 Grasshopper custom component for external parameter value adjustment

We developed a Grasshopper sharing component, written in C#. It should be placed as any other Grasshopper component inside the AD definition in order to interact with other components. The component contains 4 main input ports: *Address*, *Connect*, *Meshes* and *Shared parameters*. The corresponding representation in Grasshopper is outlined in Figure 3.2. An actual concrete example is shown in the green component in the bottom-left picture of Figure 3.1. When activated via a boolean switch linked to the *Connect* input port, the component connects to the relay server indicated by the *Address* port, via the WebSocket protocol (Fette & Melnikov, 2011). It then iterates over the input parameters that the designer has linked to the *Shared parameters* port, stores their properties (name, type, range, values and a unique identifier) and sends this information to the relay server that forwards it to the VR applications. The geometries linked to the *Meshes* port are then shared as well.

The component listens to incoming parameter adjustment messages, coming from a VR application and relayed by the WebSocket server. When one such message is received, the component iterates over the list of value changes it contains and replicates them on the corresponding input parameters in Grasshopper (identifying them thanks to the previously mentioned unique identifiers).

To ensure that messages are properly sent and received, we require a transmission protocol that provides reliable transfers; otherwise, geometries or parameter updates may never reach their destination. WebSocket is a well-established protocol that enables bidirectional communication over a Transmission Control Protocol (TCP) (Postel, 2011) connection, providing such reliable transfers. While removing the complexity of handling plain TCP connections manually thanks to well-supported libraries, WebSocket does not consume more network traffic than plain TCP, except for the initial handshake (at the start of the connection) (Skvorc et al., 2014). We therefore chose to rely on that protocol for implementing GHVRBridge.

In order to optimise the content (payload) of the messages exchanged by

---

[4]`https://google.github.io/flatbuffers/`

Figure 3.2: The GHVRBridge Grasshopper custom component for external geometry sharing and parameter adjustment.

GHVRBridge, the data is structured via Flatbuffers[4], a serialisation library that can easily convert mesh and parameter data to a binary format that minimises the size of the (binary) payload. Compared to other very popular exchange formats such as JSON (ISO/IEC-21778:2017, 2017), XML (Yergeau et al., 2008) and YAML (Ben-Kiki et al., 2009), Flatbuffers (de)serialises faster and produces smaller transmittable messages[5], reducing the traffic to be sent over the internet in the case of a remote communication.

The Flatbuffers library requires that we define the schema (format) of the data in a `.fbs` file, before we can build converters from and to that format. This schema is presented in Listing 3.1. In it, we define a `Components` message that can contain multiple instances of a `Component`, which can either be a `BooleanToggle` or a `NumberSlider`. As required by the syntax, both are encapsulated into a `GenericComponent`. Depending on the type of component, different fields are listed. The `value` field has a different type depending on the component's type, but a `NumberSlider` also requires information about the range of possible values and it therefore contains additional fields to store that information.

Based on a given schema, Flatbuffers creates a set of classes to be used in the project's codebase. The behaviour of the component we use to transfer geometries and parameters is defined in a class called `GHSharingComponent`, that instantiates `GH_Component`, so that it can be integrated into a Grasshopper (visual algorithm) definition.

---

[5] `https://google.github.io/flatbuffers/flatbuffers_benchmarks.html`

Figure 3.3 displays the class diagram of the C# project for the Grasshopper custom component we developed, including part of the `GrasshopperVRBridge.IO` package that was automatically generated using Flatbuffers (the dots on the Figure accounts for the other classes that were automatically generated based on the schema).

3.2.2 Virtual Reality application

In order to create the VR part of GHVRBridge that will consume the data provided by the custom Grasshopper component described in the previous section, we rely on Unity, a game engine that facilitates the creation of interactive three-dimensional applications. Additionally, Unity also helps us to create VR experiences in particular, through an easy access to SteamVR, the platform that handles the communication with the HTC Vive headset. Since we develop Grasshopper custom components using C#, we define the behaviour of the VR application with C# scripts. This facilitates code reuse between different components of the GHVRBridge system and lowers the risk of incompatibility issues.

The VR application supports two types of users. Simple clients can visualise the rendered geometry in VR and receive updates whenever a new one is generated in Grasshopper. On the other hand, designers additionally see parameters and can modify their values so as to edit the Grasshopper definition. Depending on how the application is started (configured for a certain type of user), it listens to a WebSocket service that only shares the necessary information for that type of user.

When receiving a geometry update, the VR application reads the Flatbuffers payload that contains what is called a mesh. This is a simplified representation of the surface of an object using polygons (typically triangles), defined by vertices, edges, and faces. From the mesh data, the VR application creates a three-dimensional object (such as the egg-shaped tower in the bottom right picture of Figure 3.1) that uses the corresponding polygons, with two peculiarities to take into account.

The first one is that Grasshopper uses a right-handed z-up coordinate system, while Unity is based on a left-handed y-up system. To clarify, Grasshopper uses a system such as the one depicted on the right (B) side of Figure 3.4, with the z axis pointing upwards, the y axis pointing forward, and the x axis pointing to the left. On the other hand, Unity's coordinate system is based on the left side (A) of Figure 3.4, with y pointing up, z pointing forward, and x pointing right. This implies that y and z values need to be switched, and x

Figure 3.3: The class diagram for GHVRBridge's custom Grasshopper component. Only the "entry classes" (`Components` and `MeshData`) of the `GrasshopperVRBridge.IO` package are shown so as to keep the diagram readable on a single page.

Listing 3.1: The Flatbuffers Schema definition for exchanging parameter data.

```
namespace GrasshopperVRBridge.IO;

enum Accuracy:byte {
    Float = 0,
    Integer = 1,
    Even = 2,
    Odd = 3
}

table BooleanToggle {
    name:string;
    guid:string;
    value:bool;
}

table NumberSlider {
    name:string;
    guid:string;
    value:float;
    accuracy:Accuracy;
    min:float;
    max:float;
    epsilon:float;
    decimal_places:short;
}

union GenericComponent {
    BooleanToggle,
    NumberSlider
}

table Component {
    abstractComponent:GenericComponent;
}

table Components {
    componentsVector:[Component];
}

root_type Components;

file_identifier "PARA";
```

values must be modified to -x.

The second need is the computation of normals, i.e., vectors perpendicular to mesh surfaces and pointing outwards. Normals allow Unity's lighting system to properly project light on the generated objects and compute shadows; there is one normal to be computed for each vertex in the mesh.

Figure 3.4:   Left-handed and right-handed coordinate systems.

As for parameter updates, the VR application reads the information according to the format presented in Listing 3.1, and generates the appropriate widgets. Boolean parameters are mapped to switch toggles, while numbers lead to sliders whose possible values depend on the received settings data. These widgets are placed on a panel attached to the user's non-dominant hand (left hand by default but the user can switch hands by interacting with a button), that is shown on the bottom-right picture of Figure 3.1.

When the user makes changes to a parameter value, the application sends a modification update to the relay server that then forwards it to the Grasshopper component. That update contains the unique identifier of the parameter that was modified, as well as the new value, which is then reflected onto the actual Grasshopper parameter.

To clarify some of the networking and input handling aspects, Figure 3.5 shows a portion of the class diagram for GHVRBridge's VR application. On the Figure, the same automatically-generated `GrasshopperVRBridge.IO` package can be found, since the data receiving classes of the VR application rely on them to read from (and create) update messages. The initiated reader will recognise the Observer and the Singleton design patterns (Gamma et al., 1995). The former is used to allow subscribing objects to listen to updates efficiently, while the latter ensures only one instance of a given class exists. Due to Unity's usage of "game object" entities, on which "scripts" that communicate through messages are attached, the Singleton design pattern is also a more efficient way

Figure 3.5: Part of the class diagram for GHVRBridge's VR application, clarifying some of the networking and input handling aspects.

to enable communication between scripts (e.g., to notify the `SocketManager` that a parameter update has to be sent) than the standard messaging system.

3.3. Enabling immersive collaboration

The GHVRBridge proof-of-concept prototype we described thus far allows for two different modes that can accommodate designers that want to tweak parameters, and simple clients that only want to visualise the geometrical representation inside a VR environment.

But the architectural design process is a collaborative endeavour; it is there-

fore desirable to allow that collaboration to continue inside the VR application. We want to allow multiple designers to work on the same model at the same time (Dagit, 1993), from the same shared VR experience. Furthermore, in order to involve clients during the design activities and thereby move to a VR-based user-centred process that increases quality and performance in architectural projects (Bullinger et al., 2010), we need to allow both types of users (designers and simple clients) to participate in the same (shared) virtual experience.

The way we designed the GHVRBridge system makes it possible to connect multiple instance of the VR application to the relay server, to facilitate the sharing of the experience by several VR users that potentially have different roles (viewer only or designer with control over the parameter values), as illustrated in Figure 3.6. We note that, as seen on the Figure, only one Grasshopper instance is running; the collaboration we enable is focused at the VR level. Desktop-level collaboration is indeed out of scope of the present research since we work with immersive technologies, and would likely need to be implemented as part of the core code of Grasshopper by the vendor itself (and not as a custom third-party component).

A video demonstrating the collaborative extension of GHVRBridge is available online[6] and our codebase is hosted as open-source software on a GitHub repository[7], available under the MIT[8] license. We also published a paper describing this collaborative extension and the challenges we encountered (Coppens & Mens, 2018).

3.3.1 Virtual co-presence

Regardless of the exact collaboration scenario (e.g., a designer showing adjusting parameter values with a client), VR users need to see one another and be able to observe the actions of other users to collaborate efficiently (Nguyen & Duval, 2014). While VR headsets typically occlude the wearer's view of the real world, it is possible to replace a physical co-presence by a virtual one, using avatars that mimic what the real person is doing.

Depending on the exact immersive devices being used, different tracking options could be available. With the HTC Vive, it is only possible to know the position and rotation of the headset and the controllers. This means we can only portray these items' motions accurately. Under the reasonable hypothesis that the user is wearing the headset and holding the controllers in his hands,

---

[6] `http://informatique.umons.ac.be/staff/Coppens.Adrien/?video=CDVE2018`
[7] `https://github.com/qdrien/GHVRBridge`
[8] `https://opensource.org/licenses/MIT`

Figure 3.6: The GHVRBridge prototype adapted for multi-user immersive adjustment of Grasshopper parameter values, with two possible modes: mesh streaming (i.e., geometry visualisation only) or parameter sharing (i.e., control over shared parameters).

we can even depict the user's head and may approximately picture his hands (although he may hold the controllers or press buttons in a different way than the one we expect). We chose to limit ourselves to the simple factual depiction of the HMD and the controllers, as shown in Figure 3.7. We therefore only depict objects that are actually tracked, without trying to infer elements (gaze, fingers, etc) that may then be incorrectly represented (e.g., if the user does not grasp the controllers the way we expect him to) even if that would have lead to a stronger level of immersion (Lin et al., 2019).

In order to enable a shared experience, head and controller positions and orientations have to be sent to all VR application instances sharing the collaborative experience. This information does not have to transit through the relay server nor reach the Grasshopper component, as it is only valuable to VR instances. Our approach therefore is to let VR instances communicate "co-presence information" only between them. When initially connecting to the relay server, a VR instance indicates that it wishes to join the shared experience, to which the server either replies with the current host's IP address, or simply by "you", thereby informing that particular instance that it will be the session's host. In the latter case, the IP address of the machine on which the VR application is running is stored by the server to be provided to other VR instances that may join the session later. In case the host is disconnected, a new one is elected by the server and broadcasted to all VR instances.

The system to decide and advertise the host allows VR instances to com-

Figure 3.7:   A collaborator waving towards another VR user, as seen from that user's point-of-view. Both the collaborator and the user are co-located in the same virtual environment, with the motions of each individual's head and controllers being tracked and shown to others.

municate directly with that host, that will forward (position and orientation) updates to other instances. Instances will have to exchange messages at a high rate to provide a somewhat smooth experience. Because of that and since losing a packet (i.e., missing a position update) is not really a problem as newer information will quickly arrive, we should this time rely on the User Datagram Protocol (UDP) (Postel, 1980) to transfer the data. In fact, that protocol does not provide mechanisms similar to TCP to ensure reliable transfer of data, but offers better speed. To implement the co-presence feature that was just described, we relied on Unity's built-in multiplayer High Level Application Programming Interface (API) (HLAPI[9]), that is implemented on top of UDP.

3.3.2   Towards collaborative parameter value adjustment

While the previous section explains how we minimised message size and latency for remote collaboration, another challenge not directly related to network aspects arises when several users are allowed to modify parameter values: handling concurrent and potentially conflicting modifications. Safeguards may be put in place to avoid a frustrating user experience.

Many researchers have worked on issues related to concurrent editing and collaboration conflicts. There are in fact even venues (both conferences and

---

[9]https://docs.unity3d.com/Manual/UNetUsingHLAPI.html

journals) specifically focused on such topics, such as the Computer-Supported Collaborative Work (CSCW) conference[10]. An extensive literature can be found on methods that handle concurrency for database manipulations in particular (Munson & Dewan, 1996).

Concurrent modification of parameters is not exactly similar to reading and writing database entries since methods used in that context need to handle transaction dependencies (modifications of elements that depend on each other are not allowed). The programming context of AD means it is tempting to look at solutions used for collaborative software development. But, in that case, semantic understanding is key and software merging techniques (Mens, 2002) are necessary since changes to the code are typically done without conflict checking mechanisms, and are only merged together later (with modern version control systems).

Since AD is a particular paradigm for CAAD, it would be equally tempting to develop upon general collaborative design tools. But these tools typically do not rely on algorithms and therefore need to deal with different issues than the ones we are being faced with. For instance, collaborative direct modelling tools must make sure that instructions sent by different collaborators do not create an invalid result (incompatible changes, potentially related to the order in which instructions are processed) (Hepworth et al., 2014).

It therefore is more appropriate to assess the potential of techniques used in other contexts, including collaborative database management. Another field that we can draw inspiration from is multi-processors systems with shared resources. The problem of avoiding conflicts between tasks that must run in disjoint time intervals is called mutual exclusion (Baker & Coffman Jr, 1996).

By reading papers from that literature and reasoning about the way to apply them in the concurrent parameter adjustment context, we ended up with a few options, listed hereafter:

- *Overwriting updates*:

  Whenever the system receives an update of a parameter value, the previous one is overwritten. This default behaviour is the most tolerant, but does not provide any safeguard to concurrent modifications. In addition, latency has to be minimal for this approach to work, otherwise the system will appear to be ignoring certain user changes. This solution can be acceptable if designers rarely modify parameters simultaneously, and if the rate of change is not too high.

---

[10]`https://cscw.acm.org/`

- *Reactive locking:*

  Whenever a designer starts modifying a parameter value, his collaborators are notified (e.g., through some visual clue) and they can no longer modify that parameter until the first designer releases it. Conflicts could still happen due to latency, if another designer tries to modify the same parameter before receiving the "lock notification". In that case, the system should notify the user that his modification request was rejected.

- *Preemptive locking:*

  Preemptive locking is an even more conservative approach that prevents the previous problem from happening. By default, parameters cannot be edited. If a designer wants to modify a parameter value, he first needs to request access to that parameter. This is similar to the voting mechanism in place in many systems tackling the mutual exclusion problem. An example that we could draw inspiration from would be an algorithm proposed in (Maekawa, 1985), that only requires $\sqrt{n}$ messages to coordinate n nodes in a decentralised system.

- *Privilege strategy:*

  In addition to the locking strategy, a mechanism based on user privileges could be put into place. Users with higher privileges might be granted the ability to take control of a parameter that is being modified concurrently by someone with lower privileges. In that case, the less-privileged user should be notified that he can no longer control the said parameter.

- *Parameter layers:*

  We could consider grouping parameters so that users could get access to different groups of parameters. This could reduce the concurrent modification problem for instances where users mostly get access to different groups of parameters, or even remove that problem entirely when all users are assigned to different groups. We should however note that grouping parameters should in general induce more conflicts, since two users willing to interact with two different parameters could be stopped if these two parameters are part of the same group. That solution, if implemented, implies that grouping elements should be made with caution.

Collaborative testing and evaluation sessions using the GHVRBridge prototype will typically involve a limited number of designers with concurrent access to the same parameters, because of hardware and space constraints, but

also because studies suggest that working groups should be rather small to be effective (e.g., at most four as per (Steiner, 1972)). The limited number of active participants should result in very few conflicts. Furthermore, immersive sessions involving multiple designers only really make sense if these designers are co-designing a project and discussing changes to be made to the geometry, based on the immersive visualisation they are experiencing. In that case, they will probably naturally avoid concurrent modifications since they will be focusing on one change at a time (the one they are discussing).

For these reasons, we chose to stick with the simple *overwriting* approach and simply added an "update rate limiter" to make sure one application cannot send too many messages to the system. This helps in reducing the risk of conflict, but also the burden on Grasshopper, that needs to recompute, regenerate and resend the geometry after each update.

3.4. Evolving the concept for Augmented Reality and Tangible User Interfaces

While the GHVRBridge prototype (presented in Section 3.2) and its adaptation to a collaborative setting (presented in Section 3.3) enable parameter sharing and geometry visualisation in VR, their simple user interface still resembles the classical WIMP approach, with a VR controller acting as a three-dimensional version of a mouse.

To provide a more natural collaboration environment within the same physical room, we worked with the Luxembourg Institute of Science and Technology to develop another prototype that combines different kinds of interaction devices. It in fact relies on a VR headset but additionally utilises AR glasses (Microsoft Hololens) and Tangible User Interfaces (a display table with tangible items on it) so as to create a multi-modal system. We call that multi-modal prototype system **GHXR**, for Grasshopper for XR.

3.4.1 Overview of the system

The general idea, depicted in Figure 3.8, is that a table display is placed in the middle of a room in which collaborators can work on a joint project. These collaborators are surrounded by a circular screen setup (covering about 300° around them), that displays three different views of the geometrical representation being worked on and its surroundings: a first-person point-of-view, a top-down (plan) view that is also reproduced on the table display, and a 3D perspective view. The designer is therefore free to choose the appropriate rep-

resentation depending on the specific aspect or problem being discussed, at any point in time. The constituents of the system and the data transfers between such constituents are depicted in Figure 3.10.

A tangible item (that we may simply refer to as "a tangible" hereafter) that roughly looks like a hockey puck is placed on the table, as shown on Figure 3.11, for each Grasshopper parameter that is shared. Such a tangible is tracked via a paper marker (see Section 2.1.2 and Figure 2.8 in particular) placed on its bottom and a camera watching through the display to recognise this marker, so that its position and orientation on the surface of the table are known. When a designer rotates a tangible mapped to a parameter, the value of that parameter is adjusted and sent to Grasshopper to update the generated geometry.

In addition to the tangibles representing Grasshopper parameters, two more tangible items are placed on the table: one to control the top-down view, allowing to zoom in or out by rotating the item as well as to reposition the view by sliding the item, and one to move the human (first-person) point-of-view (to teleport the view to the target location by moving the item and to change the view angle by rotating the item).

The human point-of-view is also visible through the VR headset, but in visualisation mode only. Write access to parameters is not allowed since they are being controlled by the tangibles on the table. In addition to the VR, table, and circular screen displays, a user wearing an AR headset can see a 3D hologram of the geometry being worked on, projected at the correct location on the table, as pictured in Figure 3.12.

Figure 3.9 depicts three users collaborating through GHXR. The system naturally enhances collaboration compared to GHVRBridge, since users can see and talk with each other, and interact with the same physical tangible items. A video demonstrating GHXR is available online[11] and our codebase is hosted as open-source software on a GitHub repository[12], available under the MIT[13] license.

The next 4 sections discuss the necessary constituents to produce GHXR: a Grasshopper custom component, a table display application, an application for a circular screen setup that surrounds the users collaborating through the table application, and two immersive applications (for the VR and the AR headsets, respectively).

---

[11]`https://youtu.be/L5dqMx7rnmM`
[12]`https://github.com/qdrien/GHXR`
[13]`https://opensource.org/licenses/MIT`

Figure 3.8: Mock-up of the GHXR system, with a table display on which users interact and collaborate. They are surrounded by about 300° circular screens, displaying different views. The example shows a building project in a city centre, with two Grasshopper parameters available to the users.

Figure 3.9: The GHXR system, with three users discussing over an AD model, using the different views offered by GHXR.

Figure 3.10:   Constituents of the GHXR prototype and data transfer between these constituents.

Figure 3.11: A Grasshopper list component and its assigned tangible item in GHXR's table application, with the corresponding software widget around it.

Figure 3.12: The additional 3D hologram of the building that AR users can see, placed at the correct location on the table's plan view.

3.4.2   Grasshopper custom component

Similar to the Grasshopper custom component for GHVRBridge we described in Section 3.2.1, we develop another Grasshopper component to exchange geometry and parameter data with a server, that itself relays that data to the applications that run on the table display, the computer controlling the circular screens, as well as on the VR and AR headsets.

The "Shareable" classes on the class diagram of Figure 3.13 are data classes used to easily convert from and to the transfer format (JSON in this case since the system is here supposed to exchange data only on a local network). On the same Figure, one can notice the `GHXRSimplifiedComponent`, handling most of the logic behind the Grasshopper custom component (its name contains "Simplified" because it requires less configuration on the user's end than a previous version).

A `DelayedMethodCaller` class can also be seen on the class diagram ; it serves to delay parameter and geometry updates, so as to limit the frequency of such updates that are sent to "consuming" applications. If another change generates a newer update within that `DelayedMethodCaller`'s timeframe, only the newest update will effectively be sent.

Instead of using WebSockets as for GHVRBridge, we here rely on Message Queuing Telemetry Transport (MQTT) (Light, 2017) as the communication protocol, since it is already supported by the framework we need to use for the table display.

Since the geometry shared from Grasshopper is localised for GHXR, we include the GPS position (manually indicated by the user) and heading information together with the mesh data (that itself has the same structure as described in Section 3.2.2). As for parameters, we support number and boolean values (as for GHVRBridge), but we also add support for lists of textual values, that are themselves mapped to any values within Grasshopper.

In order to develop the Grasshopper custom component that shares the geometries and parameters, we had to overcome a few obstacles related to concurrent access to software objects.

The first one is that modifications to Grasshopper objects (e.g., parameters) cannot be made from anywhere else than the main application thread and it is therefore necessary to rely on a message queuing system to save incoming updates for later processing.

Another difficulty we faced is that the messages queues, that are necessary to store the exchanged information in this queuing system, needs to be manipulated by both the main and background threads. We therefore have to make sure to avoid what is called race conditions, i.e., concurrent modification of an

Figure 3.13: The class diagram for GHXR's Grasshopper custom component.

object, leading to application bugs or crashes.

Figure 3.14 describes the logic we rely on to handle these issues.

When receiving incoming updates from an application, the GHXR Grasshopper component is notified through the `MessageReceived()` method that runs in a background thread. After enqueueing the message, the `MessageReceived()` method "expires" the Grasshopper solution, forcing it to be recomputed and leading to a call to `SolveInstance()`, that itself runs from the main thread.

After making sure that the custom component is properly setup and connected, the `SolveInstance()` method can dequeue messages since it is running on the main thread and the corresponding data can be processed on it. Once changes contained in these messages are reproduced in the Grasshopper definition, it still is necessary to expire the solution once again (or at least the corresponding components and the ones that depend on them), so that Grasshopper recomputes the (part of) the visual algorithm to produce the new geometrical representation.

In order to avoid race conditions, we use the locking mechanism available in C# to make sure the queues cannot be manipulated at the same time by different threads. Using that mechanism, when a thread A wants to get access to a variable that is currently locked by another thread B, A's execution is paused until B releases that variable.

Note that a similar logic was applied to solve the same sort of problems for the component described in Section 3.2.1.

3.4.3   Table application

In order to display a plan view and handle interaction with tangibles placed on the tangible table, we rely on an existing framework called TULIP (Tobias et al., 2015). This open-source framework, custom-built at the Luxembourg Institute of Science and Technology, is fully integrated with the tangible table. TULIP is a modular framework, in the sense that it allows for modules to be integrated to a project. For instance, we here use the GIS module to retrieve data from OpenStreetMap[14] to display the plan view, as well as the IOT module to handle the connection with the MQTT relay server (called a broker in MQTT terminology).

The TULIP framework works with scenarios described in XML files, that feed the core software, itself in charge of creating widgets. Widgets are interactable software entities placed on tangible items, as shown in Figure 3.11, where we see a widget for value control over a parameter simply named "List",

---

[14]`www.openstreetmap.org`

Figure 3.14: The underlying logic behind GHXR's custom component for Grasshopper, to send geometries and share parameters while handling concurrency issues. The black "End" circles indicate simple terminations of the `SolveInstance()` method, while their grey counterparts represent terminations that are preceded by the expiration of the Grasshopper solution, leading to a new call to `SolveInstance()`, hence the inclusion of dashed arrows.

that can take three values: small, medium, or big radius

The specific scenario we use for the table application tells TULIP to display a plan view of a geographical location that therefore appears on the table. The exact location is specified inside the scenario's XML file. The table application then connects to the MQTT broker and awaits a parameter sharing message. Once received, it uses that data to map each parameter to a tangible item with an appropriate widget (depending on its type and possible values) so that changes to the widget value through the tangible item are sent back to Grasshopper via the broker.

The scenario also creates additional widgets to control the top-down view and the first-person views, and shares the GPS coordinates of the corners (bounding box) of the table's plan view, so that other visualisation applications know what is being displayed on it.

3.4.4   Circular screen setup and Virtual Reality applications

Since a significant part of the code used for the VR experience is shared with the application that runs on the circular screen setup (hereafter called the visualisation application), we describe both of them in this section.

They both parse an OBJ file representing the surroundings (based on Open-StreetMap data) of the building that designers are working on and generate 3D shapes that replicate that data inside the virtual environment. They then place the geometry coming from Grasshopper at the correct location inside that environment, thanks to the geolocation data contained in the message. The inclusion of these surrounding buildings provides users with the option to consider the vicinity of the designed project when working on it.

The visualisation application displays three views, presented in Figure 3.15, that are based on three cameras placed in that environment. First, there is a top-down view, based on an isometric camera placed above the target location, that replicates the bounding box of the table display as shown on Figure 3.15a). The second camera is a standard perspective camera looking at the geometrical representation, therefore providing a perspective view, as seen on Figure 3.15c. Finally, a first-person view is provided by another perspective camera placed at human height, as shown in Figure 3.15b.

When users interact with the tangible item that controls the human view, the corresponding camera is teleported at the given location and rotates according to the given angle.

The VR application only displays the first-person view but also listens to first-person view position updates so as to teleport the user to the right location. We do not apply these updates in a continuous manner to avoid

(a) Top-down view. (b) First-person view. (c) Perspective view.

Figure 3.15: The three views available to the user of the GHXR system, on the circular screen setup.

Figure 3.16: The point of view of an AR user interacting with a widget mapped to a parameter, while visualising a hologram of the geometrical representation at the target location on the plan view displayed on the table.

cybersickness issues, and instead use an approach where an update is only applied if no other update quickly follows that one. This introduces a small delay but ensures that only stable data is processed.

In between position updates coming from the table widget, the VR user is free to explore the environment by teleporting himself using VR controllers (pointing metaphor with a curved beam) and looking around (by moving his head).

3.4.5 Augmented Reality application

We additionally developed a Hololens application that also connects to the MQTT broker, in order to retrieve geometry data, so as to create a 3D hologram to be placed on the table, as shown in Figure 3.16. Due to the limited processing power available on the Hololens headset, it was not reasonably possible to show the surrounding buildings in AR, even though the code would have easily been adapted.

The GPS data included in the geometry messages, is enough to properly locate the geometrical representation relative to the "virtual" bounding box, but it is still necessary to somehow recognise the physical table's position to map its surface to the bounding box and place the hologram on it at the correct scale.

To do so, we rely on paper markers, recognised by Vuforia[15], and placed in two opposite corners of the table. These corners are then mapped to the GPS-based bounding box and the hologram can therefore be placed relative to them.

An additional particularity that we handled was the conversion of GPS coordinates between the EPSG:3857[16] and the EPSG:4326[17] formats. Both formats are based on the approximation that the earth is close to a reference ellipsoid, that serves as the basis for a coordinate system.

In the case of EPSG:4326, coordinates are given in latitude and longitude, i.e., degrees of deviation from a reference meridian and parallel (respectively). This is the type of system humans generally refer to when talking about a GPS position.

EPSG:3857, also called the pseudo-Mercator system, uses a projection of the previous coordinate system onto a plan, resulting in a rectangular world map. It is used internally by most computer-based map software, even though such software often offer a user interface that supports EPSG:4326.

The surrounding buildings we get from OpenStreetMap are enclosed in a bounding box given in the EPSG:4326 format, while the tangible table sends its own bounding box in the EPSG:3857 format. As mentioned earlier, since humans tend to use EPSG:4326, we also expect to receive coordinates in that format for geometry position data. This means that we need to be able to convert coordinates from one system to the other.

The following formulas can be used to do so, with EH representing half of the circumference of the Earth at the Equator, ℓ a latitude, and L a longitude:

EPSG:4326 $\rightarrow$ EPSG:3857: $\begin{cases} \ell_{3857} = \frac{\log_{10}(\tan(90+\ell_{4326})*\frac{\pi}{360})}{\frac{\pi}{180}} * \frac{EH}{180} \\ L_{3857} = L_{4326} * \frac{EH}{180} \end{cases}$

EPSG:3857 $\rightarrow$ EPSG:4326: $\begin{cases} \ell_{4326} = \frac{\arctan(\exp(\ell_{3857}*\frac{\pi}{EH}))*360}{\pi-90} \\ L_{4326} = L_{3857} * \frac{180}{EH} \end{cases}$

Parts of the code also compute distances between two GPS positions (e.g., to place the geometrical representation at the specified location, relative to

---

[15] http://vuforia.com

[16] https://epsg.io/3857

[17] https://epsg.io/4326

the virtual environment's boundaries). To do so, we base ourselves on the haversine formula that is widely used in navigation. It computes the distance between two points on a sphere, therefore assuming the Earth is spherical but still manages to keep the error below 0.1%, even for long distances[18].

The exact formula we use to calculate the distance d between two points (ℓ_1, L_1) and (ℓ_2, L_2) is as follows:

$$d\Big((\ell_1, L_1), (\ell_2, L_2)\Big) = ER * 2 * \arctan2(\sqrt{\alpha}, \sqrt{1-\alpha})$$

with: $\begin{cases} ER, \text{ the Earth's radius (since the formula considers it is a sphere)} \\ arctan2, \text{ computing the arctan and returning an angle that can be} \\ \qquad \text{in any of the four quadrants of the trigonometric circle} \end{cases}$

and where: $\begin{cases} \alpha = \sin(\frac{\widetilde{\ell_2}-\widetilde{\ell_1}}{2})^2 + \cos(\widetilde{\ell_1}) * \cos(\widetilde{\ell_2}) * \sin(\frac{\widetilde{L_2}-\widetilde{L_1}}{2})^2 \\ \widetilde{\ell_2} = \ell_2 * \frac{\pi}{180} \\ \widetilde{\ell_1} = \ell_1 * \frac{\pi}{180} \\ \widetilde{L_1} = L_1 * \frac{\pi}{180} \\ \widetilde{L_2} = L_2 * \frac{\pi}{180} \end{cases}$

3.5. Validation

The GHVRBridge prototype was tested at various events, including several workshops at the Faculty of Architecture and Urban Planning (UMONS) and through demonstrations at the National School of Architecture in Nancy. While the participants were mostly enthusiastic about the prospect of modifying an AD project in VR, some of them mentioned the need to go further than parameter adjustment, with a few of them suggesting to move away from the panel-based interface (seen as mimicking standard desktop software with a point and click interaction). Both of these types of suggestions led us to develop the other 2 prototypes, with GHXR pushing the interface towards a more natural interaction and GHVRGraph (described in Chapter 4) enabling control over the visual algorithm itself.

As for the GHXR prototype, we intended to follow a more rigorous validation, and planned on inviting potential users to validate the system on site. We wanted to reach participants with different profiles (both professional practitioners and individuals with a more academic background, including students, researchers and professors) and varying experience with AD tools.

---

[18]https://docs.microsoft.com/en-us/dotnet/api/system.device.location. geocoordinate.getdistanceto

Unfortunately, due to the sanitary situation (COVID-19 pandemic), these evaluations had to be moved to an online setting and were therefore delayed. For that purpose, we implemented an alternative version of the system, that runs on a standard desktop computer and relies on a simulator for the tangible table.

We designed an evaluation protocol that lasts for around 40 minutes and includes questions on the participant's background, a presentation of the GHXR system, a task to perform and post-task interview. The complete protocol can be found in Appendix A, including the post-task interview that mostly consist of open-ended questions on different aspects of the system. It also mentions a questionnaire, available in Appendix B, based on the System Usability Scale (SUS) (Brooke, 1996) that evaluates the usability of a system based on 10 criteria. The questionnaire we used includes an additional question about the overall user-friendliness of the system (through a 7-point scale from "worst imaginable" to "best imaginable") at the end of the survey, based on (Bangor et al., 2008).

We were only able to convince 3 architects, with limited experience outside of the University context, to take part in these evaluations at the time of writing this dissertation so it is hard to draw conclusions on the otherwise excellent average SUS score we achieved (86/100) with the limited number of participants, but it at least matches the way they rated the system's overall user-friendliness (2 chose "excellent" and 1 settled for "good").

The open-ended questions brought their own light to the evaluation process. For instance, all 3 participants indicated that the surroundings were useful and that their level of detail (surrounding volumes instead of a photo-realistic representation) was sufficient for early design stages. They also all mentioned how the system's tangible interface was intuitive and how important that was for an interactive setup; they therefore see potential in the system to help with discussions between architects and other stakeholder, including clients, urban planners and construction companies.

There was however no consensus on what immersive technology was the most appropriate or useful. While 2 participants indicated the AR holograms appearing on the table were not essential to the system with a VR view being available, the other participant stated the AR visualisation was the most interesting of the available views, because it allows for face-to-face discussions while visualising a project in 3D. This tends to indicate that systems that enable both AR and VR visualisations should be preferred, so as to allow users to choose the most appropriate medium for their specific situation.

3.6. Discussion

The two prototypes (GHVRBridge and GHXR) presented in this chapter enable VR-based visualisation of geometries, so that designers and other stakeholders can see full-scale versions of such geometries through an immersive virtual medium. They can therefore experience what the final project will look like from different realistic angles before it is built.

Both proof-of-concept research prototypes go further than simple visualisation tools and even allow for more than extrinsic modifications to the geometries since they both address the adjustment of parameter values for AD definitions within immersive environments. The designer therefore does not need to remove the immersive headset to change these values, and changes to the geometry are automatically forwarded to the immersive experience so that the new version can immediately be evaluated. This is a game changer compared to simple "VR exporting" tools, resulting in a smoother integration of the technology into AD practice, making it possible to move towards a user-centred design approach that involves stakeholders during the design process.

The two hypothetical use cases introduced at the end of Section 3.1 indeed are likely to benefit from the developed prototypes. GHVRBridge is particularly suited for Use Case 1, while GHXR better correspond to Use Case 2.

The two developed prototypes differ in many ways, as summarised in Table 3.2. We would be tempted to say GHXR is more advanced in general, since it offers all the features of GHVRBridge, and adds to it the ability to see the surrounding buildings, the option to change the geometrical representation's position, and the possibility to share an additional parameter type (lists). It also is more accomplished, with more visualisation options (AR and non-immersive displays) as well as more adapted user interactions (through TUIs) that naturally favour collaboration. This overall superiority is quite logical considering we developed GHVRBridge in 2017-2018, and GHXR in 2021. GHXR could therefore benefit from prior developments made for GHVRBridge and was built upon them.

This does not mean that GHVRBridge has become irrelevant since the costs linked to the hardware requirements of GHVRBridge are much lower than those of GHXR. Another consideration is that the collaborative extension of GHVRBridge was designed with *remote collaboration* in mind, meaning it is optimised for that purpose as explained in Section 3.2.1. GHXR was instead built for a *local setup* and works with a simple JSON format, leading to additional delays when system components have to communicate over the

	Prototype 1: GHVRBridge	Prototype 2: GHXR
Control	parameter values, navigation	parameter values, navigation, geometry position
Interaction paradigm	3D-adapted WIMP	TUI
Geometry visualisation	at scale	at scale, perspective, top-down
Surroundings visualisation	not available	OpenStreetMap buildings data
Visualisation modalities	VR	VR, AR, non-immersive
Hardware cost	~2,000€, including a computer	~30,000€
Transfer protocol	WebSockets	MQTT
Collaboration	local or remote	mostly local, but VR remote visualisation is possible

Table 3.2: Comparison of the GHVRBridge and GHXR proof-of-concept research prototypes, that enable parameter adjustment of AD definitions within immersive environments.

internet. Even though the format could be changed to a more optimised one, the table display is where the control over the system is located, so remote collaboration would be limited to visualisation for off-site collaborators.

These characteristics mean that the two developed prototypes target different use cases, projects, and users. Individuals and small companies should be able to afford a system such as GHVRBridge (considering they can afford Grasshopper and therefore Rhinoceros in the first place), while only larger companies and organisations can potentially allow themselves to buy the necessary equipment for GHXR. With the many displays and views as well as the more natural interaction and collaboration, GHXR is inherently better suited for discussions that involve more collaborators than GHVRBridge, and for bigger projects (e.g., at the urban scale, to discuss with public authorities). The ability to see the surrounding buildings in GHXR also implies that it can be used to assess how a designed geometry integrates with its environment, meaning the prototype may be used for projects where the vicinity is of particular importance.

It is also worth noting that both prototypes can only handle a certain number of simultaneously modifiable parameters due to their "client" applications. While both Grasshopper custom components can virtually handle any number of parameters, GHVRBridge's VR application uses a panel that is only large enough for 6 simultaneous parameters and GHXR's table device can only track 32 tangible items (about 25 of those being available for parameters, but even that number would induce significant clutter on the table, leading to an unpleasant experience).

While GHVRBridge's panel could be extended (e.g., made larger or scrollable when more parameters are shared), the limit on GHXR's tangible items is hardware-related and adding support for additional parameters would only be possible through a rotating approach (allowing a dynamic mapping of parameters to tangible items e.g., through a side panel on the table display that would allow users to select which parameter to assign for a particular tangible item). At the same time, the need for a larger number of adjustable parameters is questionable, since both systems enable rapid VR-based collaborative iterations and it seems more appropriate to limit the flexibility to a few crucial parameters.

As reported in Section 2.5.2, since the development of GHVRBridge in 2018, other companies and research teams have commercialised and published similar tools, such as Fologram[19] and the previously mentioned prototype (Hawton et al., 2018). Our point of view on the integration of immersive tech-

---

[19]`www.fologram.com`

nologies for AD activities therefore is not an isolated opinion, as confirmed by the informal feedback we received from architects, students and laymen during the visits, workshops, and exhibitions we took part in.

While we mostly received positive feedback when presenting GHVRBridge and GHXR to architects (as stated in the previous Section), some mentioned the need to go further than mere parameter values and allow users to modify the AD visual algorithm itself (components and links, what we called *Level 3* earlier) since this would provide greater flexibility in terms of possible adjustments to the design. This pushed us to explore that possibility, which will be presented in Chapter 4.

Immersive visual programming for Algorithmic Design

> "The ultimate display would, of course, be a room within which the computer can control the existence of matter."
>
> Ivan Sutherland

According to the prototype-based iterative research process described in Section 1.4, the proof-of-concept applications enabling *Level 2* control over AD definitions (as per Table 3.1 in Section 3.1), that we presented in Chapter 3, were subject to evaluations. When presenting our VR-based solutions for parameter value adjustment to architects, some of them requested to have more control over the AD visual algorithm. Parameter value adjustment was indeed often considered to be too restrictive. This lead us to explore VR-based *Level 3* control, i.e., the ability to edit the visual algorithm, by adding or removing components or links between them.

The present chapter discusses how we achieved that goal, through the implementation of a research prototype called **GHVRGraph**. We again develop a VR-based editing tool but explore different interaction modalities, by relying on different techniques borrowed from Section 2.4 that we implement with 6-DoF controllers as well as a hand-tracking device, but also with speech-based interaction. We then reflect on that prototype and provide insights as to how such features should be implemented in an immersive context, with adapted interaction mechanisms.

4.1. A graph representation of Algorithmic Design definitions

The GHVRGraph research prototype tool enables VR-based *Level 3* control over AD definitions. Architects using it are therefore able to manipulate components and links from the visual algorithm, from an immersive environment.

Before diving into the implementation of the GHVRGraph prototype, we had to reflect on how to represent AD visual algorithms using an appropriate formalism that leads to a convenient data structure to be used for the prototype's implementation.

We use part of the Grasshopper definition shown in Figure 2.22 as a running example to discuss the representation of visual algorithms with that formalism. The sub-part of the definition that we keep for this purpose is shown in Figure 4.1 and includes two number components (sliders), one text panel containing a formula, and two components with different input and output ports.

Figure 4.1: Example of a Grasshopper definition, part of the more complete definition from Figure 2.22 generating the conical spiral displayed on the right side of this figure.

As discussed in Section 1.2, AD algorithms, such as those created with Grasshopper, can be seen as a type of dataflow models. Dataflow models can be represented with directed graphs, that we formally define in Definition 4.1.1.

Definition 4.1.1. A directed graph (digraph) is an ordered pair (V, E) such that:
$$\begin{cases} V & \text{is a set of vertices} \\ E \subseteq V \times V & \text{is a binary relation over } V \text{ that defines a set of directed edges} \end{cases}$$

Since Grasshopper definitions cannot contain a loop (i.e., it is not possible to have a link from a component A to a component B if there exists a succession of links that already connect B to A), the corresponding directed graphs would be acyclic. We therefore formally define **Directed Acyclic Graphs (DAGs)** in Definition 4.1.2.

Definition 4.1.2. A (vertex-labelled) Directed Acyclic Graph (DAG) is a triple $(V, E, vlabel)$ such that:

$$\begin{cases} (V, E) & \text{is a digraph} \\ E & \text{is an acyclic relation} \\ vlabel : V \to Label & \text{is a total function that assigns a label to each vertex} \end{cases}$$

where *Label* represents the set of all possible labels.

We would be tempted to convert AD definitions to DAGs, representing components as vertices in such graphs, and links between components as edges. However, Grasshopper algorithms also require to specify input and output ports on components. Examples of such ports in Figure 4.1 include the ones from the `Range` component, that has two input ports, respectively for the `Domain`, `Steps` parameters, as well as one output port returning the resulting `Range`).

For that reason and as exhibited on Figure 4.2, a basic (vertex-labelled) DAG is not enough to convey that information: it is no longer possible to know which ports are used by the edges based on that representation alone, and that information would consequently be lost.

A solution to that problem would be to additionally label the edges with the ports they are connected to, but we discarded that option because of implementation details. It would in fact be impractical since we would have to iterate over all the edges going from or to a component, in order to find the edges that only concern a specific port from that component. Navigating the graph and converting it to the Grasshopper file format when saving changes would then become more complex and may lead to clumsy code. We therefore opted for another solution representing ports as vertices, that we describe hereafter.

We chose to represent AD definitions using a specific graph-based formalism that is well-known in graph transformation theory (Ehrig et al., 2004). To be more specific, we decided to make use of a **type graph** in order to formally specify what a valid AD definition is composed of, and **typed graphs** to represent such definitions.

This is similar to the representation of software design models (e.g., statecharts and class diagrams) in (Mens, 2005), where typed graphs are used in order to specify model refactorings using graph transformations.

Definition 4.1.3 formally defines the notion of typed graph, as we used it in the context of visual AD.

Figure 4.2: An oversimplified DAG representation for the example previously given in Figure 4.1. Ports are not represented and the corresponding information is lost.

Figure 4.3: The typegraph for the graph structure. An arrow from one element A to another element B indicates that it is possible for an edge to connect an element of type A to an element of type B.

Figure 4.4: The vertex-labelled, typed, acyclic directed graph for the example given in Figure 4.1, converted to the typed graph representation.

Definition 4.1.3. A typed graph G is a 5-tuple $(V, E, vlabel, TG, vtype)$ such that:

$$\begin{cases} (V, E, vlabel) & \text{is a (vertex-labelled) DAG} \\ TG = (TV, TE) & \text{is a digraph} \\ vtype : V \to TV & \text{is a total function that assigns a type to each vertex} \\ \forall (v_i, v_j) \in E : & \big(vtype(v_i), vtype(v_j)\big) \in TE \end{cases}$$

We call TG the type graph and G the typed graph, and we say that G is typed over TG.

In the case of AD visual algorithms, the digraph that defines valid definitions is depicted in Figure 4.3, and formally described in Example 4.1.1.

Example 4.1.1. The digraph $TG = (TV, TE)$, that is depicted in Figure 4.3, describes valid AD graphs, with:

$$\begin{cases} TV = & \{\text{IOComponent}, \text{InputPort}, \text{OutputPort}, \text{PComponent}\} \\ TE = & \{(\text{InputPort}, \text{IOComponent}), (\text{IOComponent}, \text{OutputPort}), \\ & \quad (\text{OutputPort}, \text{PComponent}), (\text{PComponent}, \text{InputPort}), \\ & \quad (\text{OutputPort}, \text{InputPort}), (\text{PComponent}, \text{PComponent})\} \end{cases}$$

Figure 4.4 depicts the typed graph that corresponds to the running Grasshopper definition example from Figure 4.1.

This representation of Grasshopper definitions resembles the Generalized Parametric Model (GPM), proposed in (Janssen & Stouffs, 2015), which also includes a reflection on how to represent AD visual definitions using graphs. The GPM distinguishes data nodes from operation nodes (geometric or computational processing of incoming data), and considers ports as data nodes. The model therefore relies on an extended version of a DAG, where these two types of nodes are allowed. We however need to differentiate predefined ports from input data and output channels at the interaction level, to prevent invalid modifications to the AD definition (e.g., removing a link from an input port to its associated component).

While conceptually interesting, the idea of combining ports and data into a single notion of "data node" is therefore impractical for implementation purposes. The authors behind GPM indeed state that they only intend to provide an analytical device, not discuss actual implementation (Janssen & Stouffs, 2015). We therefore rely on the presented typed graph formalism to represent AD visual definitions.

That formalism also has the advantage that it comes with tools such as AGG Taentzer (1999), allowing automated checking of graph transformations and constraints, so we could have relied on such tools to validate manipulations

made to the graph representation in our application. Unfortunately, AD visual algorithms come with specific constraints (e.g., most components have a set of input and output ports that is predefined and cannot be changed by the user) and concepts (e.g., number parameters come with specific settings restricting the range of possible values) that make it hard to fully rely on such a formalism. We therefore opted to constrain and check the validity of manipulations directly in our code.

In order to prevent the aforementioned invalid modifications from occurring, GHVRGraph must for instance prevent users from adding or removing an edge (link) between a port and its component. It should also not allow them to add or remove a port on a component.

4.2. Interoperability with Grasshopper

In order to replicate the representation discussed in the previous section and to allow users to manipulate the resulting graph, we rely on the Unity-specific version[1] of the QuickGraph[2] library. This allows us to preserve a certain degree of genericity since other graph-based representations could rapidly be adapted to work with the VR-based environment and interactions we developed. The library also enables us to use graph algorithms such as topological sorting and leaf-finding functions.

4.2.1 Defining vertex objects

The QuickGraph library works with graphs whose vertices can only be of a single concrete type i.e., the corresponding class cannot be abstract. We therefore circumvent this limitation by defining a concrete `Vertex` class that contains a *chunk* field. That field corresponds to an abstract `Chunk` class, from which we can define a hierarchy of vertex types to allow us to handle the different types of vertices required for representing AD definitions.

The hierarchy of classes we use for the GHVRGraph prototype tool is shown on the `GHElements` package in the class diagram of Figure 4.5. It is mostly based on the internal representation of the corresponding elements in the Grasshopper desktop tool. All vertex types indeed inherit from the aforementioned `Chunk` abstract class, and a vertex can be of type `Port`, `Component`, or `Group`.

Except for the `Group` class that will be discussed in Section 4.2.3, these

---

[1]`https://github.com/davidgutierrezpalma/quickgraph4unity`
[2]`https://github.com/YaccConstructor/QuickGraph`

classes are also abstract, and concrete objects must therefore be of a more specific type. An `IoComponent` is a component that has at least one input or output port, and can have both types of ports, while a `PrimitiveComponent` correspond to a vertex that contains input data or is an output channel (a `PanelComponent` can be both).

We additionally created a `GenericPrimitiveComponent` to handle primitive components that do not fall into the previous three categories. This allows us to successfully read from and write to Grasshopper files that contain such components.

4.2.2 Converting Grasshopper files to a graph-based representation

While the exact implementation details are out of scope of this document, we will briefly discuss the main ideas as well as the Grasshopper file format and a challenge we encountered when parsing such files. When saving a definition within the Grasshopper desktop application, the user is offered two format options: the default binary `.gh` format, or a XML-based `.ghx` format.

A Grasshopper definition is essentially stored as a list of XML (Yergeau et al., 2008) entities called *chunks*. In addition to "header" chunks that contain the likes of versioning information, timestamps and some user preferences, most of the content in such a file is within the `DefinitionObjects` chunk, that contains "sub-chunks" that correspond to interactable entities such as components and input data. The corresponding XML element contains various information, such as a type identifier, a name, the display boundaries within the definition, and an instance identifier that uniquely identifies an element in the definition. Depending on the particular type of element, sub-chunks may be present, for instance to give information about a component's ports.

Links are indirectly stored within these sub-chunks, via `Source` items, that indicate the instance identifiers of entities that link to the current entity. In order to obtain the vertex-labelled typed graph, we must process these `Source` items and therefore construct the graph backwards (with regards to the direction of the edges).

Since there is no guarantee that a unique identifier referenced by a `Source` item corresponds to an element that has already been processed, we need to be able to create temporary placeholder components, to construct the graph with a "placeholder" for the expected component until we reach that component's description in the file.

Figure 4.5:   A section of the class diagram for GHVRGraph, showing the hierarchy of classes we use to represent Grasshopper "chunks" in the research prototype, as well as how the graph structure is stored using QuickGraph.

4.2.3   Groups and clusters

In Grasshopper, there is a grouping concept that allows designers to divide and structure their work in sets of components, similarly to how a software developer would separate some code into multiple classes and files. Examples of **groups** can be seen on Figure 1.2, where they are represented by coloured rectangles that enclose all their components. In the Grasshopper file format, a group is represented as a chunk that contains a list of one or more `ID` items to reference the constituents' unique identifiers.

There is no operational semantics linked to that group concept, in the sense that placing components in a group cannot modify the generated output geometry. Since groups only constitute a visual aid for the designer, we do not include them into the typed graph representation but still create a `Group` class to match the grouping concept and handle such groups in GHVRGraph. This allows us to display them in the VR-based representation and preserve the grouping information in the save files generated from GHVRGraph.

A related element in Grasshopper is the concept of a **cluster**, that also bundles components together. Unlike groups however, it serves as a black box that summarises its constituents' input and output ports. This black box is reusable, in the sense that multiple copies of a cluster can be included in the definition.

A user can choose to look into a cluster that was created earlier and modify its constituents ; doing so will impact all copies of that cluster. The cluster concept therefore goes beyond simple visual aid, but clusters are unfortunately stored as an encrypted item in the Grasshopper file format. We consequently cannot fully support clusters in GHVRGraph, because we have no way of allowing VR users to "open" a cluster and interact with its constituents.

4.2.4   Immersive visual representation

In order to allow designers to manipulate Grasshopper definitions in VR, we needed to represent the typed graph, that we construct from the imported Grasshopper definition, within the immersive environment. To provide VR users with interaction mechanisms that are appropriate to the 3D context, the graph representation should be three-dimensional. That representation should allow VR users to easily manipulate interactable objects such as components and links, through the corresponding vertices and edges. The 3D representation should also maintain a mapping with the Grasshopper desktop (2D) canvas, so that changes to the placement of components are preserved when switching between the desktop tool and the GHVRGraph application (e.g.,

Figure 4.6: A Grasshopper definition and its corresponding immersive visualisation in GHVRGraph, showing the table metaphor we use.

moving an input parameter closer to a related input or output component).

We decided to rely on a table metaphor to represent the typed graph in VR within GHVRGraph. Figure 4.6 shows a side by side comparison of the same AD definition, represented in the Grasshopper desktop tool on the left, and in GHVRGraph on the right. In front of the VR user's starting position, we place a virtual table on which we add boxes that roughly resemble the corresponding Grasshopper components, albeit in a three-dimensional form.

Most of the developments for the GHVRGraph prototype happened during the one-month long 2019 edition of the eNTERFACE workshop[3], where we gathered early feedback from workshop attendees (about 60 participants). Based on their reports and on the literature on ergonomics, that suggests the preferred display zone should be below the horizontal line of sight (Pheasant & Haslegrave, 2018), we opted for the metaphor of a virtual table instead of a board-based metaphor.

Another suggestion we received from the aforementioned workshop participants was to slightly incline the table, since this would allow the user to place himself in a position where the table is angled towards him. In fact, a flat and horizontal surface would require the user to incline his neck further in order to look down on the table. The suggestion to incline the surface is supported by the recommendation on viewing angle in (Pheasant & Haslegrave, 2018) and is corroborated by a study specifically addressing neck pain (Yip et al., 2008), that concludes that the associated pain decreases as the user gets closer to a forward head posture (i.e., a 90° angle between the neck and the horizontal line of sight).

---

[3]http://web3.bilkent.edu.tr/enterface19/

Techniques / Actions		Modality			Interaction type			
		6-DoF controller	Hands	Speech	Direct	Indirect	Grasping	Pointing
Component	Add	P_1	P_2	P_1^{speech}, P_2^*	P_1, P_2	P_1^{speech}, P_2^*	P_1, P_2	P_2
	Remove	P_1	P_2	P_2^*	P_1, P_2	P_2^*	P_1, P_2	P_2
	Move	P_1	P_2	P_2^*	P_1, P_2	P_2^*	P_1, P_2	P_2
Link	Add	P_1	P_2	P_2^*	P_1, P_2	P_2^*	P_1, P_2	P_2
	Remove	P_1	P_2	P_2^*	P_1, P_2	P_2^*	P_1, P_2	P_2

Table 4.1: Modalities and interaction types we explored to implement VR-based graph-editing actions in the variants of GHVRGraph, P_1, P_1^{speech} and P_2. The asterisk sometimes used for P_2^* denotes that the modality or interaction type is used for the corresponding action but needs to be combined with another modality or interaction type to produce a result.

4.3. Exploring interaction techniques for graph manipulation

Displaying a 3D rendering of a graph-based AD definition on a virtual table is only a first step. Designers need ways to manipulate components and links, not just visualise them. Based on the immersive interaction techniques identified in Section 2.4 as well as popular and affordable devices, we explored different options to enable "basic editing" of AD definitions and their underlying typed graphs in VR. By "basic editing" we understand the ability to add and remove components and links (not individual ports since we want to preserve a valid mapping with Grasshopper). We additionally allow users to move components, i.e., position them differently, without changing their connections.

The set of interaction devices we used to explore VR-based graph editing included: (1) the HTC Vive HMD together with its associated 6-DoF controllers; and (2) the Leap Motion hand-tracking system. Using them, we explored different techniques as different variants of the same GHVRGraph prototype. Table 4.1 gives an overview of these prototype variants, named P_1, P_1^{speech} and P_2, and how they cover various modalities and techniques, based on the categories we presented in Section 2.4.

4.3.1 Variant P_1: grasping metaphor with 6-DoF controllers

Variant P_1 of GHVRGraph relies on the default 6-DoF controllers provided with the HTC Vive headset. We chose for an isomorphic interaction technique based on the grasping metaphor: the user simply touches the element he wants to interact with, and presses a button to trigger the corresponding action. Figure 4.7a shows a user that is about to grasp and start moving a component,

| (a) Grabbing a component. | (b) Adding a link. | (c) Removing a link. |

Figure 4.7: A user interacting with a 6-DoF controller in variant P_1 of GHVR-Graph.

while Figures 4.7b and 4.7c show a user interacting with links.

The user selects and interacts with an element by pressing a button on the controller when in reach of the said element. If the selected element is a link, we simply remove that link from the graph. If it is a component, we attach it to the controller (that type of interaction is often referred to as the grasping metaphor). The user can then choose to release it elsewhere on the table (realising the "move vertex" action) or throw it away ("remove vertex" action). In order to add a link between two components, the user needs to select two ports consecutively. After selecting the first port and prior to selecting the second one, a temporary line between the selected port and the controller is rendered so as to give feedback to the user on the port that has been interacted with. Note that adding a link is prevented if that link would create a cycle in the graph structure (since Grasshopper does not allow that to happen).

A video demonstrating variant P_1 of GHVRGraph is available online[4] and our codebase is hosted as open-source software on a GitHub repository[5], available under the MIT[6] license. We also published a paper that further describes the system and how we explored different interaction techniques (Coppens et al., 2019).

4.3.2 Variant P_1^{speech}: Speech recognition

In addition to direct interaction techniques, specific actions can be simplified by relying on indirect approaches such as speech recognition. Since specifying an arbitrary position or selecting an existing object is easily and rather naturally done with a direct technique (e.g., relying on a grasping or pointing metaphor), speech recognisers are often used in a multimodal context when applied to 3D

---

[4]http://informatique.umons.ac.be/staff/Coppens.Adrien/?video=eNTERFACE2019
[5]https://github.com/qdrien/Grasshopper-VR-graph
[6]https://opensource.org/licenses/MIT

selection or manipulation tasks (e.g., the "Put-That-There" metaphor (Bolt, 1980)). We decided to explore this modality and integrate it as part of and extended version of P_1: variant P_1^{speech}.

An important distinction between speech engines is whether (and how much) they restrict potential input. Free speech recognisers can output any text, whereas directed dialogue (Pieraccini & Huerta, 2005) systems are limited to a set of predefined words or commands. Directed approaches can mostly be found in two forms: keyword-spotting solutions that extract specific words; and grammar-based tools that produce phrases defined by specific rules.

GHVRGraph, as a VR-based graph editing application, would benefit from vocal commands such as "Add component X" to create new components in the graph. Even though free speech and keyword-based approaches could be used for that purpose, they would not guarantee that a valid output is returned by the speech recogniser and would require post-processing of that output to parse it (which sequences of words or keywords are valid, and what action they correspond to). Grammar-based engines therefore seem to be the best option as only valid vocal commands, with regards to the grammar, can be recognised.

Since GHVRGRaph is developed with the Unity game engine, we can benefit from the engine's built-in support for the Windows Speech Recognition API[7], that includes an XML-based grammar recogniser for Speech Recognition Grammar Specification (SRGS). SRGS is a W3C standard[8] that describes a grammar format. Similarly to the grammars from compiler theory (Aho et al., 1986), a SRGS grammar describes a set of rules composed of tokens, using either an XML or an augmented BNF (Backus-Naur Format) syntax.

SRGS grammars can be augmented with Semantic Interpretation for Speech Recognition (SISR[9]) tags that contain ECMAScript (JavaScript) code to be executed when the corresponding grammar rule is matched. Those tags are typically used to assign values to variables for a matched rule. For instance, a boolean variable can see its value set to `true` when the matched text is "yes", "ok" or "yeah". Similarly, such tags can handle numbered values, so that a user saying "three" could assign the value `3` to a certain variable (e.g., called `outValue`). It is also possible to go further and perform specific operations on such a variable when specific words are recognised. For example, saying "thousands" could multiply the previous `outValue` variable by `1000`.

While SRGS alone is enough to define commands such as "Add component

---

[7]`https://docs.microsoft.com/en-us/windows/apps/design/input/speech-recognition`

[8]`https://www.w3.org/TR/speech-grammar/#S1`

[9]`https://www.w3.org/TR/semantic-interpretation/`

```xml
<rule id="Add_phrases">
    <one-of>
        <item>add</item>
        <item>add component</item>
        <item>create</item>
        <item>spawn</item>
    </one-of>
</rule>
<rule id="Basic_components">
    <one-of>
        <item>point
            <tag>out.type = "point";</tag>
        </item>
        <item>circle
            <tag>out.type = "circle";</tag>
        </item>
```

[...]

(a) A visual representation subsuming the grammar rules used to add components.

(b) Part of the corresponding definition in the XML-based SRGS format.

Figure 4.8: The SRGS/SISR rules used to provide vocal commands allowing users to add components in P_1^{speech}.

circle" or "Add boolean toggle", the capabilities of SISR are interesting to allow GHVRGraph users to add valued components directly (in one go), with commands such as "Add slider with value 7" or "Add boolean toggle with value true". While alphanumeric input in grammars is non-trivial (Wang & Ju, 2004), we relied on an existing set of rules provided in the Microsoft Speech Platform SDK[10].

Figure 4.8 presents part of the rules we use to provide vocal commands allowing users to add both valued and non-valued components in P_1^{speech}. Figure 4.8b correspond to a small portion of the rules in the textual (SRGS) format, while Figure 4.8a clarifies the general structure of the "add component" command.

As users may want to assign an arbitrary text value to a vertex (e.g., a panel component), we also incorporated a free speech recogniser, that starts listening to user input only when specific grammar rules have been processed. In the meantime, the grammar recognition engine is paused, and it only resumes when the user stops providing free speech input.

A potential lead to further improve the integration of the speech modality in GHVRGraph would be to rely on machine learning techniques. Such techniques could be used to process free speech input and directly convert it to valid commands (with regards to the SRGS grammar). There would be no need to

---

[10]https://www.microsoft.com/en-us/download/details.aspx?id=27226

split the recognition of text-valued components from other components. This could also increase the flexibility of the system, in the sense that alternative wording for the same command could be learned by the machine learning algorithm, that would convert that wording to its equivalent in the grammar specification.

4.3.3 Variant P_2: grasping and pointing metaphor with a hand-tracking system

The goal of variant P_2 of GHVRGraph is to explore interaction techniques while relying on a hand-tracking sensor. We used the Leap Motion[11] sensor, that is composed of two standard colour cameras and infrared LEDs, that give it the ability to scan the space above its surface, up to ∼60 cm away from the device. The development of this variant was started during the previously mentioned eNTERFACE workshop and continued as part of a Computer Science master student project, under my direct supervision (Willième, 2020).

P_2 comes in two modes: grasping and pointing. The former replicates the grasping metaphor used by P_1, replacing VR controllers by the user's hands with a grasping gesture acting as a substitute for the controller's button press to start a manipulation. Once grabbed, a component can be moved and the manipulation stops when the user releases his hand. Interacting by grasping objects is natural to humans, but it only allows users to interact with objects that are sufficiently close to the user, at hand's reach.

The pointing-based mode mitigates that limitation by expanding the range of interaction for VR users to objects they can point to, by using a raycasting method (see Section 2.4). The tip of the forefinger from the user's main hand is used as the starting point of the beam, using the closest phalanx to derive the beam's direction (from the phalanx to the tip). To trigger an action, the user speaks a simple "action" vocal command. Depending on the type of object that is pointed at and the state of the application, the corresponding action is performed. For instance, if nothing is selected and the user triggers the vocal command while pointing at a component, that component will be selected and become moveable, until it is released by another "action" command.

The grasping mode is illustrated in Figure 4.9, where a user is about to grasp a component to manipulate it. The component is highlighted in yellow to indicate that the hand is currently touching it and it could consequently be interacted with if the user were to perform the grasping gesture. The pointing mode is depicted in Figure 4.10, split into sub-figures 4.10a and 4.10b, where

---

[11]https://www.leapmotion.com/

Figure 4.9: A user about to grab a component in variant P_2 of GHVRGraph, that relies on a hand-tracking device.

a user respectively selects and moves a component based on his forefinger's position. Implementing both approaches allows to see how they compare in the context of graph-based 3D interaction.

To do so, evaluations based on both interaction modes of variant P_2 were conducted in (Willième, 2020). These evaluations were limited to 4 participants, partly due to the sanitary situation at the time, but showed the main advantages and drawbacks of each method. The grasping metaphor turned out to be intuitive and precise for interacting with elements at close range, but not ideal for objects further away. On the other hand, the pointing mode of P_2 was considered easy to use for both short and close range, but suffered from precision issues, especially for interactions on edges (since they are thin and therefore hard to point at).

Videos demonstrating this prototype are available online, for the grasping[12] and pointing mode[13].

4.4. Immersive visualisation of resulting structures

Similarly to the two prototypes presented in Chapter 3 (GHVRBridge and GHXR), providing VR-based editing capabilities is much more interesting if the output result (the geometrical representation) can be visualised from within the same immersive experience. We can unfortunately not directly borrow the mesh streaming approach from GHVRBridge (of Section 3.2.2). That is because, unlike GHVRBridge, GHVRGraph does not simply apply user mod-

---

[12] https://www.youtube.com/watch?v=Su3y1tnZk1s
[13] https://www.youtube.com/watch?v=oxgMxh4Fkx4

(a) Pointing to a component. (b) Placing the selected component.

Figure 4.10: The two-step process to move a component in variant P_2 of GHVRGraph, illustrating the pointing mode of that variant.

ifications through a custom Grasshopper component, but instead produces a new file in the Grasshopper format. In order to generate the new geometrical representation, we therefore need to reload the file in Grasshopper, and that would disconnect the custom component from the VR experience.

We consequently approached the issue in a different manner and relied on Rhino.Inside[14], a project that embeds Rhino and Grasshopper into other applications. Using Rhino.Inside, we can run an instance of a Grasshopper within our Unity application and communicate with it.

On launch, GHVRGraph starts an instance of Rhino and Grasshopper, through Rhino.Inside. When modifications made to the visual algorithm via the VR interface are saved, a new Grasshopper file is produced and sent to the Grasshopper instance that runs within GHVRGraph. Once received, Grasshopper generates the geometrical representation and it therefore becomes available for sharing. To do so, GHVRGraph includes a custom sharing component inside the file sent to Grasshopper, that thereby automatically sends the geometry (in the same mesh-based format as in Section 3.2.2) to the VR experience.

This rather complex process is pictured in Figure 4.11.

4.5. Validation

Since most of GHVRGraph's development happened during the aforementioned eNTERFACE workshop, we were able to ask other workshop participants (with different backgrounds) to try the prototype along the way and give

---

[14]http://rhino3d.com/inside

Figure 4.11: The proposed approach for VR-based editing of AD definitions with GHVRGraph, enabling geometry visualisation through Rhino.Inside.

their feedback. As previously mentioned, this led us to use a table metaphor with a slightly inclined table for our graph visualisation.

Additionally, as part of the survey discussed in Section 2.5.3 and as mentioned there, we included a video in order to show respondents the GHVR-Graph prototype. All respondents that indicated being aware of AD tools were shown the video, that demonstrated the prototype, with a VR user editing a Grasshopper project. After watching the video, respondents were asked about the usefulness of such VR-enabled functionality.

The answers we received for that question revealed a mixed reception, with slightly over half of the respondents (18 out of 34) considering VR functionality as "probably not useful" or "not useful", as shown on Figure 2.42a (reproduced here on Figure 4.12a).

We however noted a difference depending on the respondents' prior exposure to 6-DoF VR, and therefore further compared the responses with whether the respondents had tried VR tools for architecture-related activities. The corresponding results are plotted on Figure 2.42b (reproduced here on Figure 4.12b), that only includes 20 respondents since the question on VR for architecture was only asked to those who indicated they were familiar with VR and had tried the technology in the first place.

While GHVRGraph seemed to be slightly better received amongst respondents with prior exposure to VR, the population size did not allow us to reach statistical significance.

(a) Usefulness of VR for AD, broken down by respondents' prior exposure to 6-DoF VR devices and experiences.

(b) Usefulness of VR for AD, broken down by respondents' prior exposure to VR tools for architecture (if they were familiar with VR).

Figure 4.12: Usefulness of VR for Architectural Design, reproduced from Figure 2.42.

4.6. Discussion

GHVRGraph, the proof-of-concept research prototype presented in this chapter, goes beyond the work presented in Chapter 3, by enabling VR-based *Level 3* control over AD definitions created with Grasshopper. The prototype allows control over the graph-based AD representation itself, and not just parameter values.

This type of control allows AD architects to make deeper modifications than what *Level 2* offers, and could even lead to architectural geometries being created only through the VR interface. While we do not believe that the latter is a realistic scenario at the moment, the ability to manipulate components and links from the immersive environment certainly opens new possibilities for architects to integrate immersive technologies in their practice.

Such tools however do not truly concern other stakeholders than architects themselves, since modifying the graph-based representation (corresponding to the visual algorithm) requires expertise with the AD paradigm. While this limits the number of potential users for VR-based *Level 3* applications, a more polished version of GHVRGraph that would be delivered as commercial software could be affordable for architectural firms of all sizes, since the hardware requirements are the same as those of GHVRBridge, described in Section 3.2.

The following hypothetical use case help clarify a potential usage scenario

for GHVRGraph, where the prototype tool is likely to improve the design process.

Use Case for GHVRGraph. A small architectural firm is in charge of the construction of a monument for a big garden (10,000 square meters) at the entrance of a small castle converted to a modern art gallery. Commissioned by the gallery's board, the monument should inspire a sense of prestige. The architects are still in an early design phase with multiple options being considered. Based on the current state of the AD definition she created with Grasshopper, one of these architects wants to get a better idea of what these options look like from a human perspective. She therefore starts GHVRGraph with the intent of checking the options in VR, switching between options by editing links in the graph-based VR representation. Once immersed and after checking the initial options, the architect has a new idea: she wants to combine two of the initially proposed options. After rebranching the definition from the VR-based representation, she visualises the result. She then moves back to Grasshopper to refine the design idea based on the version she modified from GHVRGraph.

4.6.1 Towards a collaborative variant of GHVRGraph

While GHVRGraph is a single-user application, it would be straightforward to extend it to enable multi-user co-presence using the solution presented in Section 3.3.1. If all collaborators are placed in the same virtual environment and interact with the same AD definition, solutions similar to the ones discussed in 3.3.2 would be sufficient to enable conflict-free collaboration or at least provide a clear procedure for conflict resolution.

However, if users collaborate through separate representations of the same AD definition, there is no guarantee that these representations are in the same state at any point in time. Changes made to the AD definition by different collaborators must therefore be integrated to form the main joint definition. Despite the graph-based nature of the AD definitions that GHVRGraph works with, it is clear that this problem requires solutions based on software merging (Mens, 2002), or more generally model-based merging techniques (Brunet et al., 2006).

Since collaborators are not guaranteed to produce changes that are compatible with each other, model merging is sometimes not sufficient, and conflicts between incompatible local modifications need to be resolved. This is practically unavoidable when collaborative development occurs in parallel (McKee et al., 2017). The tooling at developers' disposal consequently should integrate visual code comparisons to allow for conflict resolutions to become as smooth as possible.

As for AD and visual programming, none of the popular commercial tools include merging capabilities. However, a system called MACE (Multiple Alternatives - Comparison and Editing), proposed in (Zaman et al., 2017), allows for comparison between DAGs. Using MACE, a designer can choose different versions of an AD definition and the tool will highlight changes between the selected base version and its compared alternatives. It will colour vertices and parameters depending on the type of changes that were made to them (e.g., a green vertex means that the vertex was added). This is comparable to code differencing tools used in software developments to compare different versions of the same code base (e.g., to solve merge conflicts when they occur).

Pushing the inspiration from software engineering further with capabilities that are inspired by version control systems, (Cristie & Joyce, 2019) proposed a system called GHShot, that works with snapshots of Grasshopper definitions. Using that system, a Grasshopper user can push a version of a definition to a centralised server. When doing so, the user indicates whether this new version is a simple progression of the previous one, or an alternative solution that should be included in a separate new branch. Reusing the ideas of MACE, GHShot also proposes a "diff view" that highlights changes between different versions of a definition. To our knowledge, GHShot is the AD system whose features most resemble the capabilities of the version control systems used in software development.

A potential approach, that has been ruled out in GHShot but may still be a viable lead to support merging AD definitions, is to integrate versions of a definition via the textual representation of these versions (the `.ghx` files). Since these contain a textual description of the definition, they allow for text merging techniques to be used. This should work well for collaborators that work on separate parts of a definition, since the XML entities they would create or modify are then different. However, conflicts are unavoidable when collaborators interact with the same components, and solutions such as MACE and GHShot's diff views would still be needed to allow designers to resolve these conflicts.

4.6.2   Opportunities for visualisation enhancements

The current version of GHVRGraph displays the full graph representation to the VR user. This means that bigger graphs will lead to smaller components, since the dimensions of the table are fixed. This effectively makes the current version of GHVRGraph unable to work with large Grasshopper definitions. To make the prototype scalable, zooming and panning (moving the view to another part of the graph) would need to be added to the set of available

actions. It would therefore be interesting to explore interaction techniques to perform these actions in that VR context.

In addition to the exploration of more modalities and interaction techniques that could lead to a better user experience, we could also explore different types of visualisations for AD definitions in VR. We could try to better capitalise on the three-dimensional aspect of immersive environments. A potential approach would be to "unleash" the graphs and allow users to move vertices anywhere in the 3D space, instead of constraining graph components to a (table) surface. As discussed in Section 4.2.4, one of the reasons we imposed that restriction was to preserve a mapping between the VR-based representation and the desktop tool's canvas. If we would let go of that constraint, VR users would get access to more space, and may organise the graph using the whole environment around them, placing vertices in different directions and at different heights.

In addition to the lack of mapping with the desktop software, a 3D graph representation is not necessarily beneficial, at least when the represented data is non-spatial, i.e., not inherently three-dimensional itself (Elmqvist, 2017), as is the case for visual algorithms (as opposed to flight data for instance). We do however note that there does not seem to be a consensus on that, since contributions such as (Halpin et al., 2008) positively evaluate 3D and VR general data visualisations.

As stated in (Drogemuller et al., 2018), there is a trade-off between aspects supporting VR visualisation (increased engagement, added dimension, etc) and specific issues with them (occlusion of information, navigation, etc).

Another way to capitalise on the third dimension offered by immersive experiences is to constrain the graph to a cuboid instead of a simple surface. This allows visualisations to use the upwards axis (that we will call the Z axis) to relay information about the vertices but preserves the mapping with the desktop canvas, since the graph can always be projected back to the surface by ignoring the Z value in vertices' position.

Example of information that can be conveyed by such a technique includes the depth of a vertex. This was implemented as part of the student project (Willième, 2020), previously mentioned in Section 4.3.3. The depth of a vertex is the number of edges needed to reach that vertex, starting from the root vertex in a tree (a DAG where each vertex has exactly one parent, except one vertex, the root vertex, that has no parent). We mapped AD definitions to DAGs that are not necessarily trees, but we can still define depth in that case by choosing a convention. In the aforementioned project, we chose to define the depth of a vertex as the minimum number of edges needed to reach that vertex from any vertex that does not have incoming connections.

We can then compute the depth value for each vertex and map its Z value accordingly (e.g., the deeper the vertex, the higher it is being placed). This particular visualisation if for example useful to identify input parameters easily, since they will be at a depth of 0, even if they might not be positioned on the left of the definition. In general, such a representation helps designers to visually identify the extent to which a certain vertex depends on other vertices. This is in a way similar to automated code formatting, that relies on indentation (horizontal offset) or specific characters to clarify that code blocks can be considered as an entity that is to be executed depending on the instruction generally specified above it.

While the same approach can be replicated with other metrics mapped to the z axis, it is also possible to convey a different type of information. For instance, groups of related components could (automatically or based on user input) be placed at a certain height. This would allow designers to easily visualise related elements. A VR application implementing that idea could allow its users to show or hide certain groups.

4.6.3   Genericity and adaptability

As discussed in Section 4.2, the choice to rely on a graph-based structure to represent Grasshopper definitions in VR means that GHVRGraph could easily be adapted to other AD modelling tools, or even to other domains that rely on models that can be represented as graphs (e.g., Unified Modelling Language diagrams (Object Management Group (OMG), 2017) or Entity Relationship models (Chen, 1976)). Through a simple converter for the target software, it would indeed be possible to reuse the interaction mechanisms we developed.

As mentioned in Section 2.2.1, visual programming has been applied to other domains, such as animation design or for the development of interactive applications. Such domains may be able to benefit from immersive visualisations. This is typically the case when the output produced by the visual program can take advantage of the immersive context. Adapting a solution such as GHVRGraph to these domains therefore allows "visual programmers" to visualise these resulting outputs alongside the generative program itself.

A potential example would be diagrams such as statecharts (Harel, 1987), that define the behaviour of software systems and could therefore be employed to generate applications that execute in a 3D context. In that case, the immersive visualisation of what the application produces could be coupled with editing features for these statecharts, within the same immersive environment. Other potential applications for immersive visual programming are the design of transportation networks, (gaming) scenes, or robotics with

three-dimensional path planning activities.

5

Conclusion

"Without the fun, none of us would go on!"

Ivan Sutherland

The evolution of Computer-Aided Architectural Design was highly influenced by advances in computing technologies, as we have shown in the field's history, covered in Chapter 2. As PCs and computer networks improved and became more affordable, the discipline evolved with them, with new tools and design paradigms appearing along the way. While immersive technologies have been around for more than 50 years, they matured and became available to the masses only recently.

The current use of AR and VR in architecture is mostly limited to visualisation purposes and we have shown that these technologies should not be confined to such purposes. We therefore explored the integration of immersive technologies in the architectural design process, with a focus on AD. While it is too early to reflect on the impact of today's developments on architectural practice, the present dissertation supports the following thesis statement:

> **Thesis statement**
>
> Integrating AR and VR technologies into the Algorithmic Design toolset provides opportunities for architects to improve their workflow and to better present their creations to clients.

This chapter summarises our contributions and shows where they are located in the AD landscape. We also present some practical usage scenarios for the proof-of-concept prototypes we developed and discuss their purpose. Finally, we discuss perspectives we envision for the field of immersive AD and how the research presented in this dissertation could be continued.

5.1. Research contributions

To support the aforementioned thesis statement, we conducted a series of experiments, developed a number of tools, and produced results to answer our research questions.

To verify the claim that immersive technologies should be integrated into the architectural design process, we conducted a survey (Section 2.5.3) on the potential of VR in that context. The survey results support our assumption that the usage of VR early on in the design process is desirable, motivating the need for conducting research on the topic. This section summarises the main contributions our research work brings to the CAAD domain.

We followed a prototype-based process, described in Section 1.4, to provide evidence for our thesis statement. Our principal realisations are three proof-of-concept software prototypes. They all provide immersive experiences based on AD definitions. As indicated by their names (respectively GHVRBridge, GHXR and GHVRGraph), they work with Grasshopper, a popular AD editor letting architects design models with visual algorithms. We presented these prototypes in detail in Chapters 3 and 4 but we summarise hereafter their essence and their purpose.

GHVRBridge, presented in Section 3.2, enables architects to connect their AD definitions in Grasshopper to a VR experience. After choosing which parameters to share with the VR application, the corresponding values can be adjusted from within the immersive environment. The same virtual environment provides a visualisation at scale of the geometry shared by the architect. Changes to the parameter values produce automatic updates on the visualised geometrical representation. GHVRBridge therefore allows architects to simultaneously visualise geometries designed with Grasshopper and adjust the parameters that are used to generate them. This all happens from a single VR application, with no need to leave the VR experience to interact with the model. We explored a collaborative extension to GHVRBridge presented in Section 3.3 to enable a similar experience with multiple stakeholders. It can therefore be used to show a project to a client, immersed in the same virtual environment as the architect, who still has control over parameter values used

for that project. Another possible usage scenario for such an extension is the potentially remote collaboration between architects, to allow them to discuss over a definition in a VR environment, while adjusting parameter values based on their discussion.

In Section 3.4, we presented another prototype, GHXR. It also proposes immersive parameter adjustment and geometry visualisation features, but additionally explores the use of AR and Tangible User Interfaces to do so. GHXR is based on a table display on which tangible items are placed and mapped to Grasshopper parameters. Collaborators placed around the table can therefore naturally interact with these tangible parameters. The system comes with a circular screen setup that displays the geometrical representation, generated using these parameters, from different viewpoints. The visualisation also includes the surrounding buildings in the geometry's future location. At any point in time, users can wear either an AR or a VR headset, to visualise a three-dimensional rendering of the project.

The third developed prototype, GHVRGraph, was presented in Chapter 4. GHVRGraph pushes the control over the AD definition further, by allowing its users to interact with a graph-based representation of a Grasshopper AD definition. We relied on an internal representation based on typed graphs to convert Grasshopper definitions to VR-based interactable models. GHVRGraph allows designers to make deeper changes to definitions than simple parameter adjustments, through a virtual table metaphor that represents Grasshopper components as boxes placed on such a table. We explored different interaction techniques to manipulate these boxes and the links between them, via several variants that have been described in Section 4.3.

5.2. Discussion

This section will first discuss the immersive AD landscape, locating the three developed prototypes in that landscape. Using it, we will clarify the relations between these prototypes as well as how they compare with other immersive AD tools.

On this basis, we will point out some of the limitations of the research we conducted.

Figure 5.1:   Relations between the developed prototypes and recent related work, based on differences in control level and collaboration environment.

5.2.1   Overview of the immersive Algorithmic Design landscape

A first point of comparison for the prototypes we developed is the level of control provided by these systems. This corresponds to how we structured the presentation of the prototypes in the dissertation, with Chapters 3 and 4. The horizontal axis of the landscape in Figure 5.1 corresponds to that criterion of comparison, using the control levels we previously defined (see Table 3.1 on page 72). On the figure's vertical axis, we put the collaboration environment, to differentiate between single-user and collaborative applications, and further subdivide the latter into remote or co-located collaboration.

The landscape shows that commercial tools such as Mindesk and Twinmotion are limited to extrinsic control and can therefore only modify the rendered version of a geometry or its environment. The two prototypes from chapter 3, namely GHVRBride and GHXR, allow users to adjust parameter data, and can therefore have an impact on the input of the algorithm generating the geometrical representation (intrinsic modifications). They, however, do not cover the manipulation of the visual algorithm itself; this being the purpose of GHVRGraph (chapter 4).

Figure 5.1 also includes other software tools, from researchers as well as companies working on similar topics. The next few paragraphs justify their presence in the landscape and discusses how the developed prototypes compare to them.

GHVRBridge's equivalent system described in (Hawton et al., 2018), that was developed at the same time and has been presented in Section 2.5.2, covers the same purpose with a very similar approach. Both systems cover remote collaboration scenarios since multiple users can connect to the same virtual experience using them.

Fologram[1] is the AR equivalent to GHVRBridge, and we thereby put it on the same location as GHXR in the two-dimensional landscape of Figure 5.1, since working in AR means it is naturally adapted for co-located collaboration. We however note that GHXR has additional capabilities (multiple views, VR support, TUI-based interaction, etc.) as compared to Fologram.

While the interaction modalities are different, GHVRGraph is comparable to the work described in (Castelo-Branco et al., 2020) (previsouly discussed in Section 2.5.2) in that they both enable control over the visual algorithm for a given Grasshopper definition. We additionally note that Castelo-Branco et al.'s tool technically allows users to modify parameter values but we did not reflect that fact on Figure 5.1 because it would be particularly hard to achieve a sufficient level of precision with the mirroring technique they proposed.

The coloured elements of Figure 5.1 contain tools that were all published or released after the start of our research work, showing that the belief that immersive technologies need to be integrated into the AD process is shared by many others.

The three prototypes we developed differ from each other in the potential usage scenarios they aim to cover, as shown in Figure 5.2. While these systems all target architects, GHVRBridge and GHXR can also involve clients or other stakeholders, since these prototypes allow them to join shared experiences with the architects and visualise the same geometries. This means that both these prototypes suggest a user-centred approach for AD, where architects preserve control over the AD definition while immersed with these stakeholders.

While GHVRBridge and GHVRGraph can be used for all types of architectural projects, they both provide full-scale visualisations (to take advantage of the immersive aspect of VR) and are likely not adapted for designing at the urban scale. GHXR fills that void through the various viewpoints (including non-VR ones) that are part of the system. These viewpoints allow for such design activities because users can therefore choose the appropriate viewpoint
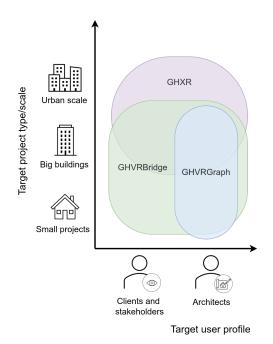
---

[1] `www.fologram.com`

Figure 5.2: Comparison between the three developed prototypes, with regards to target user profiles and types of architectural projects being covered.

when needed. However, the hardware and space requirements of GHXR imply that it only makes sense to rely on such a system for projects that are big enough to justify investing in its usage. It also follows that GHXR targets bigger companies than GHVRBridge and GHVRGraph, which are accessible to any type of company or individual.

To make the previous claims more concrete, we recall here (and assign to a particular system) the 3 hypothetical use cases we introduced in Chapters 3 and 4, since they help clarify some usage scenarios where the developed prototypes are likely to make an appreciable difference.

Use Case for GHVRGraph. A small architectural firm is in charge of the construction of a monument for a big garden (10,000 square meters) at the entrance of a small castle converted to a modern art gallery. Commissioned by the gallery's board, the monument should inspire a sense of prestige. The architects are still in an early design phase with multiple options being considered. Based on the current state of the AD definition she created with Grasshopper, one of these architects wants to get a better idea of what these options look like from a human perspective. She therefore starts GHVRGraph with the intent of checking the options in VR, switching between options by editing links in the graph-based VR representation. Once immersed and after checking the initial options, the architect has a new idea: she wants to combine two of the initially proposed options. After rebranching the definition from the VR-based representation, she visualises the result. She then moves back to Grasshopper to refine the design idea based on the version she modified from GHVRGraph.

Use Case for GHVRBridge. An architectural firm is in charge of the construction of a modern-looking clubhouse for a golf club. Based on the club's demands, the architects have designed a first version of the building that will welcome the 300 registered members. The design process in quite advanced and the building is expected to be close to its final form. The architects would like to get input from the clients (the club's board members), so they invite them to a VR session where both the lead architect and board members visualise the virtual building in its current state. The VR visualisation helps the clients understand what the actual clubhouse will look like. The architect guides the clients around the virtual environment and adjusts parameter values along the way to accommodate the remarks from the clients.

Use Case for GHXR. The municipal council of a city wants to create a public space in a disused area of 1,000 square meters, nestled amid a few apartments and shopping buildings. They appointed an architectural firm that works with Algorithmic Design to do so and asked them to include a big sculpture to

be placed somewhere in the centre of that space. The project is already well advanced in Grasshopper and the architects now want the opinion of the council to adjust parameter values. They therefore invite council members for a meeting at their office, that themselves invite representatives from commercial and residential buildings in the project's vicinity with them. Using GHXR, all these stakeholders are able to visualise the project from different angles and can take an active part in the remaining (parameter adjustment) design decisions. The AR and VR visualisations help non-architects to better appreciate the dimensions of the sculpture and how it integrates with the surroundings.

5.2.2   Limitations

The three proof-of-concept prototypes that we developed extend the integration of immersive technologies beyond their current established usage in the field of CAAD, but it is always possible to push the boundaries further. This section will discuss limitations related to the developed prototypes, pointing out areas where they could improve. We will also cover aspects and challenges that were considered out of scope of the present dissertation, but that would need to be tackled for the potential commercialisation of similar software tools. Finally, we will discuss limitations that pertain to the methodology we followed.

Limitations related to interfaces and proposed interactions

The first series of limitations are related to the user interface and the interaction modalities we proposed in the three developed prototypes. GHVRBridge is the first prototype we developed. As presented in Section 3.2, the type of interface we used is similar to those of most desktop tools, with a 6-DoF controller replacing the usual mouse or trackpad. Due to the three-dimensional context, it is desirable to go beyond such two-dimensional interfaces. This partly motivated the development of the next two prototypes: GHXR and GHVRGraph.

   GHVRBridge could also be extended by integrating smarter interactions for certain parameters. It would for example be possible to provide control over a point, surface, or volume, directly from within the VR-based environment, using three-dimensional widgets similar to those used in standard 3D modellers. The use of such widgets would replace the manipulation of separate sliders (e.g., a single handle to control a point would replace three number sliders).

   In a similar way, going further with interactions would also have been feasible for GHXR. We in fact did not enable control over the perspective

camera, as explained in Section 3.4. It is difficult to provide control over a 3D element using widgets whose position is only tracked in two dimensions (along the tangible table's surface). However, during the last few days of my research stay at LIST, we started to develop an extended version of a tangible, that is mounted with an infrared sensor able to evaluate the distance between the sensor and an object placed above it. Using that sensor, a user would be able to move his hand up and down to control an additional dimension (e.g., the height at which the camera is placed), thereby enabling three-dimensional control. This is left as a future work that would extend the capabilities of GHXR.

As for GHVRGraph, the feature set could be expanded by supporting additional actions on the graph-based representation. This includes moving and zooming on the canvas (the table surface where the graph representation is placed). The user would then be able to choose how much of the representation needs to be visualised at any given time. More advanced interaction techniques (such as the ones presented in Section 2.4) could also be employed, to take advantage of different approaches and mitigate their individual shortcomings.

It would also be interesting to integrate the parameter adjustment feature from GHVRBridge and GHXR into GHVRGraph, to provide a complete experience to the architect, who would thereby gain access to multiple control levels for different design activities.

Additional untackled aspects

The scope of this dissertation focused on specific aspects, inevitably leading to other aspects that could not be covered. As mentioned in Chapters 3 and 4, GHVRBridge and GHVRGraph are based on the assumption that the number of users, collaborating in a concurrent session with these prototypes, is limited. A commercial product based on our work would likely need to tackle the concurrent modification issue (and the conflict resolution) in a more advanced way than the one we implemented. The proposals inspired from different fields that we discussed in the corresponding chapters could be a good start to reflect on how to do so.

Other elements that would need to be perfected in order to bring the developed prototypes to a broader audience include the general look and feel of the prototypes. Better-looking VR menus and interactable objects would need to be created. The representation of the surrounding buildings we show in GHXR could also be improved and a commercial solution may want to rely on another (paid) provider to obtain more accurate or up-to-date building data

than the one we used.

Another important feature that would need to be implemented for a commercial usage of such tools is audio communication between distant collaborators (often referred to as Voice over IP (Davidson et al., 2000)). Many solutions exist to achieve such communication on game engines such as Unity, including paid out-of-the-box options that would be easy to integrate with our code.

Another limitation we mention here because it corresponds to some of the feedback we received is the potential delay between a parameter modification in GHXR or GHVRBridge, and the update of the corresponding geometrical representation rendered in the virtual environment. GHVRBridge was conceived with particular attention to network aspects, with adapted transfer protocols and optimised packet sizes. On the other hand, GHXR was designed to be run from a local setup, and should therefore typically benefit from very high speed data transfers between components of the system that do not require the same kind of optimisations as GHVRBridge.

Regardless of the level of attention given to data transfer, the main bottleneck remains the generation of the geometry by the AD software itself, at least for complex definitions. It is not uncommon to encounter complicated designs in which modifying a parameter value leads to multiple seconds of re-generation time. The added delay introduced by our solutions is typically negligible for such models. There is unfortunately no real solution to that issue, since it pertains to the optimisation of the AD software itself and because some designs are simply too complex to be computed rapidly. GHVRBridge and GHXR propose a rapid VR-based evaluation of design variations, and architects should therefore focus their updates on parameters whose adjustment does not yield a long re-generation of the geometrical representation. More complex designs that require lengthy computation times are simply not adapted to a rapid iteration process (VR-based or not).

Methodological limitations

Each of the developed prototypes was subject to evaluations with architectural students and researchers, through workshops and demo sessions that we organised during course sessions, seminars and other events. However, most of these evaluations were rather informal and limited in the number of participants by the nature of the events. This means that the evaluations we conducted were subject to threats related to the small population size (less diversity, more susceptible to outliers, etc).

Furthermore, we did not validate the prototypes with professional practitioners, so the feedback we received was biased towards a student or academic

point of view. As pointed out in (Stals & Caldas, 2020), while it is common to find studies that do not involve professional architects as part of their testing sessions, doing so would help in making sure that the developed tools answer their needs.

The lack of a more systematic validation of the developed prototypes with more professional participants was initially due to a shortage of contacts. Once we had identified and solicited enough potential participants that indicated being interested in evaluating these prototypes, we had difficulty to actually perform these evaluations, due to the sanitary situation (linked to the COVID-19 crisis) that affected the last two years of our research. We nevertheless created evaluation protocols for the developed prototypes and it would be possible to conduct such evaluations rapidly, should we have access to a pool of qualified participants and the possibility to perform on-site tests. Remote validation is hardly possible because of both hardware and software requirements.

While it is difficult to transpose the evaluation of immersive systems to an online setting, since they involve VR or AR components, we did so for GHXR. As mentioned in Section 3.5, we relied on a simulator for the display table and its tangible widgets. We in fact conducted three testing sessions where the participant had to perform a particular task using that simulator before answering a post-task questionnaire. In addition to the limited number of participants, it would be difficult to assert the irrefutable nature of such online evaluations, since the user experience is then transferred to a two-dimensional environment and the user must interact with software objects that simulate the actual tangible items of the immersive system.

5.3. Perspectives

The developed prototypes demonstrate the potential of using immersive technologies for AD activities, and can be used as a stepping stone for discussion over the topic, so that architects can engage with these technologies. They can be seen as a precursor for the development of fully-fledged commercial products that should improve the design process for many architects. That impact should however not be limited to the architects, since the use of such technologies should lead to clients and other stakeholders being more involved during the design process, shifting it to a user-centred process. Independently from the previous discussion on necessary refinements to the developed prototypes to prepare them for broader and commercial usage, this section discusses future opportunities for AD and its use of immersive technologies.

5.3.1   Enhancing the aid for Algorithmic Design

As stated in Section 4.6.2, there are opportunities to use the third dimension offered by immersive technologies to convey additional information to the designer working on an AD definition. Various representations should be explored. The usage of height variation to carry additional information appears to be the most promising since it preserves a natural mapping with the standard representation of AD models in desktop-based editors such as Grasshopper. Experiments should be run with different metrics used as the conveyed information, including indicators inspired by the software engineering literature (Fenton & Neil, 2000) (e.g., to evaluate the complexity of a definition, using metrics such as cyclomatic complexity (Ebert et al., 2016)). Such work on quality analysis of AD definitions has in fact already begun (Davis, 2013) but has yet to reach actual tools in use by practitioners.

It would also be interesting to try such alternative visualisations within desktop-based AD tools as well. The user should in that case be able to switch from these extended visualisations to the basic representation easily, depending on that user's needs. More generally, the aids offered by most textual-based development environments (e.g., syntax colouring or code autocompletion) could be adapted to the visual AD editors. In fact, desktop AD editors typically include an autocompletion feature that suggests components based on what the user starts to type. This could be extended to provide suggestions based on the context (e.g., what component was placed or clicked last) and suggestions could also appear when one end of a link has been selected (only suggesting components that contain an input port with an appropriate type for the given link).

In addition to such aids, software development tools often include automated refactoring features (Mens & Tourwé, 2004), allowing the developer to restructure the code, usually to improve on the design, while preserving its behaviour. Refactorings sometimes occur to introduce software design patterns (Gamma et al., 1995), that are generic solutions to solve regular issues in the development of software systems. Interestingly, the concept of patterns as reusable proven solutions to common problems originated from the architectural design field (Alexander, 1977), where they serve as an aid to designers. We believe it is time for the pattern concept to return to architecture and AD in particular, as suggested in (Woodbury et al., 2010), a seminal book on AD where the author defined AD patterns inspired by the software development equivalents. Based on these patterns, AD tools could propose refactorings to their users, so as to improve the overall quality (e.g., in terms of readability, maintainability or reusability) of the designed definitions.

Returning to the topic of immersive support for AD activities, the way we developed the prototypes makes their interoperability with other AD software possible, through the development of appropriate converters between such software and our representations. A combined solution blending the developed prototypes, and extending them to support other AD software than Grasshopper, would have the potential to become a single immersive platform for AD, providing a uniform experience to practitioners using different software tools.

5.3.2 Technological improvements

Until now, portable standalone headsets for both AR and VR, that do not need to be connected to a computer to function, lack computing power. This impacted the development of GHXR, since the AR headset that we used (the Hololens) was not powerful enough to display the surrounding buildings that our VR-based component employs (that component uses the HTC Vive, a headset linked to a computer that performs the computations). While it appears clear that relying on a bigger computer to perform heavy computations will lead to better performances for connected headsets, an increase in computing power for portable headsets would allow for additional compute-intensive features. This would in our case allow to add holograms of surrounding buildings to the AR component of GHXR.

Another possible direction to extend the use of AR for architectural design would be to rely on other kinds of augmentations than visual ones. Olfactory AR could be employed to further immerse users through smells and audio AR would allow for sound simulations to be run, so that users would get an idea of expected noise disturbances linked to a project.

A potential future (r)evolution for AR and VR displays is the advent of Retinal Projection Displays (RPDs) (Pryor et al., 1998), that project content directly onto the wearer's retina. Once such technology matures, it has the potential to lead to immersive "glasses" in a small form factor, that would become wearable for significantly longer periods than the current headsets. These glasses could also potentially turn into devices capable of switching between AR and VR modes, and would therefore be an ideal multi-purpose immersive display. Architects would easily get access to the types of visualisations we developed, with nothing but such glasses.

We also think and hope that immersive technologies will keep on becoming more affordable, so that they can reach architects from firms of all sizes, and thereby become integrated into architectural practice.

Appendices

GHXR evaluation protocol

Removed because arxiv does not like pdfs

GHXR System Usability Scale questionnaire

Removed because arxiv does not like pdfs

Bibliography

Abdelmohsen, S., & Do, E. Y.-L. (2007). Tangicad: Tangible interface for manipulating architectural 3d models.

Agirbas, A. (2020). Algorithmic decomposition of geometric islamic patterns: A case study with star polygon design in the tombstones of ahlat. *Nexus Network Journal*, *22*(1), 113–137.

Aho, A. V., Sethi, R., & Ullman, J. D. (1986). Compilers, principles, techniques. *Addison wesley*, *7*(8), 9.

Aish, R., & Bredella, N. (2017). The evolution of architectural computing: From Building Modelling to Design Computation. *Arq: Architectural Research Quarterly*, *21*(1), 65–73.

Alexander, C. (1977). *A pattern language: towns, buildings, construction*. Oxford university press.

Alfaiate, P., Caetano, I., et al. (2017). Luna moth: supporting creativity in the cloud. In *Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*.

Aoki, H., Oman, C. M., & Natapoff, A. (2007). Virtual-reality-based 3d navigation training for emergency egress from spacecraft. *Aviation, space, and environmental medicine*, *78*(8), 774–783.

Apellániz, D., Pasanen, P., & Gengnagel, C. (2021). A holistic and parametric approach for life cycle assessment in the early design stages.

Appleby, D. (1991). *Programming languages: paradigm and practice*. McGraw-Hill, Inc.

Arora, R., Kazi, R. H., Anderson, F., Grossman, T., Singh, K., & Fitzmaurice, G. (2017). Experimental Evaluation of Sketching on Surfaces in VR. (pp. 5643–5654). ACM Press.

As, I., Pal, S., & Basu, P. (2018). Artificial intelligence in architecture: Generating conceptual design via deep learning. *International Journal of Architectural Computing*, *16*(4), 306–327.

Asanowicz, A. (1999). Evolution of computer aided design: Three generations of CAD.

Azhar, S., Khalfan, M., & Maqsood, T. (2012). Building information modelling (BIM): Now and beyond. *Construction Economics and Building*, *12*(4), 15–28.

Baba, Y., & Nobeoka, K. (1998). Towards knowledge-based product development: The 3-D CAD model of knowledge creation. *Research policy*, *26*(6), 643–659.

Baker, B. S., & Coffman Jr, E. G. (1996). Mutual exclusion scheduling. *Theoretical Computer Science*, *162*(2), 225–243.

Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, *24*(6), 574–594.

Basdogan, C., Sedef, M., Harders, M., & Wesarg, S. (2007). Vr-based simulators for training in minimally invasive surgery. *IEEE Computer Graphics and Applications*, *27*(2), 54–66.

Ben-Kiki, O., Evans, C., & Ingerson, B. (2009). YAML ain't markup language (YAML) (tm) version 1.2. Tech. rep., YAML.org.
URL `http://www.yaml.org/spec/1.2/spec.html`

Benbelkacem, S., Zenati-Henda, N., Zerarga, F., Bellarbi, A., Belhocine, M., Malek, S., & Tadjine, M. (2011). Augmented reality platform for collaborative e-maintenance systems. *Augmented Reality-Some Emerging Application Areas, InTech*, (pp. 211–226).

Besserud, K., & Cotten, J. (2008). Architectural genomics.

Bezier, P. (1977). *Essai de définition numérique des courbes et des surfaces expérimentales*. Ph.D. thesis, Université Pierre-et-Marie-Curie, Paris.

Bibb, R., Eggbeer, D., & Paterson, A. (2014). *Medical modelling: the application of advanced design and rapid prototyping techniques in medicine*. Woodhead Publishing.

Billinghurst, M., Baldis, S., Matheson, L., & Philips, M. (1997). *3D Palette: A Virtual Reality Content Creation Tool*.

Billinghurst, M., Kato, H., & Poupyrev, I. (2001). The magicbook-moving seamlessly between reality and virtuality. *IEEE Computer Graphics and applications*, *21*(3), 6–8.

Bimber, O., & Raskar, R. (2006). Modern approaches to augmented reality. In *ACM SIGGRAPH 2006 Courses*, (pp. 1–es).

Blach, R., Landauer, J., Rösch, A., & Simon, A. (1998). A flexible prototyping tool for 3d real-time user-interaction. In *Virtual Environments' 98*, (pp. 195–203). Springer.

Blum, T., Kleeberger, V., Bichlmeier, C., & Navab, N. (2012). mirracle: An augmented reality magic mirror system for anatomy education. In *2012 IEEE Virtual Reality Workshops (VRW)*, (pp. 115–116). IEEE.

Bolt, R. A. (1980). *"Put-That-There": Voice and Gesture at the Graphics Interface*, vol. 14. ACM.

Borrego, A., Latorre, J., Llorens, R., Alcañiz, M., & Noé, E. (2016). Feasibility of a walking virtual reality system for rehabilitation: objective and subjective parameters. *Journal of neuroengineering and rehabilitation*, *13*(1), 1–10.

Bose, S., Vahabzadeh, S., & Bandyopadhyay, A. (2013). Bone tissue engineering using 3d printing. *Materials today*, *16*(12), 496–504.

Botella, C., Osma, J., Garcia-Palacios, A., Quero, S., & Baños, R. (2004). Treatment of flying phobia using virtual reality: data from a 1-year followup using a multiple baseline design. *Clinical Psychology & Psychotherapy: An International Journal of Theory & Practice*, *11*(5), 311–323.

Botella, C. M., Juan, M. C., Baños, R. M., Alcañiz, M., Guillén, V., & Rey, B. (2005). Mixing realities? an application of augmented reality for the treatment of cockroach phobia. *Cyberpsychology & behavior*, *8*(2), 162–171.

Bottecchia, S., Cieutat, J.-M., & Jessel, J.-P. (2010). Tac: augmented reality system for collaborative tele-assistance in the field of maintenance through

internet. In *Proceedings of the 1st Augmented Human International Conference*, (pp. 1–7).

Bowman, D. A., & Hodges, L. F. (1997). An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, (pp. 35–ff).

Brade, J., Lorenz, M., Busch, M., Hammer, N., Tscheligi, M., & Klimant, P. (2017). Being there again–presence in real and virtual environments and its relation to usability and user experience using a mobile navigation task. *International Journal of Human-Computer Studies*, *101*, 76–87.

Bradner, E., Iorio, F., Davis, M., et al. (2014). Parameters tell the design story: ideation and abstraction in design optimization. In *Proceedings of the symposium on simulation for architecture & urban design*, vol. 26.

Braid, I. C. (1975). The synthesis of solids bounded by many faces. *Communications of the ACM*, *18*(4), 209–216.

Brooke, J. (1996). Sus: a "quick and dirty'usability. *Usability evaluation in industry*, *189*(3).

Brooks Jr, F. P. (1987). Walkthrough—a dynamic graphics system for simulating virtual buildings. In *Proceedings of the 1986 workshop on Interactive 3D graphics*, (pp. 9–21).

Brown, D. G., Coyne, J. T., & Stripling, R. (2006). Augmented reality for urban skills training. In *IEEE Virtual Reality Conference (VR 2006)*, (pp. 249–252). IEEE.

Brunet, G., Chechik, M., Easterbrook, S., Nejati, S., Niu, N., & Sabetzadeh, M. (2006). A manifesto for model merging. In *Proceedings of the 2006 international workshop on Global integrated model management*, (pp. 5–12).

Buffa, E. S. (1964). Allocating facilities with craft. *Harvard business review*, *42*(2), 136–159.

Bullinger, H.-J., Bauer, W., Wenzel, G., & Blach, R. (2010). Towards user centred design (UCD) in architecture based on immersive virtual environments. *Computers in Industry*, *61*(4), 372–379.

Burry, M. (2011). *Scripting cultures: Architectural design and programming*. John Wiley & Sons.

Butterworth, J. (1992). 3DM: A three-dimensional modeler using a head-mounted display.

Caetano, I., Santos, L., & Leitão, A. (2020). Computational design in architecture: Defining parametric, generative, and algorithmic design. *Frontiers of Architectural Research*.

Caldas, L. G., & Norford, L. K. (2003). Genetic algorithms for optimization of building envelopes and the design and control of hvac systems. *J. Sol. Energy Eng.*, *125*(3), 343–351.

Calixto, V., & Celani, G. (2015). A literature review for space planning optimization using an evolutionary algorithm approach: 1992-2014.

Carlin, A. S., Hoffman, H. G., & Weghorst, S. (1997). Virtual reality and tactile augmentation in the treatment of spider phobia: a case report. *Behaviour research and therapy*, *35*(2), 153–158.

Carrino, F., Rizzotti, D., Gheorghe, C., Bakajika, P. K., Francescotti-Paquier, F., & Mugellini, E. (2014). Augmented reality treatment for phantom limb pain. In *International Conference on Virtual, Augmented and Mixed Reality*, (pp. 248–257). Springer.

Castelo-Branco, R., Brás, C., & Leitão, A. M. (2020). Inside the matrix: Immersive live coding for architectural design. *International Journal of Architectural Computing*.

Chaillou, S. (2019). *AI + architecture - Towards a New Approach*. Ph.D. thesis, Harvard University.

Chaplot, D. S., Gandhi, D., Gupta, S., Gupta, A., & Salakhutdinov, R. (2020). Learning to explore using active neural slam. *arXiv preprint arXiv:2004.05155*.

Chen, H., Ooka, R., & Kato, S. (2008). Study on optimum design method for pleasant outdoor thermal environment using genetic algorithms (ga) and coupled simulation of convection, radiation and conduction. *Building and Environment*, *43*(1), 18–30.

Chen, P. P.-S. (1976). The entity-relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)*, *1*(1), 9–36.

Cho, I., & Wartell, Z. (2015). Evaluation of a bimanual simultaneous 7dof

interaction technique in virtual environments. In *2015 IEEE symposium on 3D User Interfaces (3DUI)*, (pp. 133–136). IEEE.

Choi, J. P., Lee, T. Y., Ahn, E. S., Piao, G. S., & Lim, J. H. (2014). Evaluation of parameters for louver design algorithm based on direct solar radiation control performance. In *Advanced Materials Research*, vol. 838, (pp. 1917–1922). Trans Tech Publ.

Chu, C.-C. P., Dani, T. H., & Gadh, R. (1997). Multi-sensory user interface for a virtual-reality-based computeraided design system. *Computer-Aided Design*, *29*(10), 709–725.

Church, A. (2016). *The Calculi of Lambda Conversion.(AM-6), Volume 6*. Princeton University Press.

Cichocka, J. M., Browne, W. N., & Rodriguez, E. (2017). Optimization in the architectural practice. In *CAADRIA 2017-22nd International Conference on Computer-Aided Architectural Design Research in Asia: Protocols, Flows and Glitches*, (pp. 387–396). The Association for Computer-Aided Architectural Design Research in Asia ....

Coates, P., & Thum, R. (1995). Generative modelling. *London: University of East London*, (p. 2).

Conn, C., Lanier, J., Minsky, M., Fisher, S., & Druin, A. (1989). Virtual environments and interactivity: Windows to the future. In *ACM SIGGRAPH 89 Panel Proceedings*, (pp. 7–18).

Coons, S. A. (1963). An outline of the requirements for a computer-aided design system. In *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*, (pp. 299–304).

Coppens, A. (2017). *Merging Real and Virtual Worlds: An Analysis of the State of the Art and Practical Evaluation of Microsoft Hololens*. Master's thesis, University of Mons.

Coppens, A., Bicer, B., Yilmaz, N., & Aras, S. (2019). Exploration of interaction techniques for graph-based modelling in virtual reality. In *eNTER-FACE'19*. Ankara, Turkey.

Coppens, A., & Mens, T. (2018). Towards Collaborative Immersive Environments for Parametric Modelling. In *International Conference on Cooperative Design, Visualization and Engineering*. Springer.

Coppens, A., Mens, T., & Gallas, M.-A. (2018). Parametric Modelling Within Immersive Environments: Building a Bridge Between Existing Tools and Virtual Reality Headsets. In *36th eCAADe Conference*.

Coppens, A., Mens, T., & Gallas, M.-A. (2021). Integrating virtual reality during the architectural design process: a survey to identify practitioner needs. In *Proc. of the Conference CIB W78*, vol. 2021, (pp. 11–15).

Coroado, L., Pedro, T., D 'alpuim, J., Eloy, S., & Dias, M. (2015). *VIAR-MODES: Visualization and Interaction in Immersive Virtual Reality for Architectural Design Process*.

Cristie, V., & Joyce, S. C. (2019). 'ghshot': a collaborative and distributed visual version control for grasshopper parametric programming. In *Proceedings of the 37th eCAADe Conference*, vol. 3, (pp. 35–44).

Cruz-Neira, C., Sandin, D. J., DeFanti, T. A., Kenyon, R. V., & Hart, J. C. (1992). The CAVE: Audio Visual Experience Automatic Virtual Environment. *Commun. ACM*, *35*(6), 64–72.

Cunningham, J., & Dixon, J. (1988). Designing with features: the origin of features. In *Proceedings of the 1988 ASME International Computers in Engineering Conference and Exhibition*, vol. 1, (pp. 237–243). ASME New York.

Dagit, C. E. (1993). Establishing virtual design environments in architectural practice.

Dave, D., Chowriappa, A., & Kesavadas, T. (2013). Gesture interface for 3d cad modeling using kinect. *Computer-Aided Design and Applications*, *10*(4), 663–669.

Davidowitz, G., & Kotick, P. G. (2011). The use of cad/cam in dentistry. *Dental Clinics*, *55*(3), 559–570.

Davidson, J., Peters, J., Peters, J., & Gracely, B. (2000). *Voice over IP fundamentals*. Cisco press.

Davis, D. (2013). *Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture*. PhD thesis, RMIT University.

Dawood, A., Marti, B. M., Sauret-Jackson, V., & Darwood, A. (2015). 3d printing in dentistry. *British dental journal*, *219*(11), 521–529.

Delgado, J. M. D., Oyedele, L., Demian, P., & Beach, T. (2020). A research agenda for augmented and virtual reality in architecture, engineering and construction. *Advanced Engineering Informatics*, *45*, 101122.

Doboš, J., & Steed, A. (2012). 3d diff: an interactive approach to mesh differencing and conflict resolution. In *SIGGRAPH Asia 2012 Technical Briefs*, (pp. 1–4).

Donath, D., & Regenbrecht, H. (1995). Vrad (virtual reality aided design) in the early phases of the architectural design process.

Doraiswamy, H., Ferreira, N., Lage, M., Vo, H., Wilson, L., Werner, H., Park, M., & Silva, C. (2015). Topology-based catalogue exploration framework for identifying view-enhanced tower designs. *ACM Transactions on Graphics (TOG)*, *34*(6), 1–13.

Dorta, T., Kinayoglu, G., & Hoffmann, M. (2016). Hyve-3D and the 3D Cursor: Architectural co-design with freedom in Virtual Reality. *International Journal of Architectural Computing*, *14*(2), 87–102.

Drogemuller, A., Cunningham, A., Walsh, J., Cordeil, M., Ross, W., & Thomas, B. (2018). Evaluating navigation techniques for 3d graph visualizations in virtual reality. In *2018 International Symposium on Big Data Visual and Immersive Analytics (BDVA)*, (pp. 1–10). IEEE.

Dubois, E., Nigay, L., & Troccaz, J. (2000). Combinons le monde virtuel et le monde réel: classification et principes de conception. *Actes des Rencontres Jeunes Chercheurs en Interaction Homme-Machine*, (pp. 31–34).

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, *13*(2), 99–110.

Eaglesham, G. (1979). CAD: State of the art: 16 september 1978, london. Organized by TASS (technical, administrative and supervisory section) of the amalgamated union of engineering workers. *Computer-Aided Design*, *11*(2), 85–91.

Eastman, C. M. (1975). The use of computers instead of drawings in building design. *AIA Journal*, *63*(3), 46–50.

Ebert, C., Cain, J., Antoniol, G., Counsell, S., & Laplante, P. (2016). Cyclomatic complexity. *IEEE software*, *33*(6), 27–29.

Echtler, F., Sturm, F., Kindermann, K., Klinker, G., Stilla, J., Trilk, J., & Najafi, H. (2004). The intelligent welding gun: Augmented reality for experimental vehicle construction. In *Virtual and augmented reality applications in manufacturing*, (pp. 333–360). Springer.

Ehrig, H., Prange, U., & Taentzer, G. (2004). Fundamental theory for typed attributed graph transformation. In *International conference on graph transformation*, (pp. 161–177). Springer.

Elmqvist, N. (2017). 3d visualization for nonspatial data: Guidelines and challenges.
URL https://sites.umiacs.umd.edu/elm/2017/10/03/3d-visualization-for-nonspatial-data-guidelines-and-challenges/

Ens, B., Anderson, F., Grossman, T., Annett, M., Irani, P., & Fitzmaurice, G. (2017). Ivy: Exploring spatially situated visual programming for authoring and understanding intelligent environments. In *Proceedings of the 43rd Graphics Interface Conference*, (pp. 156–162). Canadian Human-Computer Communications Society.

Ercan, B., & Elias-Ozkan, S. T. (2015). Performance-based parametric design explorations: A method for generating appropriate building components. *Design Studies*, *38*, 33–53.

Farin, G. (2002). A history of curves and surfaces. *Handbook of computer aided geometric design*, *1*.

Feiner, A. O. S. (2003). The flexible pointer: An interaction technique for selection in augmented and virtual reality. In *Proc. UIST'03*, (pp. 81–82).

Fenton, N. E., & Neil, M. (2000). Software metrics: roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, (pp. 357–370).

Fette, I., & Melnikov, A. (2011). The websocket protocol.

Fleck, S., & Simon, G. (2013). An augmented reality environment for astronomy learning in elementary grades: an exploratory study. In *Proceedings of the 25th Conference on l'Interaction Homme-Machine*, (pp. 14–22).

Forsberg, A., Herndon, K., & Zeleznik, R. (1996). Aperture based selection for immersive virtual environments. In *ACM Symposium on User Interface Software and Technology*, (pp. 95–96). Citeseer.

Frees, S., & Kessler, G. D. (2005). Precise and rapid interaction through scaled manipulation in immersive virtual environments. In *IEEE Proceedings. VR 2005. Virtual Reality, 2005.*, (pp. 99–106). IEEE.

Fritz, F., Susperregui, A., & Linaza, M. T. (2005). Enhancing cultural tourism experiences with augmented reality technologies. 6th International Symposium on Virtual Reality, Archaeology and Cultural . . . .

Fuchs, H., Livingston, M. A., Raskar, R., Keller, K., Crawford, J. R., Rademacher, P., Drake, S. H., Meyer, A. A., et al. (1998). Augmented reality visualization for laparoscopic surgery. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, (pp. 934–943). Springer.

Fuchs, P. (2006). *Le traité de la réalité virtuelle*, vol. 2. Presses des MINES.

Fusté Lleixà, A. (2018). *Hypercubes: Learning Computational Thinking through Embodied Spatial Programming in Augmented Reality*. Ph.D. thesis, Massachusetts Institute of Technology.

Gaitatzes, A., Christopoulos, D., & Roussou, M. (2001). Reviving the past: cultural heritage meets virtual reality. In *Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage*, (pp. 103–110).

Gamma, E., Helm, R., Johnson, R., Vlissides, J., & Patterns, D. (1995). *Elements of reusable object-oriented software*, vol. 99. Addison-Wesley Reading, Massachusetts.

Gardiner, J. (2011). *Exploring the emerging design territory of construction 3D printing-project led architectural research*. Ph.D. thesis, RMIT University.

Garza, L. E., Pantoja, G., Ramírez, P., Ramírez, H., Rodríguez, N., González, E., Quintal, R., & Pérez, J. A. (2013). Augmented reality application for the maintenance of a flapper valve of a fuller-kynion type m pump. *Procedia Computer Science*, *25*, 154–160.

Gavish, N., Gutiérrez, T., Webel, S., Rodríguez, J., Peveri, M., Bockholt, U., & Tecchia, F. (2015). Evaluating virtual reality and augmented reality training for industrial maintenance and assembly tasks. *Interactive Learning Environments*, *23*(6), 778–798.

Geiger, C., Mueller, W., & Rosenbach, W. (1998). Sam-an animated 3d programming language. In *Proceedings. 1998 IEEE Symposium on Visual Languages (Cat. No. 98TB100254)*, (pp. 228–235). IEEE.

Ghinea, M., Frunză, D., Chardonnet, J.-R., Merienne, F., & Kemeny, A. (2018). Perception of absolute distances within different visualization systems: Hmd and cave. In *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*, (pp. 148–161). Springer.

Goetschalckx, M. (1992). An interactive layout heuristic based on hexagonal adjacency graphs. *European Journal of Operational Research*, *63*(2), 304–321.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, *27*.

Greenhalgh, S. (2016). The effects of 3d printing in design thinking and design education. *Journal of Engineering, Design and Technology*.

Grubert, J., Witzani, L., Ofek, E., Pahud, M., Kranz, M., & Kristensson, P. O. (2018). Text entry in immersive head-mounted display-based virtual reality using standard keyboards. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, (pp. 159–166). IEEE.

Gruenewald, T., & Witteborn, S. (2020). Feeling good: Humanitarian virtual reality film, emotional style and global citizenship. *Cultural Studies*, (pp. 1–21).

Gu, N., Kim, M. J., & Maher, M. L. (2011). Technological advancements in synchronous collaboration: The effect of 3d virtual worlds and tangible user interfaces on architectural design. *Automation in Construction*, *20*(3), 270–278.

Halpin, H., Zielinski, D. J., Brady, R., & Kelly, G. (2008). Exploring semantic social networks using virtual reality. In *International Semantic Web Conference*, (pp. 599–614). Springer.

Hanan, H., Suwardhi, D., Nurhasanah, T., & Santa Bukit, E. (2015). Batak toba cultural heritage and close-range photogrammetry. *Procedia-Social and Behavioral Sciences*, *184*, 187–195.

Harding, J. E., & Shepherd, P. (2017). Meta-parametric design. *Design Studies*, *52*, 73–95.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of computer programming*, *8*(3), 231–274.

Hawton, D., Cooper-Wooley, B., Odolphi, J., Doherty, B., Fabbri, A., Gardner, N., & Haeusler, M. H. (2018). Shared Immersive Environments For Parametric Model Manipulation. In *Learning, Adapting and Prototyping*, vol. 1. Hong Kong.

Henrysson, A., Billinghurst, M., & Ollila, M. (2005). Face to face collaborative ar on mobile phones. In *Fourth ieee and acm international symposium on mixed and augmented reality (ismar'05)*, (pp. 80–89). IEEE.

Hepworth, A., Tew, K., Trent, M., Ricks, D., Jensen, C. G., & Red, W. E. (2014). Model consistency and conflict resolution with data preservation in multi-user computer aided design. *Journal of Computing and Information Science in Engineering*, *14*(2).

Heun, V., Hobin, J., & Maes, P. (2013). Reality editor: programming smarter objects. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, (pp. 307–310).

Hillyard, R., & Braid, I. (1978). Analysis of dimensions and tolerances in computer-aided mechanical design. *Computer-Aided Design*, *10*(3), 161–166.

Hoffmann, C. M. (1989). Geometric and solid modeling.

Hoffmann, C. M. (2005). Constraint-based computer-aided design.

Horváth, I., & Vroom, R. W. (2015). Ubiquitous computer aided design: A broken promise or a Sleeping Beauty? *Computer-Aided Design*, *59*, 161–175.

Howard, T. J., Culley, S. J., & Dekoninck, E. (2008). Describing the creative design process by the integration of engineering design and cognitive psychology literature. *Design studies*, *29*(2), 160–180.

Huang, W., & Zheng, H. (2018). Architectural drawings recognition and generation through machine learning.

Hugues, O., Fuchs, P., & Nannipieri, O. (2011). New augmented reality taxonomy: Technologies and features of augmented environment. In *Handbook of augmented reality*, (pp. 47–63). Springer.

Innes, D., Moleta, T., & Schnabel, M. (2017). 'Virtual inhabitation and creation: A comparative study of interactive 1: 1 modelling as a design method'. In *Conference: DADA 2017 International Conference on Digital Architecture:"Digital Culture*.

ISO/IEC-21778:2017 (2017). Information technology – The JSON data interchange syntax. Standard, International Organization for Standardization, Geneva, CH.

Jack, D., Boian, R., Merians, A. S., Tremaine, M., Burdea, G. C., Adamovich, S. V., Recce, M., & Poizner, H. (2001). Virtual reality-enhanced stroke rehabilitation. *IEEE transactions on neural systems and rehabilitation engineering*, *9*(3), 308–318.

Janssen, P., Li, R., & Mohanty, A. (2016). Mobius: a parametric modeller for the web. In *Proceedings of the 21st International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*.

Janssen, P., & Stouffs, R. (2015). Types of parametric modelling.

Jayaram, S., Vance, J., Gadh, R., Jayaram, U., & Srinivasan, H. (2001). Assessment of VR technology and its applications to engineering problems. *Journal of Computing and Information Science in Engineering*, *1*(1), 72–83.

Jhamb, S., Enekvist, M., Liang, X., Zhang, X., Dam-Johansen, K., & Kontogeorgis, G. M. (2020). A review of computer-aided design of paints and coatings. *Current Opinion in Chemical Engineering*, *27*, 107–120.

Johnson, T. E. (1963). Sketchpad III: A computer program for drawing in three dimensions. In *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*, (pp. 347–353).

Johnston, W. M., Hanna, J. P., & Millar, R. J. (2004). Advances in dataflow programming languages. *ACM computing surveys (CSUR)*, *36*(1), 1–34.

Juan, M. C., Alcaniz, M., Monserrat, C., Botella, C., Baños, R. M., & Guerrero, B. (2005). Using augmented reality to treat phobias. *IEEE computer graphics and applications*, *25*(6), 31–37.

Juan, M. C., & Pérez, D. (2009). Comparison of the levels of presence and anxiety in an acrophobic environment viewed via hmd or cave. *Presence: Teleoperators and virtual environments*, *18*(3), 232–248.

Juan, Y.-K., Chen, H.-H., & Chi, H.-Y. (2018). Developing and evaluating a virtual reality-based navigation system for pre-sale housing sales. *Applied Sciences*, *8*(6), 952.

Kale, S., & Arditi, D. (2005). Diffusion of computer aided design technology in architectural design practice. *Journal of Construction Engineering and Management*, *131*(10), 1135–1141.

Kameyama, K.-i. (1997). Virtual clay modeling system. In *Proceedings of the ACM symposium on Virtual reality software and technology*, (pp. 197–200).

Kan, T.-W., Teng, C.-H., & Chen, M. Y. (2011). Qr code based augmented reality applications. In *Handbook of augmented reality*, (pp. 339–354). Springer.

Kato, H., & Billinghurst, M. (1999). Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, (pp. 85–94). IEEE.

Kaveh, A., Hassani, B., Shojaee, S., & Tavakkoli, S. M. (2008). Structural topology optimization using ant colony methodology. *Engineering Structures*, *30*(9), 2559–2565.

Kazi, R. H., Grossman, T., Cheong, H., Hashemi, A., & Fitzmaurice, G. W. (2017). Dreamsketch: Early stage 3d design explorations with sketching and generative design. In *UIST*, vol. 14, (pp. 401–414).

Kemmis, S., McTaggart, R., & Nixon, R. (2014). The action research planner: Doing critical participatory action research.

Khan, S., & Tunçer, B. (2019). Gesture and speech elicitation for 3d cad modeling in conceptual design. *Automation in Construction*, *106*, 102847.

Kim, M. J., & Hall, C. M. (2019). A hedonic motivation model in virtual reality tourism: Comparing visitors and non-visitors. *International Journal of Information Management*, *46*, 236–249.

Koma, S., Yamabe, Y., & Tani, A. (2017). Research on urban landscape design using the interactive genetic algorithm and 3d images. *Visualization in Engineering*, *5*(1), 1–10.

Koutamanis, A. (2005). A biased history of CAAD.

Krijn, M., Emmelkamp, P. M., Biemond, R., de Ligny, C. d. W., Schuemie, M. J., & van der Mast, C. A. (2004). Treatment of acrophobia in virtual reality: The role of immersion and presence. *Behaviour research and therapy*, *42*(2), 229–239.

Kuş, A. (2009). Implementation of 3d optical scanning technology for automotive applications. *Sensors*, *9*(3), 1967–1979.

Kwieciński, K., Markusiewicz, J., & Pasternak, A. (2017). Participatory design supported with design system and augmented reality. *SharingofComputableKnowledge!*, (p. 745).

Kymmell, W. (2008). *Building Information Modeling: Planning and Managing Construction Projects with 4D CAD and Simulations (McGraw-Hill Construction Series)*. McGraw-Hill Education.

Lam, A. H., Chow, K. C., Yau, E. H., & Lyu, M. R. (2006). Art: augmented reality table for interactive trading card game. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, (pp. 357–360).

Lanier, J. (1988). A vintage virtual reality interview.

LaViola, J. J. (2017). *3d User Interfaces: Theory and Practice*. Hoboken, NJ: Pearson Education, Inc, 2nd edition ed.

Lee, G. A., Nelles, C., Billinghurst, M., Billinghurst, M., & Kim, G. J. (2004). Immersive authoring of tangible augmented reality applications. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, (pp. 172–181). IEEE Computer Society.

Lee, W., Woo, W., & Lee, J. (2005). Tarboard: Tangible augmented reality system for table-top game environment. In *2nd International Workshop on Pervasive Gaming Applications, PerGames*, vol. 5.

Leitão, A., Santos, L., & Lopes, J. (2012). Programming languages for generative design: A comparative study. *International Journal of Architectural Computing*, *10*(1), 139–162.

Liang, J., & Green, M. (1994). Jdcad: A highly interactive 3d modeling system. *Computers & graphics*, *18*(4), 499–506.

Liggett, R. S. (2000). Automated facilities layout: past, present and future. *Automation in construction*, *9*(2), 197–215.

Light, R., & Gossard, D. (1982). Modification of geometric models through variational geometry. *Computer-Aided Design*, *14*(4), 209–214.

Light, R. A. (2017). Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, *2*(13), 265.

Lin, L., Normoyle, A., Adkins, A., Sun, Y., Robb, A., Ye, Y., Di Luca, M., & Jörg, S. (2019). The effect of hand size and interaction modality on the virtual hand illusion. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, (pp. 510–518). IEEE.

Liu, D. T., & Xu, X. W. (2001). A review of web-based product data management systems. *Computers in industry*, *44*(3), 251–262.

Lo, T. T., Xiao, Z., & Yu, H. (2019). Designing 'action trigger' for architecture modelling design within immersive virtual reality. In *Proceedings of the 24th CAADRIA Conference*.

Louis, T., Troccaz, J., Rochet-Capellan, A., Hoyek, N., & Bérard, F. (2020). When high fidelity matters: AR and VR improve the learning of a 3D object. In *Proceedings of the International Conference on Advanced Visual Interfaces*, (pp. 1–9).

Loureiro, S. M. C., Guerreiro, J., & Ali, F. (2020). 20 years of research on virtual reality and augmented reality in tourism context: A text-mining approach. *Tourism Management*, *77*, 104028.

Macalino, S. J. Y., Gosu, V., Hong, S., & Choi, S. (2015). Role of computer-aided drug design in modern drug discovery. *Archives of pharmacal research*, *38*(9), 1686–1701.

Mackay, W. E. (1996). Augmenting reality: A new paradigm for interacting with computers. *La recherche*, *284*.

MacKenzie, I. S. (1992). Fitts' law as a research and design tool in human-computer interaction. *Human-computer interaction*, *7*(1), 91–139.

MacKenzie, I. S. (2012). Human-computer interaction: An empirical research perspective.

Maekawa, M. (1985). A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems (TOCS)*, *3*(2), 145–159.

Maher, A., & Burry, M. (2003). The parametric bridge: connecting digital design techniques in architecture and engineering. In *Proceedings of the ACADIA Conference*.

Maleki, M. M., & Woodbury, R. F. (2013). Programming in the model—a new scripting interface for parametric cad systems. In *Proceedings of the 33rd Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*.

Mandell, D. J., & Kortemme, T. (2009). Computer-aided design of functional protein interactions. *Nature chemical biology*, *5*(11), 797–807.

Mapes, D. P., & Moshell, J. M. (1995). A two-handed interface for object manipulation in virtual environments. *Presence: Teleoperators & Virtual Environments*, *4*(4), 403–416.

Marin, P., Marsault, X., Saleri, R., & Duchanois, G. (2012). Creativity with the help of evolutionary design tool.

Martel, E., Su, F., Gerroir, J., Hassan, A., Girouard, A., & Muldner, K. (2015). Diving head-first into virtual reality: Evaluating hmd control schemes for vr games. In *FDG*.

McKee, S., Nelson, N., Sarma, A., & Dig, D. (2017). Software practitioner perspectives on merge conflicts and resolutions. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, (pp. 467–478). IEEE.

Mens, T. (2002). A state-of-the-art survey on software merging. *IEEE transactions on software engineering*, *28*(5), 449–462.

Mens, T. (2005). On the use of graph transformations for model refactoring. In *International Summer School on Generative and Transformational Techniques in Software Engineering*, (pp. 219–257). Springer.

Mens, T., & Tourwé, T. (2004). A survey of software refactoring. *IEEE Transactions on software engineering*, *30*(2), 126–139.

Michalski, S. C., Szpak, A., Saredakis, D., Ross, T. J., Billinghurst, M., & Loetscher, T. (2019). Getting your game on: Using virtual reality to improve real table tennis skills. *PloS one*, *14*(9), e0222351.

Miles, J. C., Sisk, G., & Moore, C. J. (2001). The conceptual design of commercial buildings using a genetic algorithm. *Computers & Structures*, *79*(17), 1583–1592.

Milgram, P., & Kishino, F. (1994). A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, *77*(12), 1321–1329.

Milgram, P., Takemura, H., Utsumi, A., & Kishino, F. (1995). Augmented reality: A class of displays on the reality-virtuality continuum. In *Telemanipulator and telepresence technologies*, vol. 2351, (pp. 282–292). International Society for Optics and Photonics.

Mine, M., Yoganandan, A., & Coffey, D. (2014). Making vr work: building a real-world immersive modeling application in the virtual world. In *Proceedings of the 2nd ACM symposium on Spatial user interaction*, (pp. 80–89).

Mitchell, W. J. (1989). Afterword: The design studio of the future. In *The Electronic Design Studio: Architectural Knowledge and Media in the Computer Era*, (pp. 479–494). Cambridge, USA.

Mobach, M. P. (2008). Do virtual worlds create better real worlds? *Virtual Reality*, *12*(3), 163–179.

Molla, E., & Lepetit, V. (2010). Augmented reality for board games. In *2010 IEEE International Symposium on Mixed and Augmented Reality*, (pp. 253–254). IEEE.

Monedero, J. (2000). Parametric design: A review and some experiences. *Automation in Construction*, *9*(4), 369–377.

Morrison, J. P. (1994). *Flow-Based Programming: A New Approach to Application Development*. Von Nostrand Reinhold.

Mossel, A., Venditti, B., & Kaufmann, H. (2013). 3dtouch and homer-s: intuitive manipulation techniques for one-handed handheld augmented reality. In *Proceedings of the virtual reality international conference: laval virtual*, (pp. 1–10).

Moubile, M. (2018). *Towards Real-Time Parametric Structural Modeling in Virtual Reality Using a Game Engine*. Master's thesis.

Mousavi Hondori, H., Khademi, M., Dodakian, L., Cramer, S. C., & Lopes, C. V. (2013). A spatial augmented reality rehab system for post-stroke hand rehabilitation. In *Medicine Meets Virtual Reality 20*, (pp. 279–285). IOS Press.

Mueller, C. T., & Ochsendorf, J. A. (2015). Combining structural performance and designer preferences in evolutionary design space exploration. *Automation in Construction*, *52*, 70–82.

Munson, J., & Dewan, P. (1996). A concurrency control framework for collaborative systems. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, (pp. 278–287).

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, *1*(1), 97–123.

Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, L., Wang, R., Zhao, D., & Benjamin, D. (2017a). Project Discover: An Application of Generative Design for Architectural Space Planning. In *Symposium on Simulation for Architecture and Urban Design*.

Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, L., Wang, R., Zhao, D., & Benjamin, D. (2017b). Project discover: An application of generative design for architectural space planning. In *Proceedings of the Symposium on Simulation for Architecture and Urban Design*, (pp. 1–8).

Najork, M. A. (1996). Programming in three dimensions. *Journal of Visual Languages & Computing*, *7*(2), 219–242.

Najork, M. A., & Kaplan, S. M. (1991). The CUBE languages. In *Proceedings 1991 IEEE Workshop on Visual Languages*, (pp. 218–224). IEEE.

Nasman, J., & Cutler, B. (2013). Evaluation of user interaction with daylighting simulation in a tangible user interface. *Automation in construction*, *36*, 117–127.

Nguyen, T. T. H., & Duval, T. (2014). A survey of communication and awareness in collaborative virtual environments. In *Collaborative Virtual Environments (3DCVE), 2014 International Workshop On*, (pp. 1–8). IEEE.

Nicholson, D. T., Chalk, C., Funnell, W. R. J., & Daniel, S. J. (2006). Can virtual reality improve anatomy education? a randomised controlled study of a computer-generated three-dimensional anatomical ear model. *Medical education*, *40*(11), 1081–1087.

Nicholson-Cole, D. (1998). *The GDL Cookbook*. Marmalade Graphics.

Noh, Z., Sunar, M. S., & Pan, Z. (2009). A review on augmented reality for virtual heritage system. In *International conference on technologies for E-learning and digital entertainment*, (pp. 50–61). Springer.

Object Management Group (OMG) (2017). OMG® Unified Modeling Lan-

guage (OMG UML), Version 2.5.1. OMG Document Number formal/2017-12-05 (`https/www.omg.org/spec/UML/`).

O'Kelly, M., Abbas, H., & Mangharam, R. (2017). Computer-aided design for safe autonomous vehicles. In *2017 Resilience Week (RWS)*, (pp. 90–96). IEEE.

on 3dbiology.com, D. J. (2018). Nurbs vs. bezier: What's the difference? URL `https://www.3dbiology.com/nurbs-vs-bezier-whats-the-difference/`

Paes, D., Arantes, E., & 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*, 292–303.

Pheasant, S., & Haslegrave, C. M. (2018). *Bodyspace: Anthropometry, Ergonomics and the Design of Work*. CRC Press.

Pieraccini, R., & Huerta, J. (2005). Where do we go from here? research and commercial spoken dialog systems. In *6th SIGdial Workshop on Discourse and Dialogue*.

Pierce, J. S., Stearns, B. C., & Pausch, R. (1999). Voodoo dolls: seamless interaction at multiple scales in virtual environments. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, (pp. 141–145).

Postel, J. (1980). Rfc0768: User datagram protocol.

Postel, J. (2011). Transmission control protocol.

Poupyrev, I., Billinghurst, M., Weghorst, S., & Ichikawa, T. (1996). The go-go interaction technique: non-linear mapping for direct manipulation in vr. In *ACM Symposium on User Interface Software and Technology*, (pp. 79–80). Citeseer.

Poupyrev, I., Ichikawa, T., Weghorst, S., & Billinghurst, M. (1998). Egocentric object manipulation in virtual environments: empirical evaluation of interaction techniques. In *Computer graphics forum*, vol. 17, (pp. 41–52). Wiley Online Library.

Pratt, M. J. (1984). Solid modeling and the interface between design and manufacture. *IEEE Computer Graphics and Applications*, *4*(7), 52–59.

Pratt, M. J. (2001). Introduction to iso 10303—the step standard for product data exchange. *Journal of Computing and Information Science in Engineering*, *1*(1), 102–103.

Pratt, M. J. (2005). Iso 10303, the step standard for product data exchange, and its plm capabilities. *International Journal of Product Lifecycle Management*, *1*(1), 86–94.

Pryor, H. L., Furness III, T. A., & Viirre III, E. (1998). The virtual retinal display: A new display technology using scanned laser light. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 42, (pp. 1570–1574). SAGE Publications Sage CA: Los Angeles, CA.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Raskar, R., Welch, G., Cutts, M., Lake, A., Stesin, L., & Fuchs, H. (1998). The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, (pp. 179–188).

Reiners, D., Stricker, D., Klinker, G., & Müller, S. (1999). Augmented reality for construction tasks: Doorlock assembly. *Proc. Ieee And Acm Iwar*, *98*(1), 31–46.

Renevier, P. (2004). *Mobile Collaborative Mixed Systems : Design and Development*. Ph.D. thesis, Université Joseph-Fourier - Grenoble I.

Rizzo, A., Reger, G., Gahm, G., Difede, J., & Rothbaum, B. O. (2009). Virtual reality exposure therapy for combat-related ptsd. In *Post-traumatic stress disorder*, (pp. 375–399). Springer.

Rose, F. D., Brooks, B. M., & Rizzo, A. A. (2005). Virtual reality in brain damage rehabilitation. *Cyberpsychology & behavior*, *8*(3), 241–262.

Ross, D. T. (1960). *Computer-aided design: A statement of objectives*. MIT Electronic Systems Laboratory.

Roupé, M., Johansson, M., Viklund Tallgren, M., Jörnebrant, F., & Tomsa, P. A. (2016). Immersive visualization of Building Information Models. In *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 2016), (pp. 673–682).

Salvendy, G. (2006). *Handbook of human factors and ergonomics*, vol. 144. Wiley New York.

Sammer, M., Leitão, A., & Caetano, I. (2019). From visual input to visual output in textual programming.

Sato, Y., Nakamoto, M., Tamaki, Y., Sasama, T., Sakita, I., Nakajima, Y., Monden, M., & Tamura, S. (1998). Image guidance of breast cancer surgery using 3-d ultrasound images and augmented reality visualization. *IEEE Transactions on Medical Imaging*, *17*(5), 681–693.

Schnabel, M. A., & Kvan, T. (2003). Spatial understanding in immersive virtual environments. *International Journal of Architectural Computing*, *1*(4), 435–448.

Schowengerdt, B. T., Seibel, E. J., Kelly, J. P., Silverman, N. L., & Furness III, T. A. (2003). Binocular retinal scanning laser display with integrated focus cues for ocular accommodation. In *Stereoscopic Displays and Virtual Reality Systems X*, vol. 5006, (pp. 1–9). International Society for Optics and Photonics.

Schwald, B., & De Laval, B. (2003). An augmented reality system for training and assistance to maintenance in the industrial context.

Serrano, A., Sitzmann, V., Ruiz-Borau, J., Wetzstein, G., Gutierrez, D., & Masia, B. (2017). Movie editing and cognitive event segmentation in virtual reality video. *ACM Transactions on Graphics (TOG)*, *36*(4), 1–12.

Shah, J. J. (1998). Designing with parametric cad: Classification and comparison of construction techniques. In *International Workshop on Geometric Modelling*, (pp. 53–68). Springer.

Shah, J. J., & Mäntylä, M. (1995). *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*. John Wiley & Sons.

Shah, J. J., & Rogers, M. (1988). Expert form feature modelling shell. *computer-aided design*, *20*(9), 515–524.

Skvorc, D., Horvat, M., & Srbljic, S. (2014). Performance evaluation of websocket protocol for implementation of full-duplex web streams. In *2014 37th*

International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), (pp. 1003–1008). IEEE.

Smith, B. (1983). Iges: a key to cad/cam systems integration. *IEEE Computer Graphics and Applications*, (8), 78–83.

Smith, B., & Wellington, J. (1986). Initial graphics exchange specification (iges); version 3.0. Tech. rep., US. Nat. Bureau Stand.

Stals, A., & Caldas, L. (2020). State of XR research in architecture with focus on professional practice–a systematic literature review. *Architectural Science Review*, (pp. 1–9).

Stals, A., Elsen, C., & Jancart, S. (2018a). L'immersion pour l'appréhension des outils de modélisation paramétrique en conception architecturale. In *SHS Web of Conferences*, vol. 47, (p. 01010). EDP Sciences.

Stals, A., Jancart, S., & Elsen, C. (2018b). Influence of parametric tools on the complexity of architectural design in everyday work os sme's. *Archnet-IJAR*, *12*(3), 206–227.

Steed, A., & Slater, M. (1996). A dataflow representation for defining behaviours within virtual environments. In *Proceedings of the IEEE 1996 Virtual Reality Annual International Symposium*, (pp. 163–167). IEEE.

Steiner, I. D. (1972). *Group process and productivity*. Academic press.

Stiles, R., & Pontecorvo, M. (1992). Lingua graphica: A visual language for virtual environments. In *Proceedings IEEE Workshop on Visual Languages*, (pp. 225–227). IEEE.

Stoakley, R., Conway, M. J., & Pausch, R. (1995). Virtual reality on a wim: interactive worlds in miniature. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 265–272).

Suh, A., & Prophet, J. (2018). The state of immersive technology research: A literature analysis. *Computers in Human Behavior*, *86*, 77–90.

Sutherland, I. E. (1964). Sketchpad a man-machine graphical communication system. *Simulation*, *2*(5), R–3.

Sutherland, I. E. (1968). A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, (pp. 757–764).

Taentzer, G. (1999). Agg: A tool environment for algebraic graph transformation. In *International Workshop on Applications of Graph Transformations with Industrial Relevance*, (pp. 481–488). Springer.

Takahashi, H., & Hirooka, S. (2008). Stereoscopic see-through retinal projection head-mounted display. In *Stereoscopic displays and applications XIX*, vol. 6803, (p. 68031N). International Society for Optics and Photonics.

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*, vol. 74. John Wiley & Sons.

Tan, C. T., Leong, T. W., Shen, S., Dubravs, C., & Si, C. (2015). Exploring gameplay experiences on the oculus rift. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, (pp. 253–263).

Thomas, B., Close, B., Donoghue, J., Squires, J., De Bondi, P., & Piekarski, W. (2002). First person indoor/outdoor augmented reality application: Arquake. *Personal and Ubiquitous Computing*, *6*(1), 75–86.

Tobias, E., Maquil, V., & Latour, T. (2015). Tulip: a widget-based software framework for tangible tabletop interfaces. In *Proceedings of the 7th acm sigchi symposium on engineering interactive computing systems*, (pp. 216–221).

Tornincasa, S., Di Monaco, F., et al. (2010). The future and the evolution of CAD. In *Proceedings of the 14th International Research/Expert Conference: Trends in the Development of Machinery and Associated Technology*, vol. 1, (pp. 11–18).

Turrin, M., Von Buelow, P., & Stouffs, R. (2011). Design explorations of performance driven geometry in architectural design using parametric modeling and genetic algorithms. *Advanced Engineering Informatics*, *25*(4), 656–675.

Turrin, M., Von Buelow, P., Stouffs, R., & Kilian, A. (2010). Performance-oriented design of large passive solar roofs. In *A method for the integration of parametric modeling and genetic algorithms. in: Proceedings of eCAADe 2010, 28th Conference. Future Cities. Zurich, Switzerland, 15–18 September*, vol. 2010, (pp. 321–330).

Versprille, K. J. (1975). *Computer-aided design applications of the rational b-spline approximation form.*. Syracuse University.

Villaggi, L., Stoddart, J., Nagy, D., & Benjamin, D. (2018a). Survey-Based

Simulation of User Satisfaction for Generative Design in Architecture. In K. De Rycke, C. Gengnagel, O. Baverel, J. Burry, C. Mueller, M. M. Nguyen, P. Rahm, & M. R. Thomsen (Eds.) *Humanizing Digital Reality*, (pp. 417–430). Singapore: Springer Singapore.

Villaggi, L., Stoddart, J., Nagy, D., & Benjamin, D. (2018b). Survey-based simulation of user satisfaction for generative design in architecture. In *Humanizing Digital Reality*, (pp. 417–430). Springer.

Vlahakis, V., Karigiannis, J., Tsotros, M., Gounaris, M., Almeida, L., Stricker, D., Gleue, T., Christou, I. T., Carlucci, R., Ioannidis, N., et al. (2001). Archeoguide: first results of an augmented reality, mobile computing system in cultural heritage sites. *Virtual Reality, Archeology, and Cultural Heritage*, *9*(10.1145), 584993–585015.

Voelcker, H. B., & Requicha, A. A. (1977). Geometric modeling of mechanical parts and processes. *Computer*, *10*(12), 48–57.

Wang, T., Qian, X., He, F., Hu, X., Huo, K., Cao, Y., & Ramani, K. (2020). Capturar: An augmented reality tool for authoring human-involved context-aware applications. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, (pp. 328–341).

Wang, Y.-Y., & Ju, Y.-C. (2004). Creating speech recognition grammars from regular expressions for alphanumeric concepts. In *Eighth International Conference on Spoken Language Processing*.

Welch, G., & Foxlin, E. (2002). Motion tracking: No silver bullet, but a respectable arsenal. *IEEE Computer graphics and Applications*, *22*(6), 24–38.

Wesche, G., & Seidel, H.-P. (2001). Freedrawer: a free-form sketching system on the responsive workbench. In *Proceedings of the ACM symposium on Virtual reality software and technology*, (pp. 167–174).

Willième, R. (2020). *Interactions gestuelles pour la manipulation de graphes en réalité virtuelle*. Master's thesis, University of Mons.

Wilson, P. (1987). A short history of cad data transfer standards. *IEEE Computer Graphics and Applications*, (6), 64–67.

Wolf, P. R., Dewitt, B. A., & Wilkinson, B. E. (2014). *Elements of Photogrammetry with Applications in GIS*, chap. 1, (pp. 11–13). McGraw-Hill Education.

Woodbury, R., et al. (2010). Elements of parametric design.

Wright, J., & Farmani, R. (2001). The simultaneous optimization of building fabric construction, hvac system size, and the plant control strategy. In *Proc. of the 7-th IBPSA Conference*, vol. 1, (pp. 865–872).

Wu, P., Wang, J., & Wang, X. (2016). A critical review of the use of 3-d printing in the construction industry. *Automation in Construction*, *68*, 21–31.

Wyss, H. P., Blach, R., & Bues, M. (2006). isith-intersection-based spatial interaction for two hands. In *3D User Interfaces (3DUI'06)*, (pp. 59–61). IEEE.

Xie, Y., Loh, G. H., Black, B., & Bernstein, K. (2006). Design space exploration for 3d architectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *2*(2), 65–103.

Xu, Z., Lu, X., Guan, H., Chen, C., & Ren, A. (2014). A virtual reality based fire training simulator with smoke hazard assessment capacity. *Advances in engineering software*, *68*, 1–8.

Yergeau, F., Sperberg-McQueen, M., Paoli, J., Bray, T., & Maler, E. (2008). Extensible markup language (XML) 1.0 (fifth edition). W3C recommendation, W3C. Https://www.w3.org/TR/2008/REC-xml-20081126/.

Yip, C. H. T., Chiu, T. T. W., & Poon, A. T. K. (2008). The relationship between head posture and severity and disability of patients with neck pain. *Manual therapy*, *13*(2), 148–154.

Zaman, L., Stuerzlinger, W., & Neugebauer, C. (2017). Mace: A new interface for comparing and editing of multiple alternative documents for generative design. In *Proceedings of the 2017 ACM Symposium on Document Engineering*, (pp. 67–76).

Zhang, J., Tai, L., Liu, M., Boedecker, J., & Burgard, W. (2017). Neural slam: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*.

Zhang, L., & Oney, S. (2020). Flowmatic: An immersive authoring tool for creating interactive scenes in virtual reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, (pp. 342–353).

Zollmann, S., Langlotz, T., Loos, M., Lo, W. H., & Baker, L. (2019). Arspectator: Exploring augmented reality for sport events. In *SIGGRAPH Asia 2019 Technical Briefs*, (pp. 75–78).

Integrating Immersive Technologies for Algorithmic Design in Architecture

Architectural design practice has radically evolved over the course of its history, due to technological improvements that gave rise to advanced automated tools for many design tasks. Traditional paper drawings and scale models are now accompanied by 2D and 3D Computer-Aided Architectural Design (CAAD) software.

While such tools improved in many ways, including performance and accuracy improvements, the modalities of user interaction have mostly remained the same, with 2D interfaces displayed on 2D screens. The maturation of Augmented Reality (AR) and Virtual Reality (VR) technology has led to some level of integration of these immersive technologies into architectural practice, but mostly limited to visualisation purposes, e.g. to show a finished project to a potential client.

We posit that there is potential to employ such technologies earlier in the architectural design process and therefore explore that possibility with a focus on Algorithmic Design (AD), a CAAD paradigm that relies on (often visual) algorithms to generate geometries. The main goal of this dissertation is to demonstrate that AR and VR can be adopted for AD activities.

To verify that claim, we follow an iterative prototype-based methodology to develop research prototype software tools and evaluate them. The three developed prototypes provide evidence that integrating immersive technologies into the AD toolset provides opportunities for architects to improve their workflow and to better present their creations to clients. Based on our contributions and the feedback we gathered from architectural students and other researchers that evaluated the developed prototypes, we additionally provide insights as to future perspectives in the field.

UMONS
Université de Mons