CAULDRON: A REAL-TIME PICTORIAL COMPOSITIONAL TOOL IN VR FOR VISUAL ARTIST

BY

MATHUS TUACHOB

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF MASTER OF
ENGINEERING (INFORMATION AND COMMUNICATION
TECHNOLOGY FOR EMBEDDED SYSTEMS)
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY
THAMMASAT UNIVERSITY
ACADEMIC YEAR 2018

CAULDRON: A REAL-TIME PICTORIAL COMPOSITIONAL TOOL IN VR FOR VISUAL ARTIST

BY

MATHUS TUACHOB

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF MASTER OF
ENGINEERING (INFORMATION AND COMMUNICATION
TECHNOLOGY FOR EMBEDDED SYSTEMS)
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY
THAMMASAT UNIVERSITY
ACADEMIC YEAR 2018

CAUDRON: A REAL-TIME PICTORIAL COMPOSITIONAL TOOL IN VR FOR VISUAL ARTIST

A Thesis Presented

By

MATHUS TUACHOB

Submitted to

Sirindhorn International Institute of Technology

Thammasat University

In partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING (INFORMATION AND COMMUNICATION TECHNOLOGY FOR EMBEDDED SYSTEMS)

Approved as to style and content by

Advisor and Chairperson of Thesis Committee [illegible]

(Assistant Professor Itthisek Nilkhamhang, Ph.D.)

Co-Advisor Toshiaki Kondo

(Associate Professor Toshiaki Kondo, Ph.D.)

Committee Member and
Chairperson of Examination Committee Pished Bunnun

(Pished Bunnun, Ph.D.)

Committee Member Shoichi Hasegawa

(Associate Professor Shoichi Hasegawa, Ph.D.)

AUGUST 2018

Abstract

CAUDRON: A REAL-TIME PICTORIAL COMPOSITIONAL TOOL IN VR FOR VISUAL ARTIST

By

MATHUS TUACHOB

Bachelor of Science in Information Technology
Sirindhorn International Institute of Technology, 2009
Master of Engineering in Information and Communication Technology for Embedded System, Sirindhorn International Institute of Technology, 2018

In this thesis, we describe the development of an immersive VR application that lets visual artists quickly visualize and prototype their pictorial design ideas in virtual environments. The design requirements are outlined first, and then the implementation of the application prototype in Unreal Engine 4 is explained. We perform preliminary user tests and discuss the feedback, as well as further improvements and usage scenario of the software.

We also describe the development of an interaction technique to manipulate multiple objects in VR, in context of a design software. The technique is developed specifically to reduce repetitive actions and aimed to improve efficiency when designing in VR by assign 'command' to multiple objects and let them execute it automatically, allows the user to work on a bigger picture. We describe the idea and concept for the technique, explain the implementation in Unreal Engine 4, and performed user test with 30 subjects and discussed the results.

Keywords: Virtual Reality, Visualization, Virtual Object Manipulation, Human-Computer Interaction, Design Practice

Acknowledgements

This research is financially supported by Thailand Advanced Institute of Science and Technology (TAIST), National Science and Technology Development Agency (NSTDA), Tokyo Institute of Technology and Sirindhorn International Institute of Technology (SIIT), Thammasat University (TU).

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Background and Motivation

1.1.1 Design Tools for Visual Artists

Creating an image is hard. Visual artists, such as painters, illustrators, comic-artists, and photographers, employ a variety of tricks and techniques to translate their vision or ideas into the final image [Gurney, J. 2009]. Numerous sketches, drawings, clay models, toys and photographs of real-world objects are used as references, as shown in Fig. 1.1. However, this process of preparing and gathering references can be long, laborious and costly. Also, sometimes the design involves entirely made-up locations or situations that make it impossible to find a reference. Therefore, the artist needs to combine various images or objects to try and work out perspective, lighting information and other technical aspects of the images, with the aim of creating a believable end result. This makes the process even more complicated and takes up additional resources.

Furthermore, seemingly great ideas can sometime refuse to lend themselves easily into great artworks due to insufficient preparation from the artist. This occurs when the artist does not have a concrete idea of what he is trying to accomplish and therefore needs to spend time figuring things out on the set of reference photoshoots, instead of creating artwork itself, or from limitation of resources and budget that ends up restricting the scale or possibility of creating the artwork in the first place.

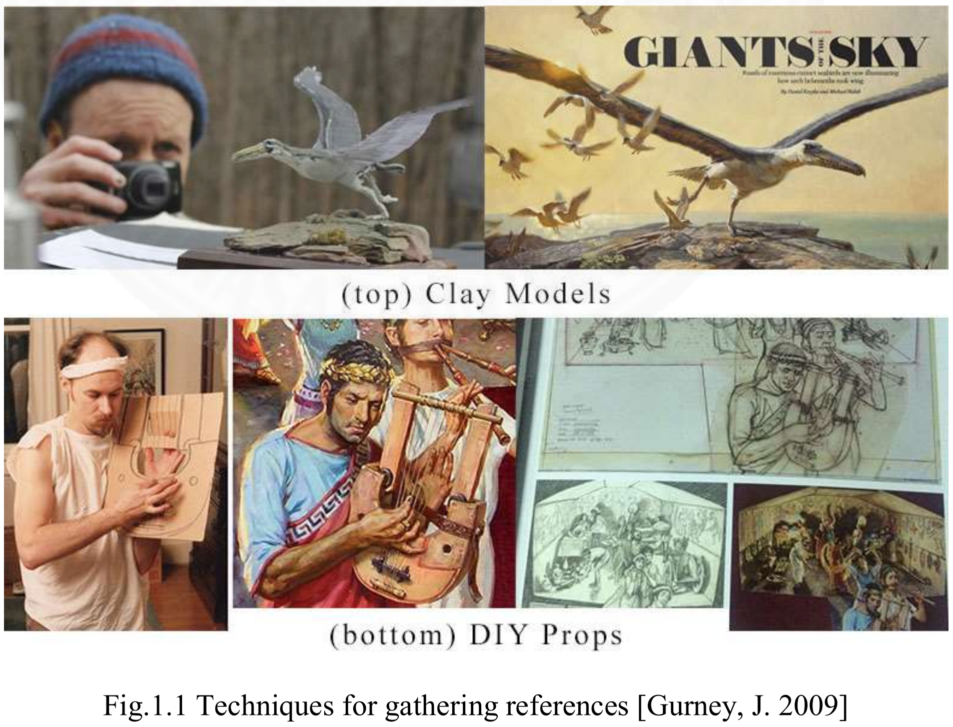


Fig.1.1 Techniques for gathering references [Gurney, J. 2009]

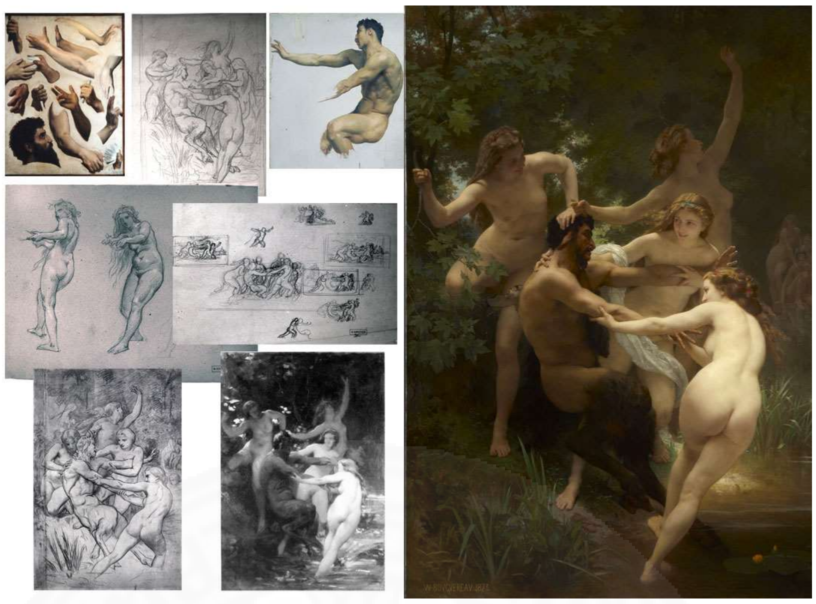


Fig.1.2 Sketches and studies for a painting [Lyon 2014]

When it comes to preparing and planning, whether for gathering references or creating artwork, many artists still rely on age-old techniques of drawing or sketching similar to what is shown in Fig. 1.2, making notes, or gathering a mood board, which is a collection of images that is similar to the result he is trying to achieve. These images or notes are then used to communicate his ideas with colleagues, such as assistants, staffs or models, or simply as a reminder for the artist himself. This can also cause problems, because sometimes the image that the artist sees in his head might not be communicated successfully to others, even with all the references. This is especially important when dealing with professional models, who are paid by the hour and not generally working close enough to know what the artist have in mind. The result is more wasted time spent on the set, or in some worst cases, the whole shoot can even produce unusable images.

To address these problems, technology can and should be used to off-load the technical aspects of the process and help the artist make a better plan. Modern-day machines and software are very powerful and capable of rendering and simulating complex phenomena, as is often seen in computer games or architectural renderings. We want to harness that power to make the image creation process more manageable and efficient. Let the machines do what it does best, that is crunching numbers, and let the artist focus on creativity.

Modern artists also uses 3D software such as Maya, 3ds Max and Daz Studio to help in creating their art work. However, such software commands very steep learning curve and require time (months if not years) and experience to master [Quora 2018]. Therefore, only artists who are already technically-inclined or CG artists who used to work digitally would consider this option, as most traditionally trained artist would simply shy away from the complexity and focused on their preferred medium instead.

The use of virtual reality (VR) and augmented reality (AR) as information visualization or design review tools have been studied since early 2000s. [Kirner et al. 2000] developed InfoVis, a virtual walk-through environment to represent data, such as number of visits, types and quantity of a museum inventory, from Cerqueira Cesar Museum, in Sao Carlos, Brazil. [Dongsik et al. 2007] developed a prototype to evaluate the design and usability of mobile devices in VR. [Marks et al. 2014] applied immersive VR visualization technique to areas of scientific and engineering data to visualize a 3D spiking neural network and CAD data of a yacht model. [Ibayashi et al. 2015] created Dollhouse VR, a system using asymmetric collaboration between multiple users in and outside of VR to design offices, restaurant, and other architecture-scale spaces. Planwell, a spatial AR interface to facilitate petroleum-engineering tasks, was developed by [Nittala et al. 2015]. [Fischer 2016] built a live programming system for digital artists to create artwork in virtual environments. Besides the selected examples, many more products and visualization projects from various industries are using VR technology to help review and evaluate the design.

Many of the existing systems or projects use VR merely as a viewing tool [Marks et al. 2014; Barreau et al. 2014; Hervy et al. 2014; Mateevitsi et al. 2008], similar to traditional 2D monitor screens, which provides a level of interaction aimed only towards the end-user. In contrast to this, our proposed framework integrates VR technology directly into the design process. We want to create a system that is appealing in terms of usefulness and immediate suitability for real, practicing professionals, according to the guideline provided by [Stolterman 2008; Roedl et al. 2013; Stolterman et al. 2012].

1.1.2 Interaction Techniques for Designing in VR

We also want to address one main problem that users will likely face when designing in VR, which is the need to perform the same basic tasks over and over again, such as moving objects, lights, etc. that can lead to physical fatigue and discomfort. Therefore, we aim to develop a new object manipulation technique that addresses this specific issue for our framework as well.

Interaction technique for manipulating objects in VR is a rich research area that has been studied for decades. Many of the techniques, such as direct hand manipulation, tracked proxy objects [Mazalek et al. 2007] and HOMER [Bowman et al. 1997] are categorized and outlined in The VR Book by Jason Jerald [Jerald 2016]. Others, such as [Kim et al. 2014], try to improve the fidelity and intuitiveness of virtual interaction, or [Lee et al. 2016], [Gugenheimer et al. 2016] explore a new way of using touch devices to interact with the virtual world, while [Clark et al. 2016] combined machine learning technique and hand gesture recognition into an interaction system.

However, some tasks require repetitive actions, such as moving many objects while creating the visual design, or involve manipulating multiple objects simultaneously. We want to build a specialized kind of interaction for designing and visualizing in VR that addresses this scenario. Such a specialized interaction system was studied by [Azhar et al. 2015] where they built a multi-layered hierarchical bounding box technique, for viewing and manipulating objects in an educational context. [Mine et al. 2014] developed an interaction technique that is geared towards immersive 3D modeling tasks. For our proposed technique, we focus on higher-level decisions, closer to natural language, instead of fine detail control with complex interfaces and numbers. This leads to it becoming more intuitive to use, even if the user might not be technologically proficient. Although not directly the same goal, we also used the design philosophy outlined in [Mine et al. 2014], as we find that both tasks are similar and therefore the design philosophy of usability can be applied for our proposed technique.

1.2 Objective of the Study

The purpose of this research is to build an immersive VR prototype application that lets the target user (visual artists) quickly visualize and prototype pictorial design ideas in virtual environments. The goal is to improve the efficiency of the art creation workflow by letting users transfer their imagination into a more concrete form, such as scenes in virtual world or images, which can then be used as references, shared among colleagues or staffs to make plans, such as the one shown in Fig. 1.3, and creative decisions more easily than trying to explain by words or other non-visual means.

We will also develop a new interaction technique for virtual object manipulation, to be used within the application. Since our application is aimed toward real-world usage, we find that existing VR interaction techniques are not quite suitable for prolonged use while performing many repetitive tasks. We will conduct experiments to test the usability and obtain user feedback, as well as compare the performance of our technique with existing methods commonly used in VR games and other applications.

We will be evaluating the following hypotheses:

H1: Users consider our application a useful add-on to their workflow.

H2: Our interaction technique is comfortable to use, relatively easy to understand and require less physical action than common techniques employed in most applications, even though it might provide less immersion.

H3: In specific types of tasks, such as moving several objects to the same location, our interaction technique performs better in terms of time and amount of click/press of a button.

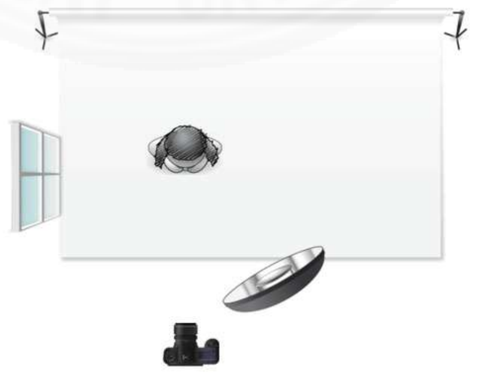


Fig.1.3 Photoshoot lighting-setup plan [Sylights 2018]

1.3 Scope and Limitation

The software prototype will consists of temporary assets and codes as a proof of concept, rather than an optimized and ready-to-deploy software. The focus will be on functionality, rather than graphical user interfaces (GUI) and other graphical assets. Due to time and resource constraints, we will make use of art contents, such as 3D models, textures, level, etc., that ships with Unreal Engine 4 (UE4) as majority of the contents in our system. This will allow us to test and let users try our prototype without spending months and months creating art assets by ourselves. Additional art contents and plugins are also purchased from Unreal Marketplace to help speed up the development of the prototype.

1.4 Thesis Outline

Chapter 1 introduces the background and motivation for the problems, as well as the objectives and scope of this study.

Chapter 2 provides the literature review of VR systems, various object manipulation schemes, and a basic understanding of UE4, the game engine that is used to create the prototype.

Chapter 3 contains the proposed prototype software framework. It outlines the design considerations, how the system was implemented, gives examples of how the system could be use in real design work, and user evaluation of the prototype.

Chapter 4 proposes the command-based object manipulation technique in VR for visualization and design tasks. The concepts and implementation of the technique is explained, along with the workflow/process for using the prototype. It also includes performance and user evaluation, as well as a discussion of the results.

Chapter 5 gives the conclusion and suggestions for further study.

Chapter 2
Literature Review

2.1 Virtual Reality

Virtual reality (VR) is defined by [Merriam-Webster 2018] as "an artificial environment which is experienced through sensory stimuli (as sight and sounds) provided by a computer and in which one's actions partially determine what happens in the environment." It has been invented and studied since the 1800s, according to [Jerald 2016]. There are many forms of VR systems existing today. To keep it manageable, we will classify them based on visual display technology used. There are three main types: head-mounted displays, world-fixed displays, and hand-held displays. The technical challenges for VR applications will also be considered.

2.1.1 Display Technologies in VR

Head-mounted display (HMD) is a visual display that attaches to the head of the user. It employs small monitors, such as OLED, LCD, or CRT panels, placed directly in front of the user's eyes at a very close distance in order to provide stereoscopic images. HMDs can further be divided into three types: non-see-through HMDs that block all clues from the real-world; optical-see-through HMDs that overlay computer-generated imagery onto the real-world, which is considered as an augmented reality (AR) system; and video-see-through HMDs that combine the two technologies. Position and orientation tracking of the HMD is very important because the visual information needs to match user movement as they turn their head. If implemented correctly, this type of display provides the most immersive virtual experience for the users. However, it is also the most demanding in terms of computational power, technological sophistication, and subsequently has the highest cost among all of the VR system types. There are many technical challenges that need to be overcome, as will be discussed in the following section.

World-fixed displays render graphics onto a surface that does not move with the head. The surface can be a standard monitor, known as fish-tank VR, or a display wall that completely surrounds the user, also called a cave system. Display surfaces are typically flat, but complex shapes can sometimes be used as well. For this type of display, head tracking is also important, but not as critical in terms of accuracy and latency as HMDs. High-end systems can also be highly immersive but can become expensive as it requires multiple high-resolution and large-sized displays or multiple projectors, as well as space to implement the whole system. World-fixed displays are sometimes considered mixed reality because real-world objects can easily be used as part of the experience. Examples include driving or airplane simulators that have a physical cockpit.

Hand-held displays are devices that can be held with the hand(s) and has recently grown in popularity due to the spread of smartphones and tablets. They usually do not require precise tracking or alignment with the head or eye of the user, but it is possible to do so using some sensors embedded within the device itself. This requires less computational power and system requirements than the other two display technologies.

2.1.2 Technical Challenges for VR Application

There are many challenges when creating a VR application. The following are some technical considerations that the creator needs to be aware of when creating VR experiences, such as latency, computational power, screen resolutions, physical fatigue and other health effects.

Latency has been, and still is, the biggest technical challenge that VR faces. It refers to the time a system takes to respond to a user's action or movement. Latencies in HMD-based system appear as lag or mismatch between head movement and visual or other perceptual cues, causing sensory conflict that in turn leads to simulation sickness. It is therefore essential for good VR experiences to reduce latency as much as possible.

When displaying stereoscopic images in VR, as shown in Fig. 2.1, the screen needs to draw the content twice, once for each eye, essentially cutting down the effective framerate by half. In order to ensure a smooth experience, it is therefore required to have a very powerful PC to run the VR system, as well as contents that are specifically optimized for VR, to hit the target framerate. In addition, most modern rendering technologies that are used in video games are still too computationally expensive to be used in VR. Thus, VR experiences today need to sacrifice image quality and graphical fidelity for performance in order to keep the latency low.

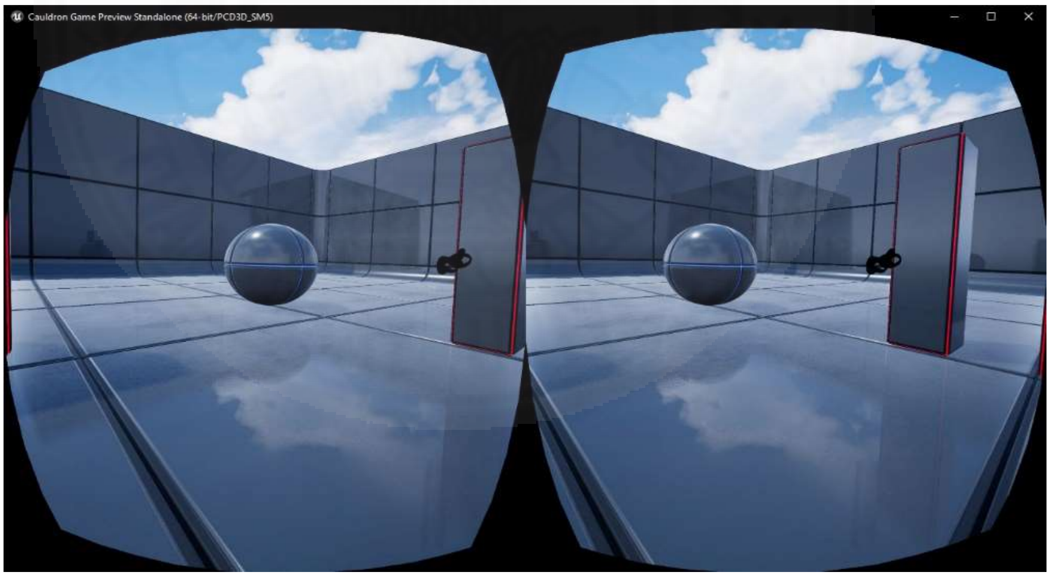


Fig.2.1 Stereoscopic rendering inside HMD

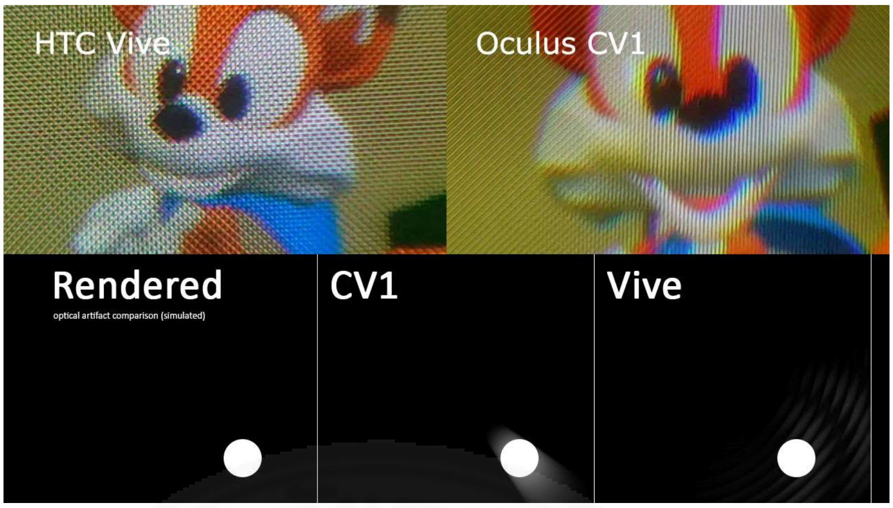


Fig.2.2 Rendering artifacts in modern HMDs include the screen-door (top) [Cerga 2018] and halo effects (bottom) [Metaclassofnil 2016]

The limitation of current display technology is also a factor because the HMD screen sits very close to the eye, causing users to easily discern pixelation or notice the blurriness of the overall picture due to low screen resolution. This leads to decreased legibility of texts inside VR, known as the screen-door effect, as well as other rendering artifacts, as shown in Fig. 2.2. While we can push the screen resolution higher, that would also mean the computer needs to have more power in order to hit the target framerate because of the increased number of pixels to render, otherwise latency will be increased.

Designing a VR experience to be comfortable to use is also an important goal that is difficult to achieve. The experience of using futuristic interfaces is completely different in real-life from the ones seen in movies. Having to lift and hold your arms up can be exhausting after only a few minutes, which refers to a symptom called gorilla arms. In addition, due to the limitations of currently available technology, using VR HMDs for more than twenty minutes will likely induce some sort of fatigue and/or nausea. Lastly, hygiene is a major concern, because the HMD will be stuck on the user's face for the duration of the experience, causing sweat to build up, especially for games or any application that require many physical motions.

Simulation sickness, also known as cyber or VR sickness, refers to sickness that is caused by using VR. Symptoms such as nausea, headache, eye strain, blurred vision and dizziness can result from a poor VR experience. While certain users, such as women and children, are more susceptible than others [LaViola 2000], many researches have been done in this field in order to provide better experiences for everyone who wishes to use VR. Therefore, certain guidelines exist for developing content for VR, such as Oculus Best Practices [Oculus (1), (2) 2018], Google Cardboard Guidelines [Google 2018], Walt Disney Imagineering [Mine et al. 2014], Epic Games [Unreal Engine Documentation 2018], Intel [Intel 2018].

The following are some examples of these guidelines to ensure better VR experiences for users.

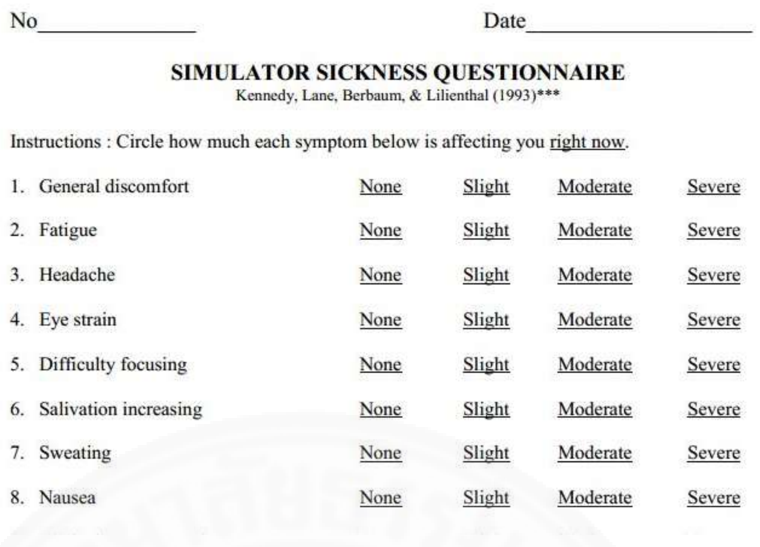
- Optimize the application as much as possible, in order to hit the target refresh rate of the HMD, which is 90Hz for most modern HMDs. Lag and delays in rendering can cause information miss-match and leads to simulation sickness.
- Design the experience to match with real-world counterpart as much as possible in order to reduce chance of simulation sickness. Include matching the scale of user and objects in the world to be 1:1 with real-world, avoid taking viewpoint control from the user, have user rotate in real-world instead of in VR, have constant velocity instead of ramping-up acceleration, reduce field-of-view (especially when moving) and try to minimize user's head movement or discourage any sudden head turns.
- Use diegetic interface, which is UI that exists in actual 3D space, instead of traditional 2D menu overlay on screen. Also design UI to operate while user's hand are at low at their side or laps to avoid arm fatigue.
- Design for shorter experience, preferably within 20 minutes or less, and try to reduce repetitive actions as much as possible.

2.2 Evaluating a VR Experience

A VR experience can be difficult to quantify. Adverse health effects caused by VR, such as simulation sickness, consist of many symptoms and cannot be measured by a single variable [Llorach at al. 2014; Bigoin at al. 2007]. There is also a large variation in terms of physical ability, tolerance and experiences between individuals [Arns et al. 2005; Vinson et al. 2012]. Therefore, questionnaires or symptom checklists are commonly utilized. These methods have a long history of use, but are subjective and require the participant to correctly identify and report the results.

The Kennedy simulator sickness questionnaire (SSQ) [Kennedy et al. 1993], similar to the one shown in Fig. 2.3, has become a standard for measuring simulator sickness and can provide clues to what need improvements in the VR application. The SSQ, in conjunction with a Likert-type scale [Likert 1932], was used to measure the user experience in our study.

The performance of interaction and virtual object manipulation techniques, such as [Kim et al. 2014; Kaul et al. 2017; Mine et al. 1997], are measured by evaluation procedures that typically consist of building test environment(s) and having users complete specific task(s). The test collects key performance data in order to compare with other techniques. Task completion time (measured in seconds) is the data often used to evaluate efficiency. In addition, the number of action performed and error rates are also useful for measuring performance. This study will focus on the task completion time and number of action performed instead of error and accuracy, as are often found in other studies, because it is more relevant to the objective of minimizing the amount of repetitive tasks.


No______________ Date____________________

SIMULATOR SICKNESS QUESTIONNAIRE

Kennedy, Lane, Berbaum, & Lilienthal (1993)***

Instructions : Circle how much each symptom below is affecting you right now.

1. General discomfort	None	Slight	Moderate	Severe
2. Fatigue	None	Slight	Moderate	Severe
3. Headache	None	Slight	Moderate	Severe
4. Eye strain	None	Slight	Moderate	Severe
5. Difficulty focusing	None	Slight	Moderate	Severe
6. Salivation increasing	None	Slight	Moderate	Severe
7. Sweating	None	Slight	Moderate	Severe
8. Nausea	None	Slight	Moderate	Severe

Fig.2.3 Example of SSQ [UQO Cyberpsychology Lab 2013]

2.3 Interaction Patterns in VR

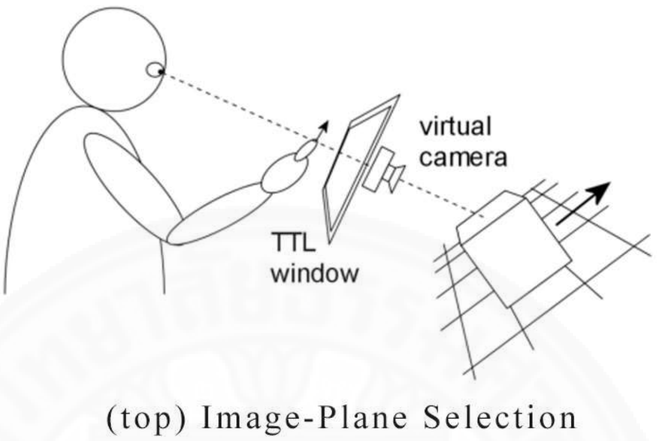
This subsection overview several examples of interaction pattern that relate to HMD-based VR and to our study. An interaction patterns refers to "a generalized high-level interaction concept that can be used over and over again across different applications to achieve common user goals." [Jerald 2016] It is a high-level design concept, from a user point of view, and often implementation independent.

2.3.1 Object Selection

Selection means to specify one or more objects from a group in order to set commands. Selection patterns in VR include hand selection, pointing, image plane selection and volume-based selection.

Hand selection uses direct object grabbing/touching in a realistic manner that is similar to the real-world. The hand in VR can be both realistic and non-realistic, such as a 3D cursor or shaped like the controller itself. The go-go technique [Poupyrev et al. 1996] was developed to allow the user to reach for object(s) that is far beyond normal reach, which is the limitation of hand selection techniques in general. It works by growing the length of the arm in a nonlinear manner the further the hand is away from the user body.

Pointing is the most fundamental and commonly used method for selection. It works by casting a ray into the distance and the first object that it intersects then can be selected. In most HMD-based VR today, head pointing patterns can be used when there is no tracked controller present or a realistic interaction is not required, such as selecting UI buttons. However, a delay is typical between head pointing and the actual select action. Two-handed pointing [Mine et al. 1997] is a technique to improve precision by casting a ray from one hand through another hand, allowing for increased control of ray rotation. Similarly, image-plane selection is a combination of eye and hand position for selection. The user holds one or two hands between the eyes and the desired object, then provides the signal to select when the object is lined up with the hand and eye, as illustrated in Fig. 2.4 (top). Lastly, volume-based selection uses a volume, such as cone, box or sphere, to determine the selection. The data itself can be volumetric (voxels), point clouds or surfaces. This selection pattern is often used in medical CT datasets, as shown in Fig. 2.4 (bottom).



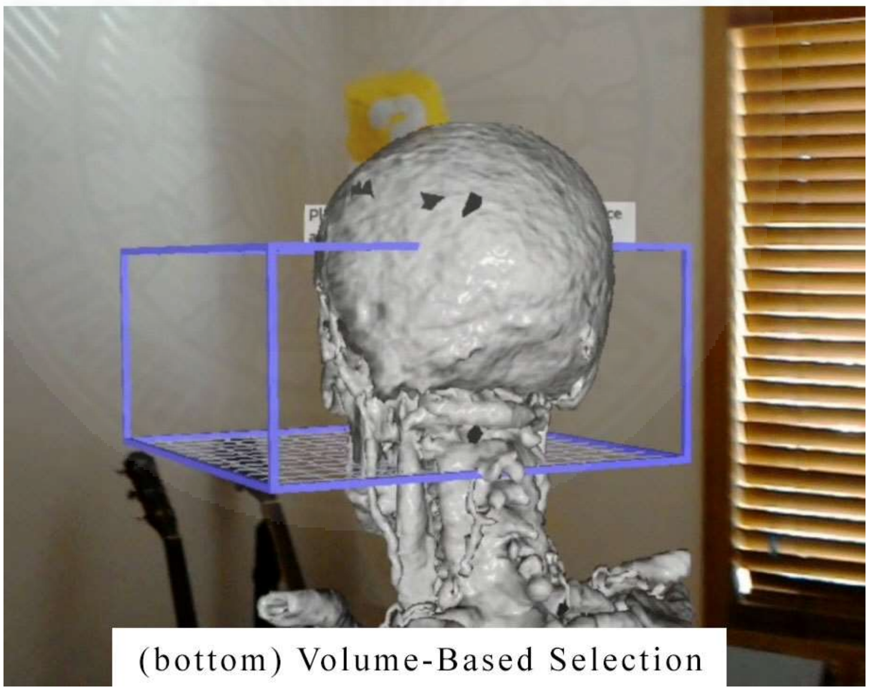


Fig. 2.4 Object selection patterns [Hill et al. 2008]

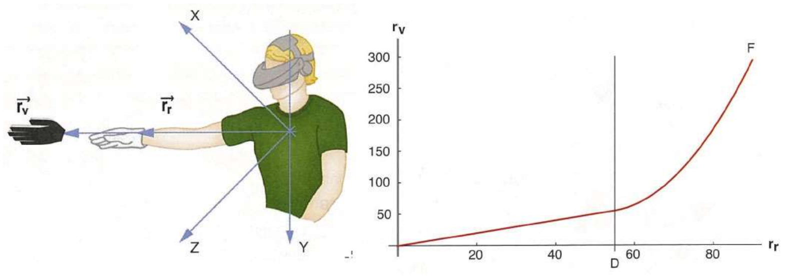


Fig.2.5 Go-go technique [Billinghurst 2017]

2.3.2 Object Manipulation

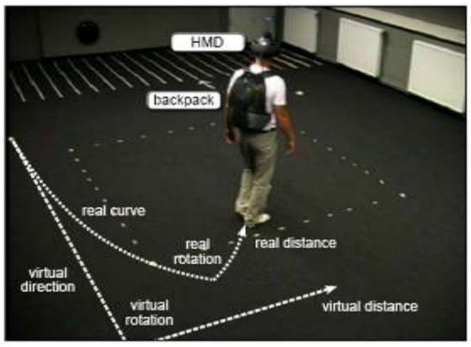
Manipulation involves the modification of attributes, such as position, orientation, shape and size, for one or more objects. This is usually done after the selection process. Common methods include direct hand manipulation, proxy and 3D tools. Direct hand manipulation is similar to how we manipulate objects in the real-world. It is easier to understand, more efficient and immersive than other patterns. Its limitation is also similar to the real-world, which is the physical reach of the user. The go-go technique shown in Fig. 2.5 can also be used to manipulate objects, in addition to selection. Alternatively, a proxy is a local object that can be employed to manipulate a remote object. Examples are tracked physical props, which are real-world objects that are mapped to one or more objects in the virtual world that the user can directly manipulate. 3D tools enable users to manipulate virtual tools in their hands, which in turn manipulate the target object. For example, handheld tools like screwdrivers, paintbrushes, and jigs or guide objects such as grid and rulers, are easily understandable to the user as they correlate with real-world counterparts.

2.3.3 Viewpoint Control

Controlling the viewpoint refers to moving, rotating or scaling user perspectives in the virtual world. For example, travel or locomotion is a form of viewpoint control. Typical methods for viewpoint control consist of walking, steering, 3D multi-touch, and automated motion.

As the name implies, walking requires the user to use his feet for controlling the viewpoint. This includes real walking motion that maps the physical and virtual world 1:1, redirected walking that allows for covering larger distances in VR than physical space allows, or a treadmill-based system where the user walks in place on a specialized device [Cakmak at al. 2014], as shown in Fig.2.6.

Steering allows control of the viewpoint direction while the virtual avatar is moving. There are many patterns, such as moving by leaning, gazed-directed steering, one and two hands flying, analog stick steering similar to typical first person video game controls, and the use of flight-sticks or steering wheels in both the physical and virtual world. However, these patterns can easily lead to simulation sickness because variations in acceleration and rotation in the virtual world may lead to disorientation.



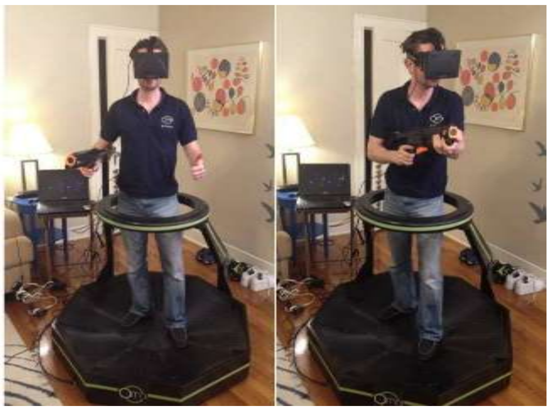


Fig.2.6 walking patterns: redirected walking (left) [Steinicke 2009] and VR treadmill (right) [Benson 2013]

In 3D multi-touch systems, the user uses one or both hands to modify the position, orientation, and scale of the world viewpoint by grabbing on to and moving empty spaces in order to walk or rotate. This pattern can be challenging to implement as small differences in sensitivity can greatly affect the usability of the system, and it generally requires some level of learning curve for the user.

Another viewpoint control method is automated motion, where the user is seated in a computer-controlled moving vehicle, sidewalk or elevator. This pattern can also induce disorientation and simulation sickness if not implemented well. It can be improved by keeping the speed of travel and the direction constant, also known as minimal acceleration, providing stabilizing visual cues such as car interior or cockpit console for spaceship, and creating leading indicators or arrows to let users anticipate upcoming motion. Other forms of automated patterns include target-based travel [Bowman et al. 1998] and route planning [Bowman et al. 1999] that let users select the target location then passively move him to that location along a defined path. Teleportation, illustrated in Fig.2.7, relocates the user to a new location without any perceptible motion at all. This pattern helps the most with reducing simulation sickness, but is disorienting for the user [Bowman et al. 1997] and requires some time to figure out the new orientation immediately following the teleport action.

2.3.4 Indirect Control Patterns

Indirect control patterns provide an intermediary to modify objects, environment or the system. It differs from the earlier patterns in the way that it only asks user to specify what action needs to be accomplished or to give commands. The system will automatically do the rest, as supposed to giving full control on what and how it should be done.

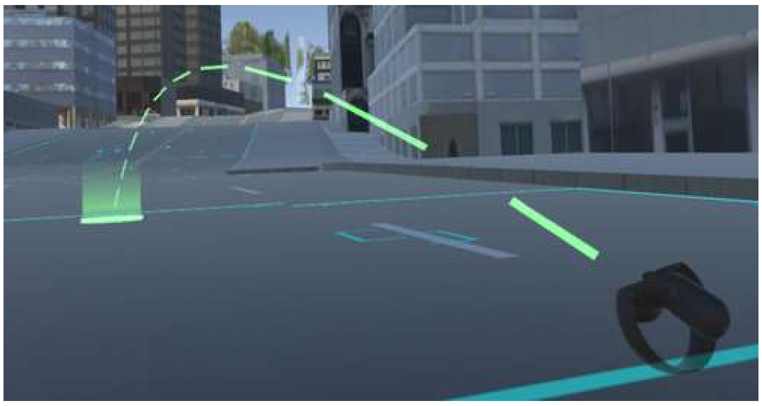


Fig.2.7 Teleportation [Lipp 2017]

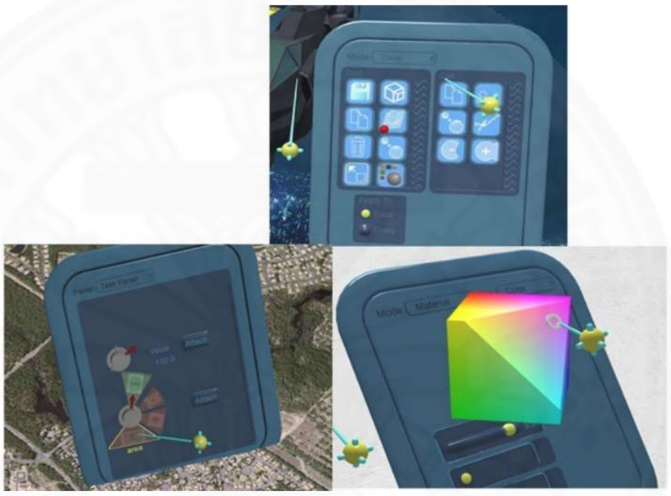


Fig.2.8 Widget and panels [Jerald (2) 2016]

Widgets and panels are the most common forms of indirect control in VR and work similarly to typical desktop widget and windows. Widgets is the geometric UI elements, consisting of labels, buttons, sliders and dials, and panels are containers that can house multiple widgets. Examples include ring or pie menus and virtual hand-held panels, as shown in Fig.2.8. They are usually activated via pointing pattern but can also be used with other selection methods. It can provide more accuracy than directly manipulating objects, however, it also requires longer time to learn and familiarize with. Physical panels can also be used as objects that are tracked to manipulate objects in the virtual world. It can provide fast and accurate manipulation [Linderman et al. 1999] but could also cause problems such as arms fatigue or misplacement.

Lastly, non-spatial control such as speech or gesture can be useful when there are a small number of options to control. However, the accuracy of these systems are low because gesture and voice accents can vary significantly between users, or even for a single user. Gesture control in particular can cause physical fatigue over long usage period.

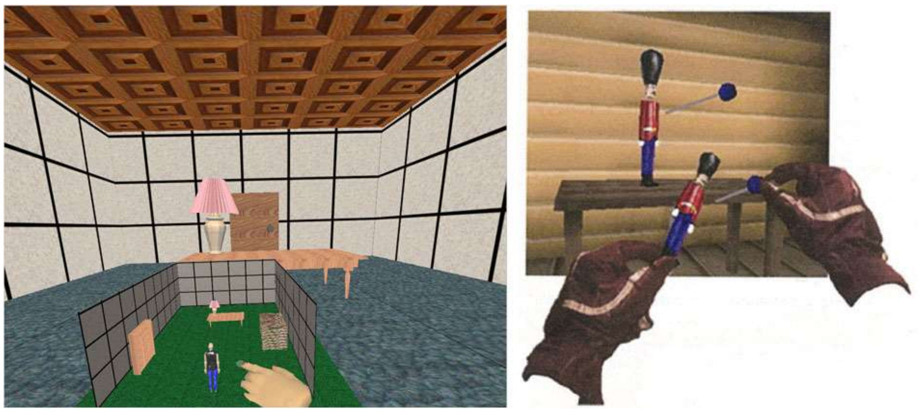


Fig.2.9 World-in-miniature (left) and voodoo doll (right) [Billinghurst 2017]

2.3.5 Compound Patterns

Compound patterns combine two or more interaction patterns together. An example is the pointing hand method in which faraway objects are first selected by pointing to them and then can be manipulated as if they were held in the hands of the user, thus combining the pointing and direct hand manipulation patterns. HOMER [Bowman et al. 1997] is one such example. It works by moving the virtual hand of the user to the location of the target object after selection by pointing, allowing the user to directly manipulate the object from long distances. Another method is world-in-miniature (WIM) that uses a mini-map to represent the whole virtual space that the user can hold and manipulate in the hand. When the user moves an object or even his own avatar in WIM, the actual object also moves in the virtual space. An example is the voodoo doll technique [Pierce et al. 1999] or Viewbox [Mlyniec et al. 2011] that combine image-plane selection, volume-based selection and 3D multi-touch. This is illustrated in Fig.2.9.

Multimodal uses different input modes, such as speech, gesture and pointing, together. An example is the put-that-there interface [Bolt 1980] that uses pointing to select (that and there) and voice control (put). Another example is automatic mode switching techniques, such as allowing the application to switch between image-plane or pointing selection automatically depending on the location of the hand in the user's field-of-view.

2.4 Software Development Environment

In this study, we use Unreal Engine 4 (UE4) as a tool to implement our proposed framework. UE4 is a game engine and software development environment designed to build video games [Wikipedia (1) 2018] and was developed by Epic Games [Epic Games 2018]. Written in C++, it consists of a rendering engine, a physics engine, sound, animation, networking, scripting, and more. UE4 is used to create many PC and console games, AR and VR projects, as well as many enterprise projects, such as automotive and architectural visualization, film, science, and education. The source code is freely available for both commercial and non-commercial usage [GitHub 2018] but requires free account with Epic Games.

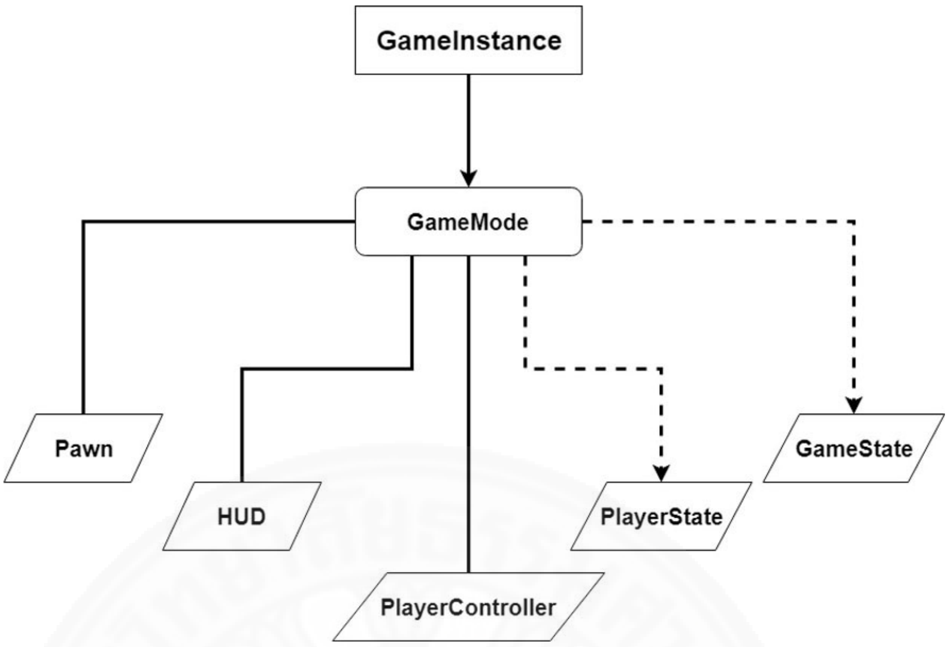


Fig.2.10 Unreal Engine framework

2.4.1 Framework Overview

To understand the general framework of UE, consider the overview shown in Fig. 2.10. When starting the engine or launching the standalone application, UE4 will create and initialize a **GameInstance** class that stores global data for that entire session. It is created at the start and deleted only when closing the application, while other game framework classes are created at the level start, and deleted after the user travel to another level. **GameMode** stores the definition of the application, including rules, user default classes and win/lose or terminating conditions. **GameMode** has events to manage when user(s) connect or disconnect, and add users to the world. Every level needs to be connected to a **GameMode** but not necessarily a unique one.

Pawn is a visual and physical representation of what the user is controlling. This can be a character, vehicle, or anything that represents the user in the virtual world. A common subclass is **Character** that has logic to fine-tune the movement of the user. **PlayerController** is the primary class that receives input from the user. This class has no visual representation in the 3D world, but will possess a **Pawn** instance. During the runtime, a user may possess many different **Pawns** while the **PlayerController** remain the same for the whole duration of that level. Generic actions such as opening GUI menus should be implement in **PlayerController** and not in **Pawn** classes. **PlayerController** also contain **PlayerCameraManager**, which is a a class that handles the camera views.

Object is the base class of all the Unreal Engine classes, shown in Fig. 2.11. **Actor** can be spawn in the level. It can have graphical representation as a 3D model, and therefore have draw-calls overhead. It is the most used class, the base for any of the object in the level, including users, doors, walls, and any other gameplay objects. Classes such as **GameMode** are also actors.

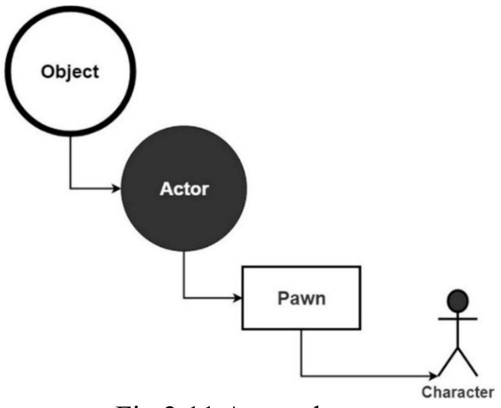


Fig.2.11 Actor classes

ActorComponent handle one particular task inside an **Actor**. A subclass of this component is called **SceneComponent** and is a base for anything with transformation, such as position, rotation and scale, in the world. **Components** are often created in the constructor of the **Actors**, but also can be created and destroy at runtime via Blueprint script. Together with **Actor**, these **Components** are the building block of any UE4 game or application. Custom **Components** can be create to do specific things during runtime. We also heavily utilized the **Components** system in creating most of the functionality of our prototype.

2.4.2 Blueprint Visual Scripting System

Blueprint is a gameplay scripting system using node-based interface to create gameplay elements from within the Unreal Editor. It is used to define object-oriented classes or objects in the engine. Programmers can also extend build-in Blueprint functionality by creating custom Blueprint nodes via C++. Each node of Blueprint, shown in Fig. 2.12, represents events, functions, variables, structs, etc. and connected together with color-coded wires, with each color corresponding to each data type.

The Blueprint system can be considered as another programming language that exists only within the Unreal Editor. It works like any other object-oriented programming (OOP) language. Any programmer who understands OOP concepts should not have problems picking up and utilizing the Blueprint system in a relatively short amount of time. Since it is a language that runs on a virtual machine, similar to Java, programmers do not need to wait for it to compile and restart the engine, therefore it is faster to test the code and iterate via Blueprint than in C++, making it an ideal tool for implementing our proposed framework.

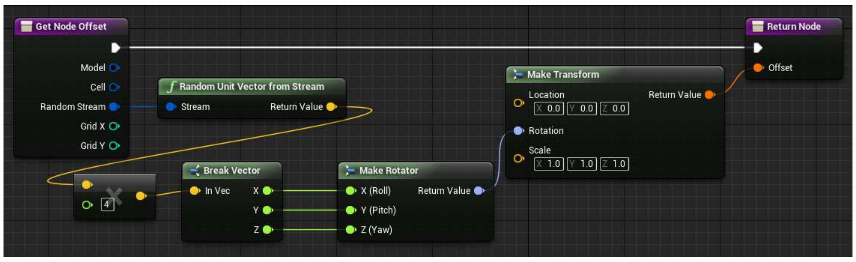


Fig.2.12 Example of a Blueprint function

Chapter 3
Real-Time Pictorial Compositional Tool in VR

3.1 Cauldron: Design and Concept

The purpose of this research is to build an immersive VR application called Cauldron that lets the target user (visual artists), quickly visualize and prototype pictorial design ideas in virtual environments. The goal is to improve the efficiency of the art creation workflow by letting the user transfer their imagination, i.e. the images that they see in their head, into a more concrete form, such as scenes in the virtual world or actual digital images. This can then be used as references or shared among colleagues and staffs to make plans and creative decisions, in a manner that is easier than trying to explain by verbal communications or other non-visual means.

3.1.1 Design Considerations

Cauldron is designed for and intended to be used by visual artists, who may or may not be proficient in computer simulation, games, or 3D content creation. The following are design considerations that aim to give the smoothest user experience, while still providing a useful set of tools that can improve the user's day-to-day workflow in a meaningful way.

Approximate tasks that artists perform in real-life situations

The system needs to be able to perform tasks such as placing and orienting various kinds of object in the scenes, allow for different lighting setups and different time of day, pose and move virtual characters around the environment. We also created a virtual camera that let user take and printout screenshots for use as references outside of the system.

Experiment with multiple design variations

The system should allow the artist to quickly go through various design variations with minimal loading time or interrupts, so that he can continue to work without breaking the creative flow due to technical limitations. The artist can compare between different versions of the design and make decisions easier, without ever leaving the system.

Quick iteration time

The system is designed to be quickly used to try out new ideas, without much complexity and/or elaborate setup. The goal is to be comparable to how artists traditionally use sketchbooks to draw or write down a few sketches. Each idea is represented by one saved file in our system and can later be worked on or discarded easily.

Provide real-time high quality visual feedback

The system requires a high level of interaction while also providing real-time visual feedback to the user. The graphical quality is also considered one of the requirements, because we want a system that is visually appealing to artists, and more importantly, makes them see that they can immediately use the system in day-to-day work, not just as an interesting research project.

Intuitive control

The user experience overall is designed for artists. The user does not need to be an expert in 3D applications or an engineer to use. The number of buttons used on the controller will be kept to a minimum, with an interaction interface that mimics or is comparable to real-life interaction that is simple to understand and use, without the need of supervision.

To meet these design requirements, we feel that using a game engine is most suitable for our purposes, because modern game engines and graphics hardware are very powerful, capable of outputting the level of realism that is adequate for this application. The aim is not to try and create an accurate physics simulation, but rather to approximate it in order to give an idea of how things would look in real-life, while providing the rich level of interaction required for our system. In addition, all of this needs to happen in real-time to let the user feel as natural while using the system as much as possible.

VR is also a major component of the system, because when we try to approximate or recreate what the artist needs to see in real photoshoot situations, depth is an important piece of information that usually gets neglected when designing in a 2D medium, such as paper or PC monitor screens. With the immersion that VR headset brings, along with 1:1 tracked hands and head movement, users can literally be inside their own design. The ability to judge depth and get the sense of scale immediately while designing the image is an important information that can greatly improve the way artists work.

3.1.2 Application Scenario

Fine Art, Illustration, Manga and Comics

Especially for artists that compose images with multiple figures, or design scenes that involve very complex lighting situations, such as shown in Fig. 3.1. The system can be used for design placement of multiple figures, obtain lighting information for anatomical references, solve perspective and scale directly in VR. Also, besides using Cauldron as a planning tool, the artist can use the output from the program as references for creating artwork directly.

Film Pre-Visualization

For storyboard artists, cinematographers, and directors to try out different camera angles or lighting setup without the need to spend time on creating pre-visualization (pre-vis) inside 3D programs or run a quick test in VR first, before setting out to create pre-vis.

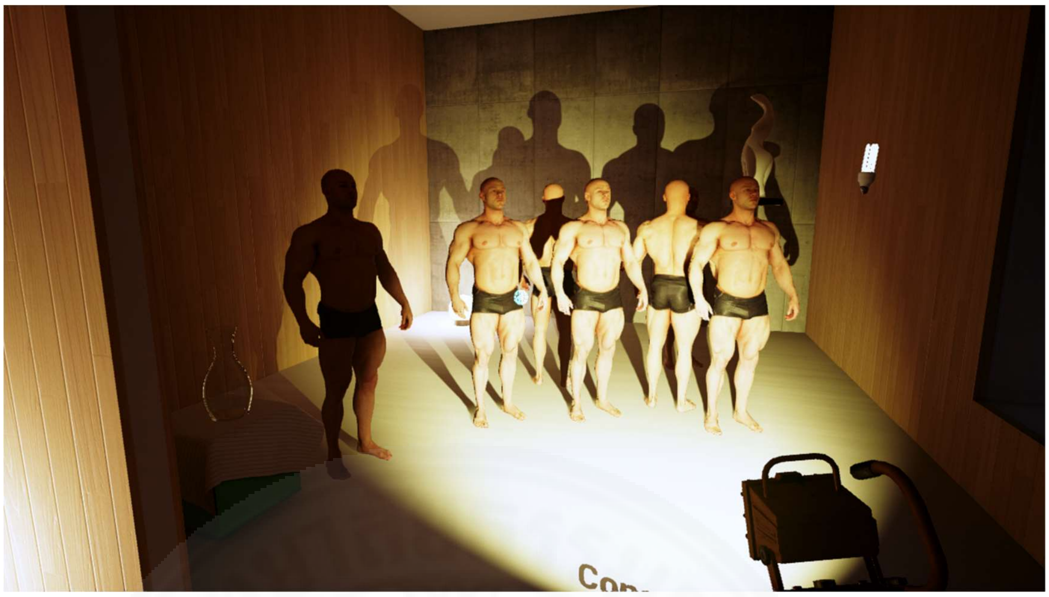


Fig.3.1 Example of complex design with multiple figures

Photoshoot Planning and Advertisement

Photoshoots involve many equipment and often the artist needs to work with make-up artist, stylist, as well as assistants. To get the idea or design across so that everybody is on the same page can be difficult. Planning in Cauldron first can help solve potential technical issues and reduce the time to experiment while on the set, making sure that everybody on the team understands your vision, which can result in improving overall productivity.

3.1.3 System Design

The VR device used in this study is one of three commercially available system called Oculus Rift, shown in Fig. 3.2. The HMD uses a Pentile OLED [OLED-Info 2018], 1080×1200 (per eye) resolution, 90 Hz display, connected to a personal computer via HDMI and USB 3.0 port. It has built-in headphones that provide 3D audio effects, though this feature is not used in this study, as well as position and rotation tracking. The tracking system is performed by two USB 3.0 infrared sensors that pick up the lights emitted by the HMD's integrated IR LEDs. The tracking sensors are placed on a desk and the 3D space created by overlapping both sensor's cone of view is the playable area, where the user can walk, stand, sit or jump. The main input device is the hand-held controllers called Oculus Touch. The controllers are held in each hand, and each consists of one analog stick, three buttons and two trigger buttons, with a mechanism to detect the gesture of the user's fingers while holding it. The position and rotation of the controllers are tracked by the same USB IR sensors. [Wikipedia (2) 2018]

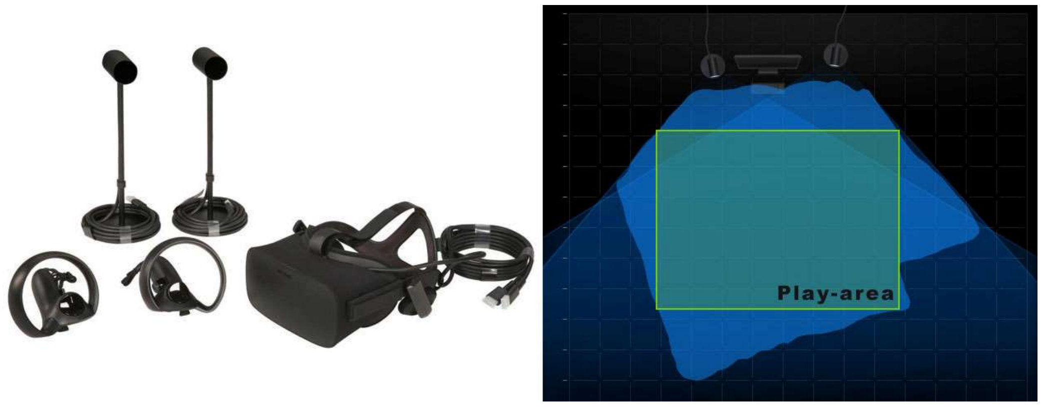


Fig.3.2 VR device platform: Oculus Rift (left) and setup area (right) [Newegg 2018]

All processing will be done on the connected computer. The recommended system requirements provided by Oculus [Oculus (3) 2018] are: Intel Core i5-4590 or above, at least 16 GB of RAM, at least Nvidia GeForce GTX 970 graphics card, an HDMI 1.3 output, three USB 3.0 ports, and one USB 2.0 port, 64-bit Windows 7 SP1 or later. This system requirements is considered mid-to-high end at the time of this study. However, because the same system will also be running Unreal Engine and other SDKs that will be used to create the VR environment, it is our recommendation to have significant head room for handling the extra processing power those SDKs require, especially in terms of system memory.

The basic idea for Cauldron is an immersive virtual sandbox where artists can freely experiment with various design ideas in VR. Starting by selection of the scene or level within the program, the user can then quickly add a predefined set of objects, such as characters, furniture and lightings that are provided in the content library, into the scene, move and change them around, do basic post-processing until the user is satisfied with how the scene looks, then either save it to work on later, or export image files employing a simulated camera system for use as references. This process flow is shown in Fig. 3.3.

The prototype system of Cauldron is implemented using UE4, which is capable of providing high quality real-time renders and physics simulation, as well as interfacing directly with the VR hardware. The large online community of UE4 developers signify many available resources and tutorials. Oculus Rift is used to display the VR environment, though the user can also use standard 2D monitors as well. For input devices, the current prototype system supports standard keyboard and mouse, Xbox gamepad, as well as Oculus Touch for VR.

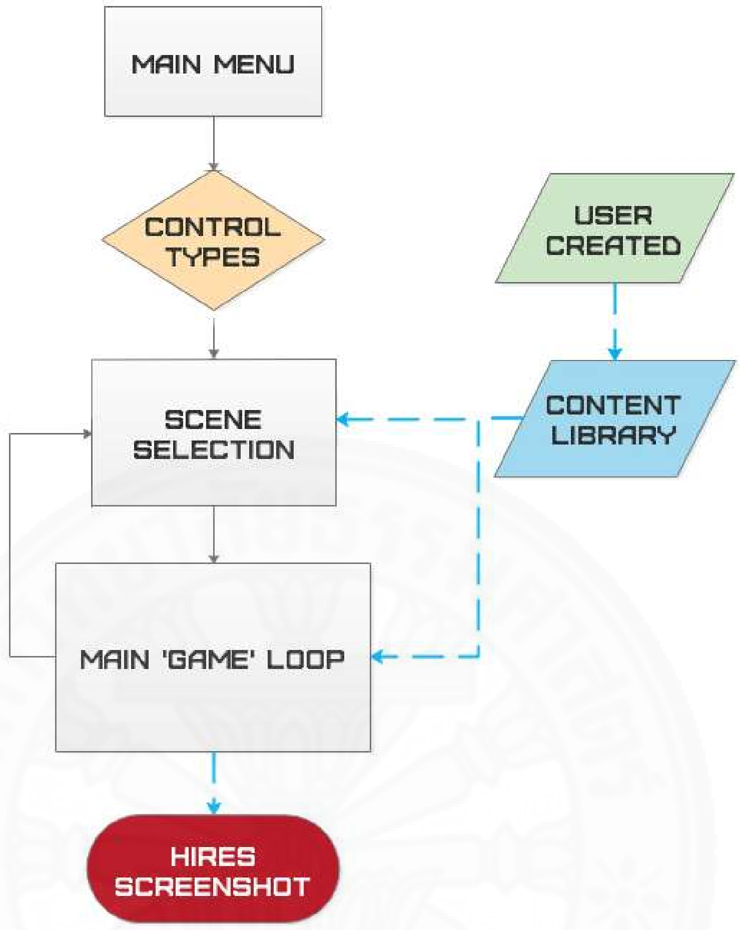


Fig. 3.3 Program flow

To use Cauldron, the user starts by creating a profile and specifying the interaction device. Current choices include a monitor with keyboard and mouse (**Monitor**), head-mounted display with gamepad (**HMD**) or HMD with tracked motion controller (**VR**). Once the profile is created, it will be automatically loaded the next time the user runs the program. The user can change the name and input devices by selecting **Profile** from the main menu, as shown in Fig. 3.4. However, in the current prototype of Cauldron, the profile is tied to the name of the user, meaning if the user changes the name, all the progress and saved files will no longer be associated with the new profile. This feature was chosen for simple implementation, but will be extended to allow for multiple users in future versions.

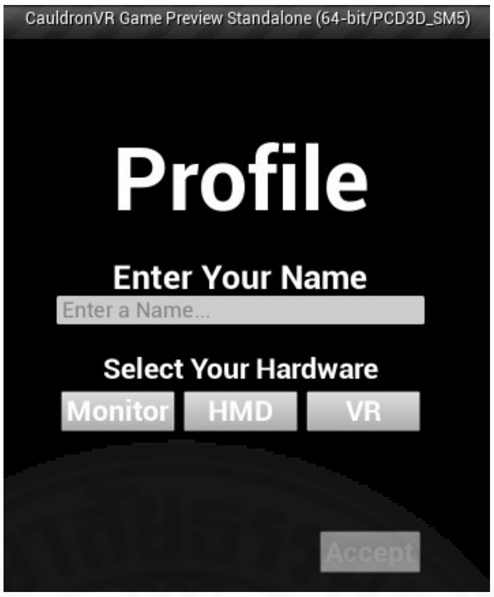


Fig. 3.4 Profile creation menu

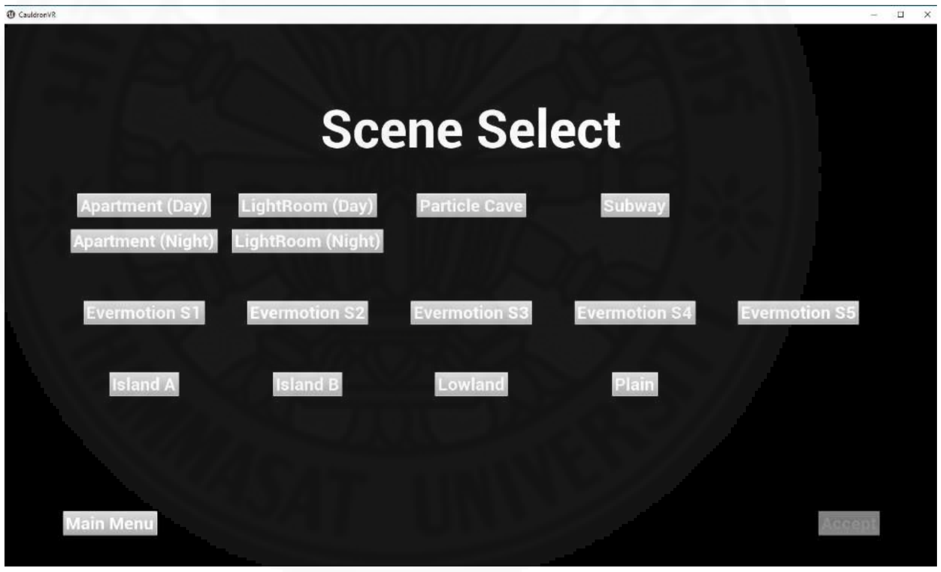


Fig. 3.5 Scene selection menu

After the profile is created or selected, the user then chooses a scene or virtual environment as a background for the design, as illustrated in Fig. 3.5. If using an HMD, the user can start to wear the device at this point. When the scene is fully loaded, the user can then design by directly manipulating objects or open menus to add more objects into the scene, change post-processing settings, as well as save and load existing designs from the hard drive. Once the user is satisfied with a design, the final output can be exported as image file(s).

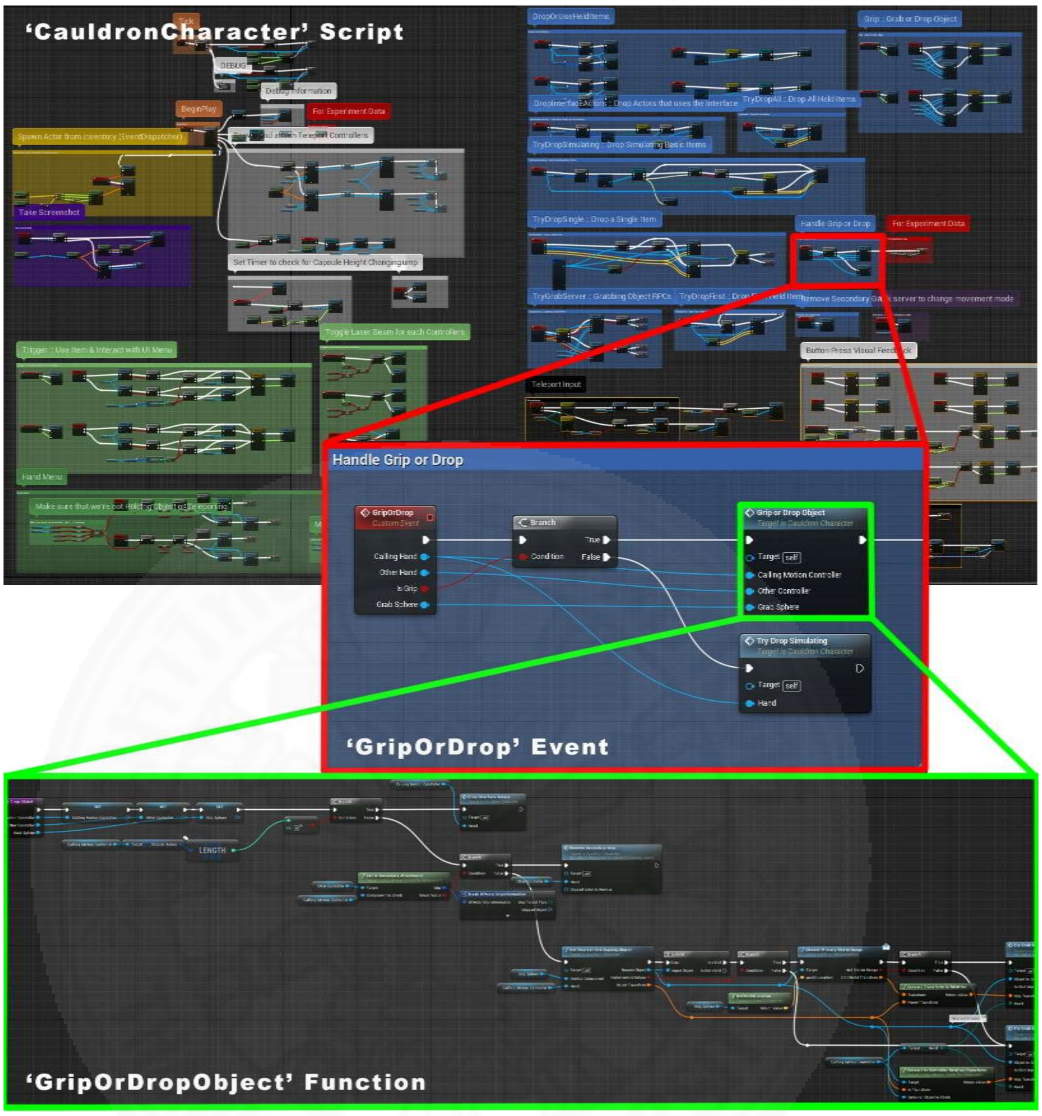


Fig. 3.6 Script for player-controlled pawn

3.2 Implementation

Using UE4 terminology, we implement the pawn or user-controlled character, called **CauldronCharacter**, which consists of many subsystems. Each of the subsystem, represented by a colored-block group in the code, will only interact with the corresponding classes of objects or menus within the scene. This way, we can add/delete and test different types of objects without modifying other unrelated systems. We can also have multiple types of pawns to suit different input and display devices, or a specialized pawn type that can interact with limited classes of object later, such as a guest pawn that can only walk and look around, in case we want to extend the system to support a multi-user environment in the future.

The most basic interactions that the user needs to perform are move, rotate and use (ex. press button, turn switch on/off). This is handled inside the **World Interactor** subsystem, shown as blue-colored blocks in Fig. 3.6, that can manipulate all physical objects within the scene, as well as different types of lights and particle effects. **World Interactor** works by first creating an invisible sphere of a specific size at the user's hand location. This is visualized by the light-green sphere in Fig. 3.8. A collision check is performed to determine if any object intersects with the sphere, as well as the type of the object. If the object is also a **World Interactor** then it belongs in the same collision channel and is valid. This calls the event **GripOrDrop**, where the **World Interactor** would grip the new object or drop an existing object. Gripping is performed by attaching the object to the hand by copying the position and rotation values from the user hand, along with turning off the objects physics simulation and collision while setting friction to 0. Dropping removes the object from the parent (user's hand) by maintaining the position and rotation from the last rendered frame, turning on physics simulation, collision, linear and angular damping. In the case that the object is used instead, it will call the corresponding event. For example, using lights will call

```
//Light ON/OFF
OnUse(){
	LightSwitch(bEnable);
}
```

as shown in Fig. 3.7.

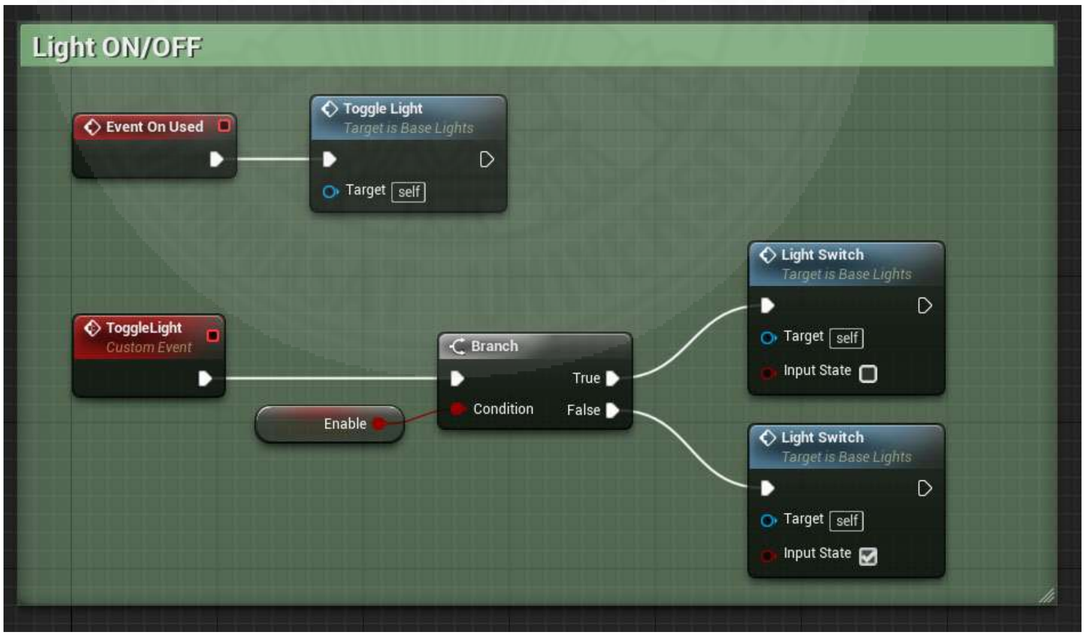


Fig. 3.7 Example of **OnUse()** function inside a spotlight class

In the case of mouse and keyboard or gamepad setup, regardless of display devices, this type of pawn can have only one **World Interactor** system attached because of the degree-of-freedom (DOF) limitation. The user needs to be directly looking at the object to place the interactor cursor on it, then press a button to grab or use. However, for VR setups with tracked controllers, we can have one interactor per hand, attaching directly to the position of each hand and allowing the user to interact with both hands independently. Two interaction schemes are implemented for comparison. The first method is a variation of HOMER technique [Bowman et al. 1997] with the **World Interactor** locked in the middle of the screen, representing the hand. The user needs to press a button to grab an object, simultaneously with other modifier buttons to move or rotate it. A second method places two visible hands locked in front of the camera (user's head) at a fixed distance. The user needs to turn his head in order to rotate an object, but it allows for user to walk while holding the object and grab two objects simultaneously. Both methods are shown in Fig. 3.8.

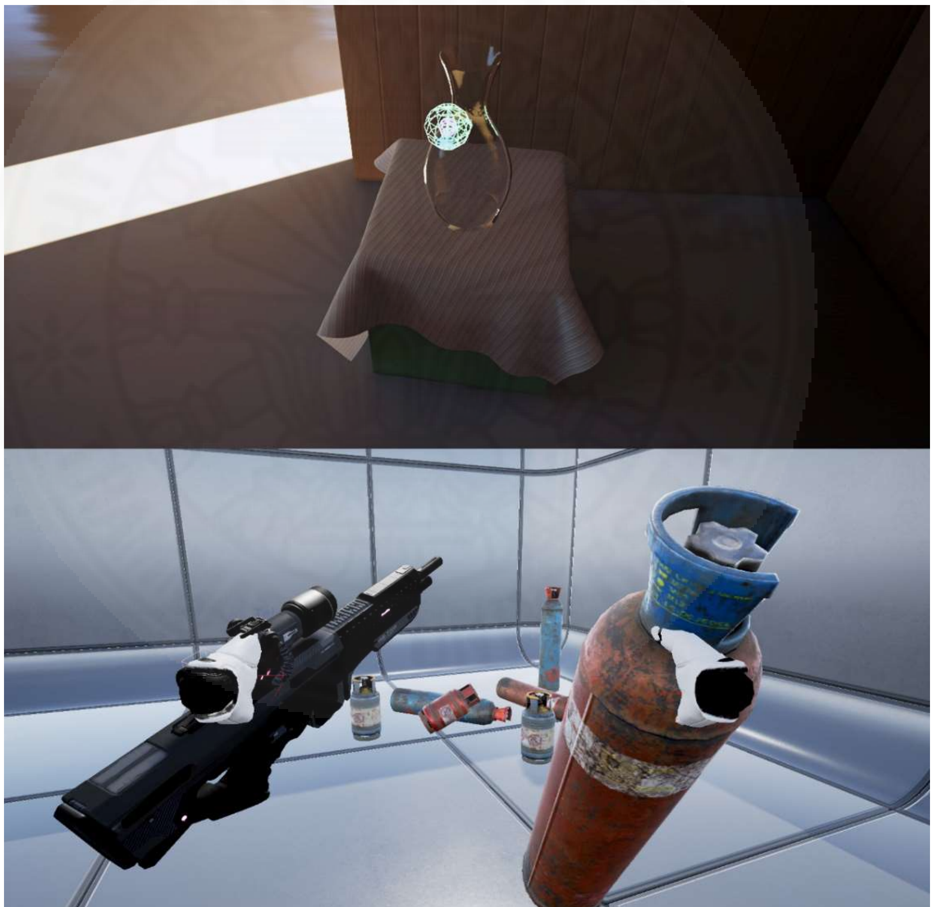


Fig.3.8 Interaction schemes used for testing: a version of HOMER method (top) and using two visible hands that are attached directly to the camera (bottom)

A **Mannequin Interactor** is used to pose and move mannequins, as shown in Fig. 3.9, with the ability to show or hide the control points as needed. The control points are made of basic geometric shapes that are also derived from **World Interactor** objects, and used to move the position and rotation of the corresponding bone of the character via the Blueprint script in shown Fig. 3.10. The user can grab and move it, in order to pose the character as they desire.

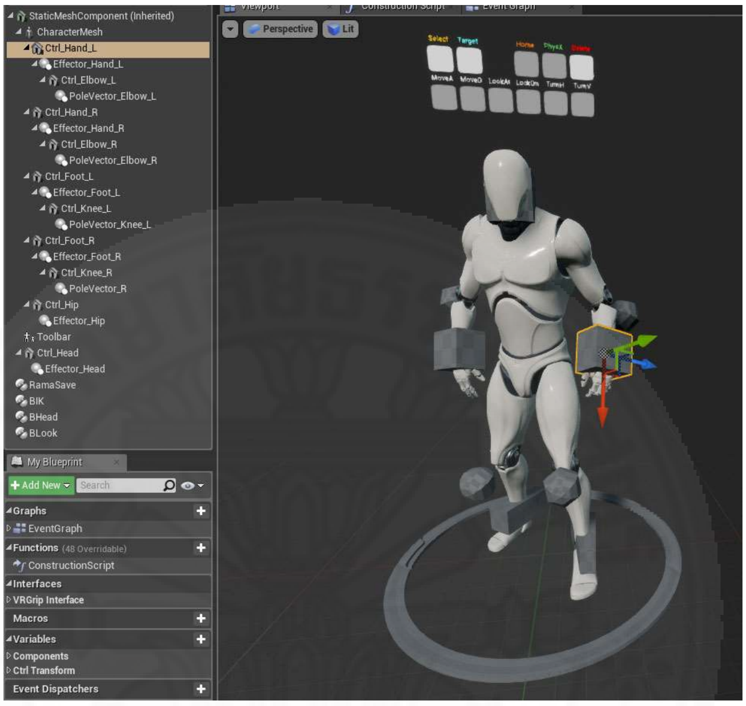


Fig. 3.9 Mannequin and Control Points setup

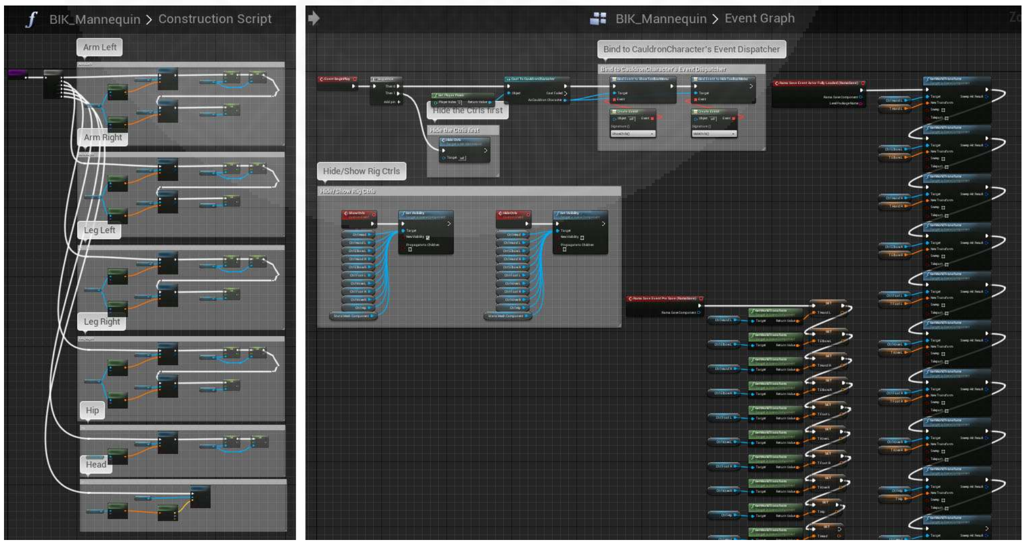

BIK_Mannequin > Construction Script
Arm Left
Arm Right
Leg Left
Leg Right
Hip
Head
BIK_Mannequin > Event Graph
Bind to CauldronCharacter's Event Dispatcher
Hide the Ctrls first
Hide/Show Rig Ctrls

Fig. 3.10 Control points are bonded to the mannequin bones at and during runtime

Other subsystems, such as a **Widget Interactor,** are used for interacting with all 3D GUI menus within the world, represented by green colored-block groups in Fig.3.6. **Camera** represents the viewpoint of the user. In case of 2D monitors, the viewpoint will be controlled by the user manually, like in conventional video game control schemes. For VR HMDs, the camera will be controlled by the user's head movement, shown as grey colored-block groups. When adding more subsystems to the pawn, we can first implement it as a separate subsystem on an empty pawn, allowing us to test and optimize it without risk of breaking other working subsystems.

We provide a template class for each actor type, which is a UE term referring to any object that can be placed within the world, whether it is visible to the user or not. These templates are associated with the pawn subsystem. For example, we create all the objects and lights from the same parent class (**Objects**) shown in Fig.3.11, which can interact with the **World Interactor**. All the objects and lights in the scene can be grabbed and used by any pawn through the **World Interactor**. The green colored-blocks in Fig. 3.11 correspond to the **World Interactor** of the user-controlled pawn and consist of scripts to turn on/off the object's physics and other related properties when being grabbed or dropped. The purple and grey blocks are for interaction schemes that will be discussed in the next chapter.

Two different categories of objects and lights are defined by the system: physical objects that behave like real-world objects with physics simulation; and design objects that can be placed by the user primarily to realize the design, without real-world constraints such as physics or collision detection. In other words, the design object is not simulating physics, has no weight and has excessively large friction values to allow suspension in mid-air when dropped from the user's hand.

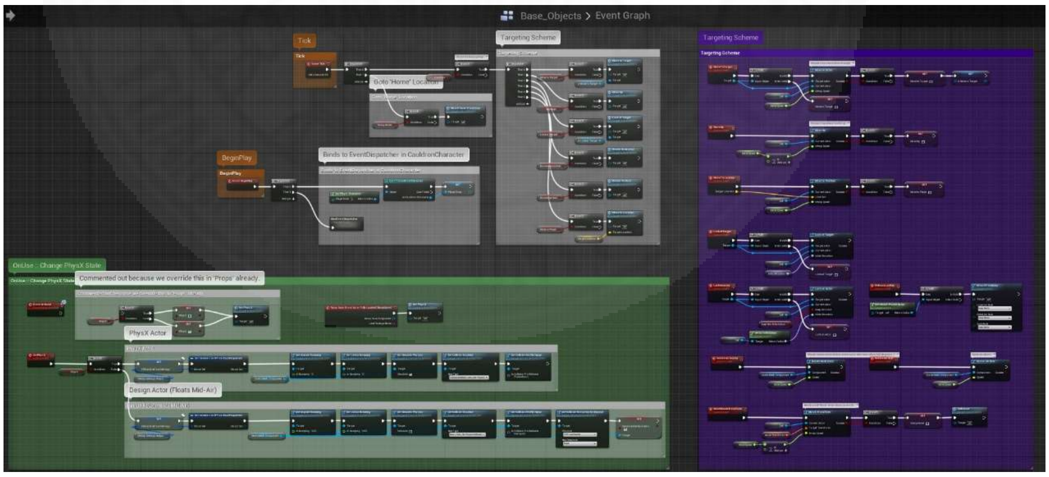

Base_Objects > Event Graph
Tick
Targeting Scheme
Sets "Home" Location
BeginPlay
Binds to EventDispatcher in CauldronCharacter
Targeting Scheme
OnUse : Change PhysX State
Commented out because we override this in 'Props' already.
PhysX Actor
Design Actor (Floats Mid-Air)

Fig. 3.11 Template for all of Cauldron's 3D object type

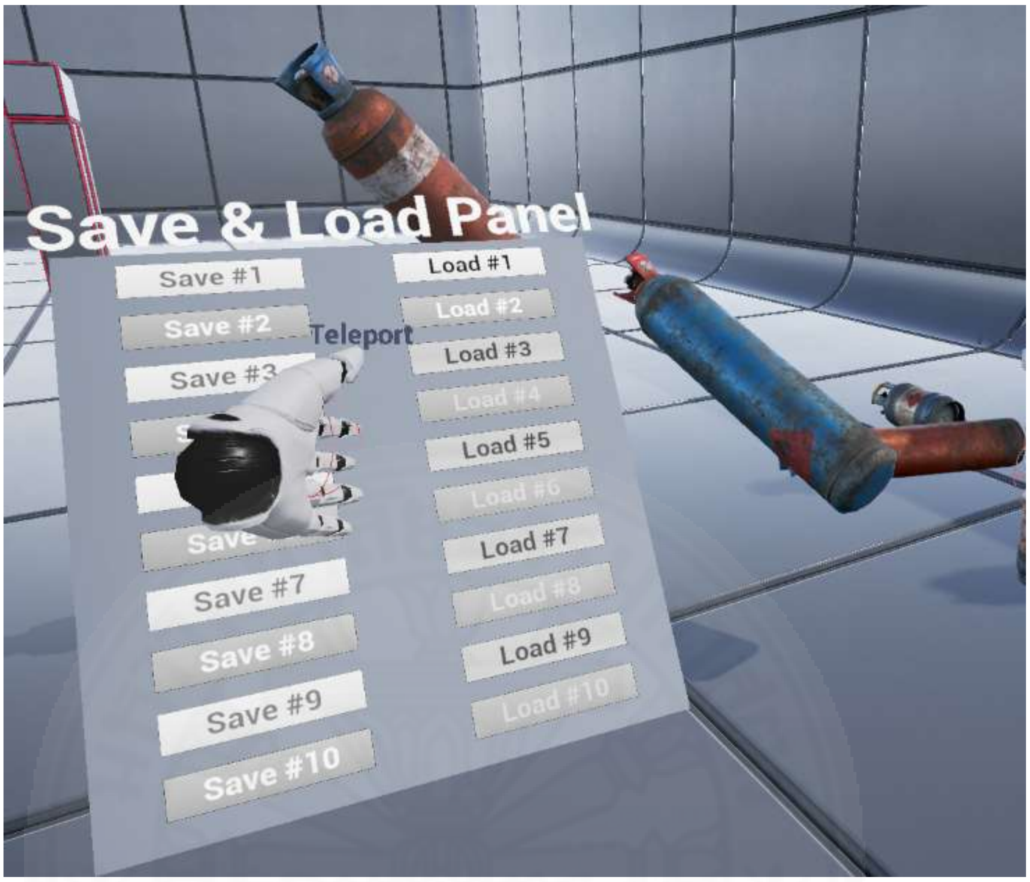


Fig.3.12 Screenshot showing design objects and 3D UI

The **Save and Load** system is implemented differently from typical games or VR applications. The way it works is similar to a transparency slide or tracing paper. The scene or background is considered as a static image or photograph, with a transparency slide or tracing paper placed on top. The user will be conducting the design process on the slide, and not on the scene itself. Therefore, it is only necessary to save the list of user-added actors that are currently present on the scene, its state (ex. on/off, in the case of lights) and its transformation (position and rotation). The saved file is very small, allow us to perform save and load operations instantly, while the user still has full control of the pawn within the scene and in VR. This allows the user to toggle between different save files the same way we can turn the page of a book, or scroll an image back and forth on the desktop to quickly make comparison and artistic decisions. The saved files itself, in the current prototype, is tied to the level. Each level can have up to 10 different slots of saved files per user, as shown in Fig. 3.12.

3.3 Evaluation

A small-scale, informal user study is performed involving four male and two female subjects, aged from 23 to 30. The testers all have moderate to advanced background in computers and programming. Two of the male testers are experienced gamers, while the rest have very little to no video games experience. This is also the first time for all the testers to experience VR. The testers can be seen trying out the prototype system in Fig. 3.13.

We first let them become comfortable with VR by showing the demo that comes with Oculus for approximately 10-15 minutes. Each tester is then allowed to rest while another tester tries the demo. The system used in this experiment consists of Intel Core i7-920 running at 2.67 GHz with 18 GB of RAM and Nvidia GTX 970 graphics card, which is slightly below the minimum requirement suggested by Oculus in terms of the CPU requirement. Some tracking loss was observed if the users turn their heads very rapidly, but overall this was not a significant problem.

The main purpose of this evaluation is to determine whether the user can understand the concept of the application or not, and also the preference of the user regarding the way to interact with the objects in the virtual world. Specifically, it explored various interaction and locomotion methods to determine user preference, so that we can focus our development efforts on certain techniques to be used as a baseline. We were also interest to see which parts of the application that the user would struggle with, and which parts need to be improved.

Fig.3.13 Testers trying out the prototype system

The first issue that comes up is the locomotion methods. The prototype has two methods employing teleportation and joystick movement as used in video games. Testers who are experienced gamers feel teleportation is quite restrictive and prefer to move using joysticks. However, when using the joystick for movement, all testers, except for one male experienced gamer, felt simulation sickness to varying degrees. It is therefore concluded that teleportation is the preferred locomotion method, with comments from testers that we should provide more visual-aids to help orient the user before and after performing teleportation.

Another main issue is the method of interaction. At the time of testing, we did not have access to a motion controller. All the testers clearly had problems using a gamepad to control and manipulate objects in VR. Two interaction methods were explored: (a) a version of HOMER method and (b) two visible hands in front of the user, as presented in Fig. 3.8. Neither interaction methods provided satisfactory results. For gamers, method (a) is usable, but required a steep learning curve, with one tester showing some proficiency after using it a few times. However, it was practically un-usable for the rest of the testers, because they were not familiar with the gamepad enough to be able to press three or more buttons simultaneously to operate without looking at the controller. For method (b), the majority of testers can use it adequately, except for one female tester, who showed difficulty in gauging the distance between the floating hand and the object she wanted to grab, so it became frustrating after a while and she eventually gave up. One tester showed that he can use method (b) to stack several boxes in a complex manner after a few minutes, but this method also introduced fatigue much faster than method (a) because the user needs to turn and twist his head all the time, which in turn led to dizziness due to a combination of both fatigue and simulation sickness.

As far as the features and usefulness of the system, all the testers were satisfied and understood the goal of how this tool can be helpful for artists. Some testers, especially the ones who are not gamers, were amazed with the fidelity and capability of the modern game technology. One tester commented that the ability to judge the depth and sense of space while designing is very powerful and he is sold on the idea of designing in VR.

After evaluating the overall user experiences, it is concluded that support for HMD with gamepad setup should be dropped. The limitations of the gamepad result in the users constantly struggling with it and being very conscious about the controller, rather than using it transparently as a tool. A new interaction scheme to manipulate the objects in the virtual world should be designed in order minimize user fatigue without losing fidelity of interaction.

Chapter 4
Command-Based Object Manipulation in Virtual Reality

The goal of this study is to develop a new interaction technique that improves overall efficiency of design tasks in VR by reducing the amount of time used, as well as decreasing the amount of physical actions, such as pressing the controller buttons, needed to complete a task. We achieved this by developing a system to issue commands to the object(s) and let them perform that task automatically. There is no need to micromanage every step of the design process, allowing the user to see the bigger picture, and finish the task faster. This can lead to less chances of the user experiencing simulation sickness, discomfort and physical fatigue.

4.1 Command-Based Concept: the 'Assistant' Metaphor

When designing the interaction technique to be as efficient as possible, we narrowed down the tasks that an artist needs to perform over and over both during photoshoot and designing in our VR software. We found that the most common tasks involves moving object(s) either from point A to point B or setting one object to follow another object, such as a lighting setup that follows the main actor. In real life, especially in big budget production, there are people specifically assigned to do these tasks so that the photographers/directors do not need to micromanage all the props themselves. We want to recreate this in our software as well, essentially build an assistant that receives our order and execute it repeatedly, without the need to micromanage, until we say otherwise. The set of commands that will be the most useful and currently implemented in the software are: move to (target), move here (to user's position), follow (target), lock-on (follow and orient itself in relation to target), look-at target (orient itself to target but does not move), as well as some helper commands such as select all (objects), deselect and delete.

Since we did not want to go with artificial intelligence, which could add layers of complexity and overhead, we used math to calculate and change the transformation of each object. That means we will be giving the command to the object directly. Essentially, each and every object itself acts as our assistant.

This assistant concept is different from other interaction concepts in the sense that the user is not the active party in the action. Users only give out the initial commands, and then the system itself will determine how the object will move and when the action is completed. It is a compound between an automated control pattern and widget and panels for issuing commands, while for most other interaction schemes the user needs to be the active party and perform all of the actions himself.

4.2 Implementation

The system is divided into two parts: the object itself that needs to keep track of the current commands and target; and the execution of the command. All the codes are implemented directly in UE4 visual scripting system called Blueprint.

We execute each command as a small and modular function that can be called by any object or gameplay code. This is called a Blueprint function library in UE4. The function takes the reference to source object and target, and then returns the resulting position. The object that calls the function will handle all checks, such as completion and validity. A base class is created for all objects that will be able to use our interaction scheme, as well as extra variables that are required in order to keep track of the target and commands. From the manner in which we implemented the commands, inside each object are Boolean flags corresponding to each command, gating the block of code that will check the validity, check complete conditions or call the command function. If the object receives a command then it will set the flag to **TRUE**, and execute the command code every frame. When the object receives the stop command or reaches the complete condition, then it will set all Boolean flags to **FALSE** and set all related variables to **NULL**. We execute the code every frame in order to slowly interpolate the position of the object towards the goal, instead of immediately snapping the object to the target positions, to let user see what is happening and reduce confusion.

4.3 Usage

Every object capable of interaction has a toolbar at the top of it that can be toggle on/off, as shown in Fig. 4.1. The user can select, target or delete the object by checking the appropriate box. The darker icons in the bottom row of Fig.4.2 are for debugging purposes only and display the commands that the object is currently executing (if any).

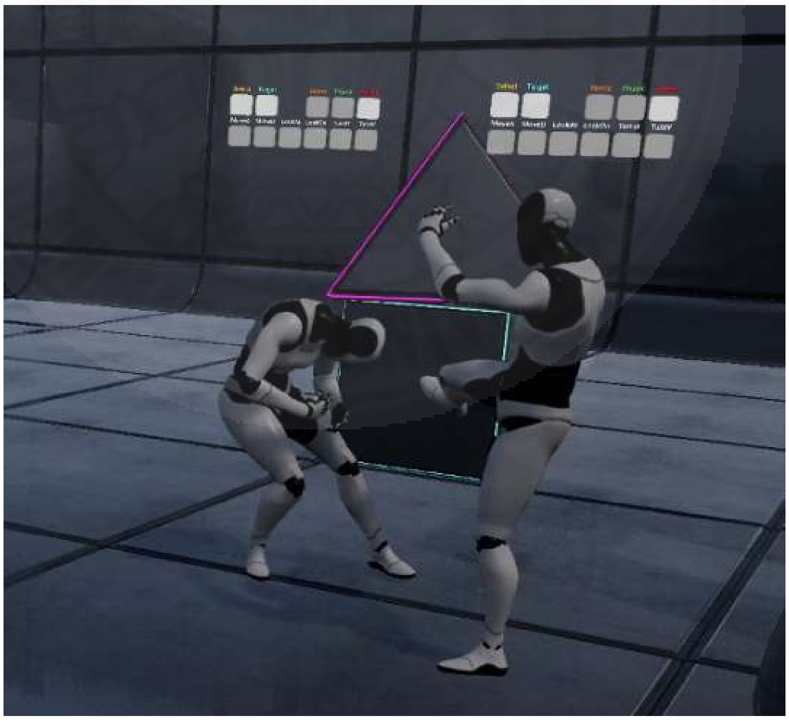


Fig.4.1 Objects with toolbar on top

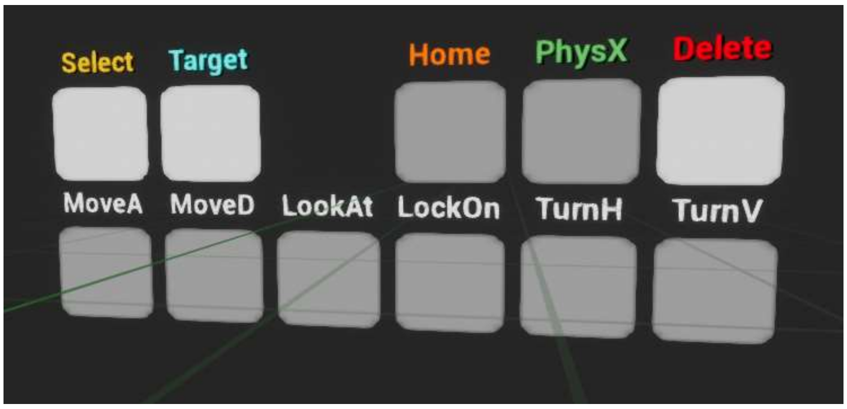


Fig.4.2 Expanded view of the toolbar

The user first needs to select the object(s) that will be receiving the commands, then select which object will be the target. Next, the user opens the hand menu panel shown in Fig. 4.3 and chooses which commands to execute. The object will slowly interpolate its location or orientation toward the target. The toolbar on top keeps displaying as long as there is some command executing, as a reminder to the user. For move commands, after it reaches the target location the object will stop and clear the command variables automatically, but still keep itself in the selected state. Other commands will keep running every frame indefinitely. The user is encouraged to press **Stop!** after completing the design element in order to free up CPU and memory resources. The workflow is shown in Fig. 4.4.

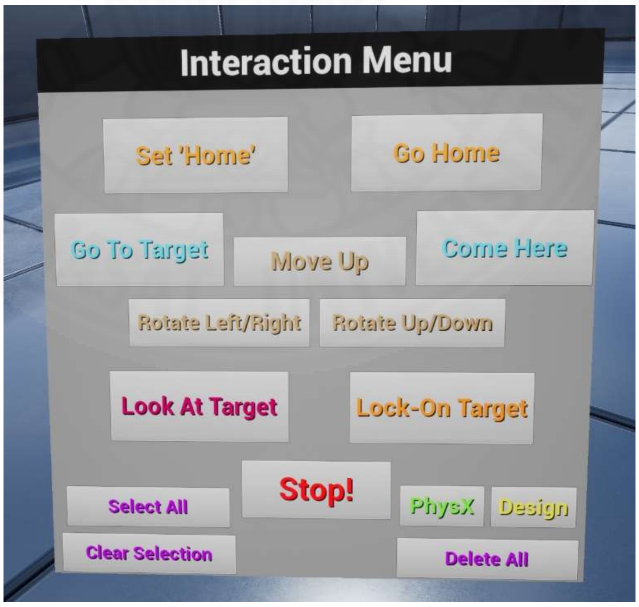


Fig.4.3 Hand menu panel to select the commands

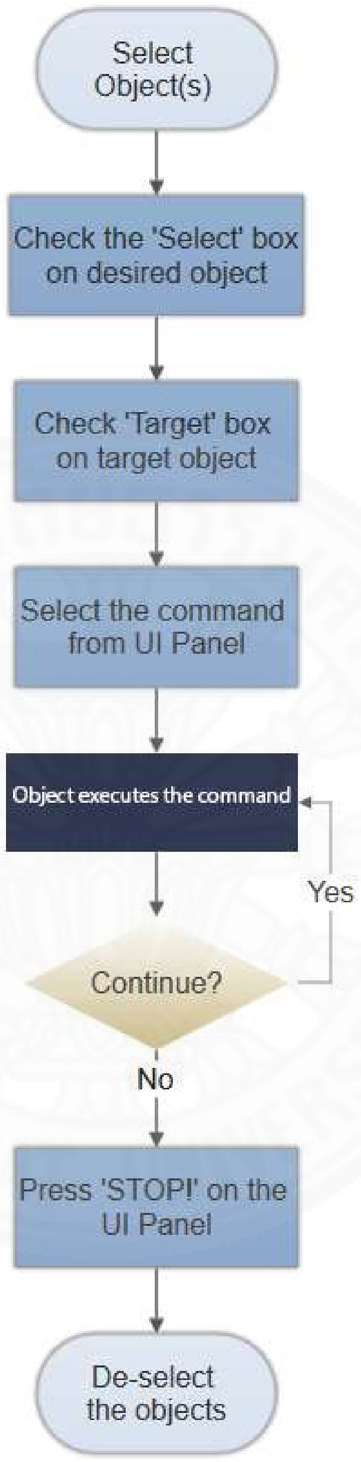

Select Object(s)
Check the 'Select' box on desired object
Check 'Target' box on target object
Select the command from UI Panel
Object executes the command
Continue?
Yes
No
Press 'STOP!' on the UI Panel
De-select the objects

Fig.4.4 The workflow for command-based technique

The system is designed to be nested up together when used. For example, consider a table with four chairs and a few books on top. The chairs and books can be set to follow the table. Therefore, we need only to move the table when we are arranging the design. The table can also be set to follow another object, such as character A, so that when we place character A in the scene, the table and all of its child objects will also be placed. This allows the user to design the scene as modular parts, then lock them together and move them as one unit easily. Also, for objects such as lighting equipment, we can set them to light our character once, make it follow the character and then move the character around freely without the need to worry about moving the lights again.

4.4 Evaluation

To evaluate our technique, we to created small empty test levels that focus only on moving multiple objects from point A to point B, as this task represents the most common action the user will perform when designing or visualizing in VR. A comparison is made with direct manipulation method, which is the most commonly used, immersive and easy to learn technique. Direct manipulation is also the closest technique to the action performed in real life, which is reaching out and grabbing the object in the hand.

4.4.1 Participants

Thirty students, consisting of 17 males and 13 females subjects, from a local university participated in the evaluation. The body of students consists of information technology (10), multimedia technology and animation (10) and fine arts students (10). The age range is from 19 to 22. This was the first VR experience for all participants. Some have extensive experience with video games while others have never played any games before.

The students from multimedia technology and animation are currently studying 3D software (Maya, 3ds Max, etc.) for digital animation purposes. Therefore, they will serve as a baseline to compare the usability and learning curve of our prototype to the tools that are commonly used in the industry today. However, fine art students represent our main target group and allow us to gauge how well non-technical users will learn and understand our prototype.

The participants were introduced to the Cauldron software, which is the program that we developed for pictorial design in VR, and took turns to use the software for around 15 to 20 minutes each to become familiarized with the controls and feature sets. Then they were allowed to rest for a brief period. After everyone got a chance to try the software, we explained the procedures and purpose of the experiment, gave a demonstration performed by the author, then let the participants take turns to complete the test.

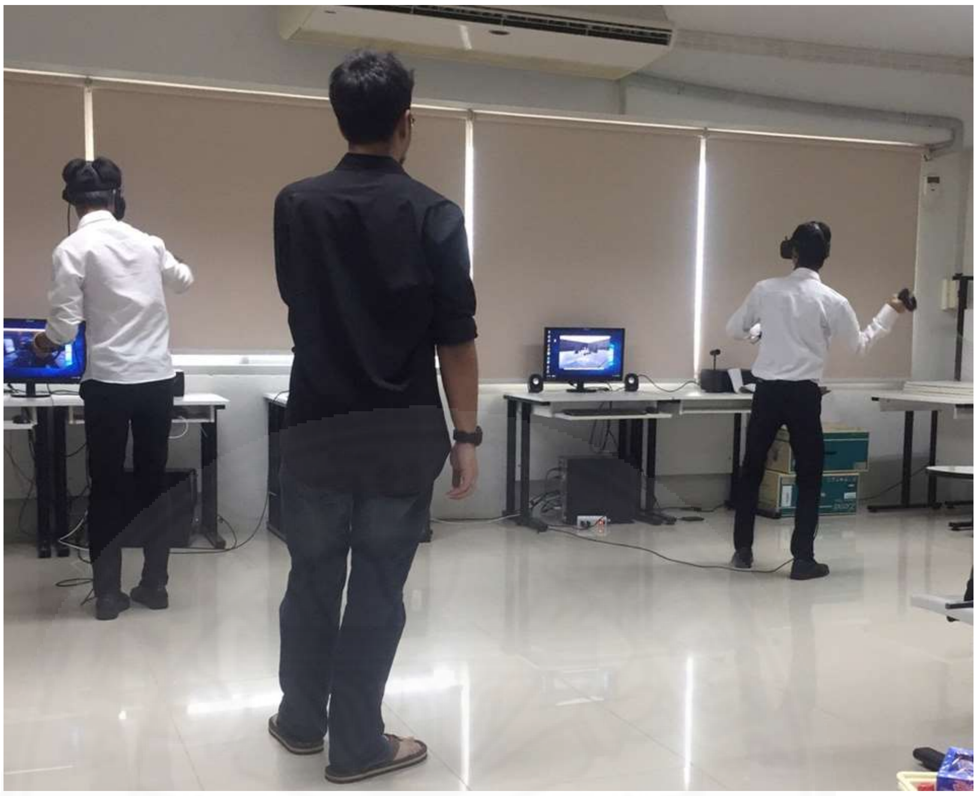


Fig.4.5 Test stations setup

4.4.2 Apparatus

We used Oculus Rift and Oculus Touch with two tracking sensors in the standard recommended setup, running on computers equipped with Intel Core i5, 16 GB of RAM and Nvidia GTX 1060 graphics cards. There were two test stations in the room for simultaneous evaluation. Fig. 4.5 is a photo taken during the test session.

4.4.3 Procedures

We created a virtual environment specifically for the task shown in Fig. 4.6. The room consists of a starting area, where the objects are initially located, and a goal area, where the user needs to place the object, directly behind. The object can be placed in any manner within the goal area in order to pass, as long as it is within the boundary of the goal. The user is already facing the starting area when the test starts, and both starting area and goal area are color-coded to prevent confusion of the direction.

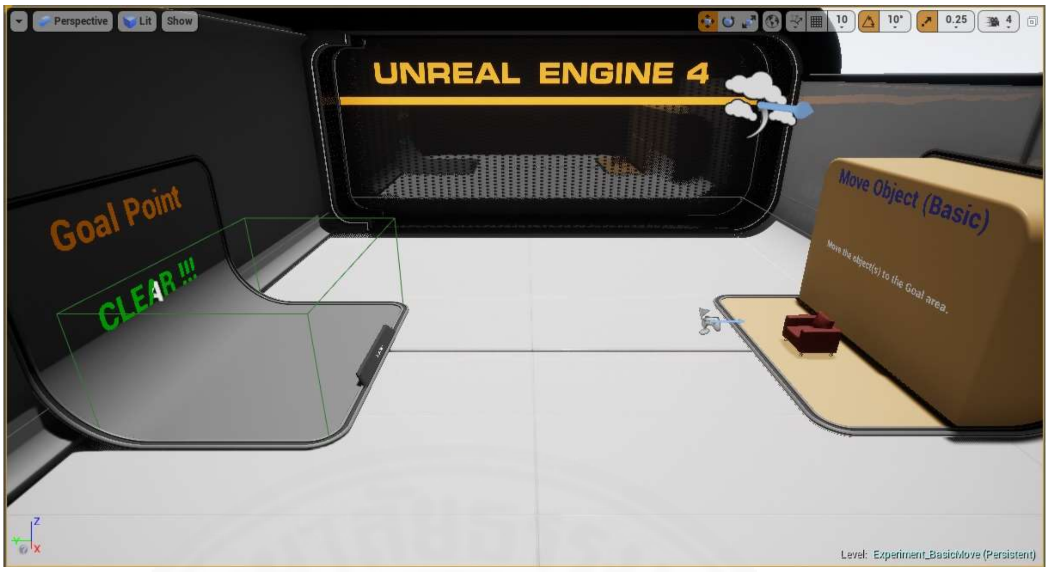

Perspective
Lit
Show
UNREAL ENGINE 4
Goal Point
CLEAR !!!
Move Object (Basic)
Move the object(s) to the Goal area.
Level: Experiment_BasicMove (Persistent)

Fig.4.6 Test environment in VR

We wanted to discourage the user from randomly throwing the object to the other side of the room in the hope that it will land in the goal area, so the roof of the space was left open, and if any object fell out of the room, the user will have to re-take the test again. The distance between the starting area and goal area was also introduced, which requires the user to teleport, the main VR locomotion methods we used in the software, at least once before placing the object in the goal, also to prevent throwing.

We divided the task into three levels of difficulty: basic, intermediate, and advance. Basic level has only one object that the user needs to move. Intermediate has five objects of the same type and size, while the advance level has eighteen objects with many different sizes, as seen in Fig. 4.7. The participants carried out the tasks with the proposed command-based interaction technique first, and then followed with the direct manipulation technique immediately. The total levels needed to pass the test is therefore six. Each participant took the test in the same order. The type and placement of the objects are exactly the same for both techniques.

During test levels intended for evaluating the command-based interaction technique, the user can use the command **Select All** and then set one of the object to be the target. This will lock the other objects to the target. The user can simply grab the main target and teleport to the goal area and place it in, along with all the child objects. The intermediate and advance levels require the exact same steps, regardless of the number of objects. During test levels intended for direct manipulation technique, the user needs to grab and move each object to the goal area one at a time. At the end of the test, there is a screen similar to Fig. 4.8 to display the result to the user. The system also automatically takes a screenshot of the result for record keeping.

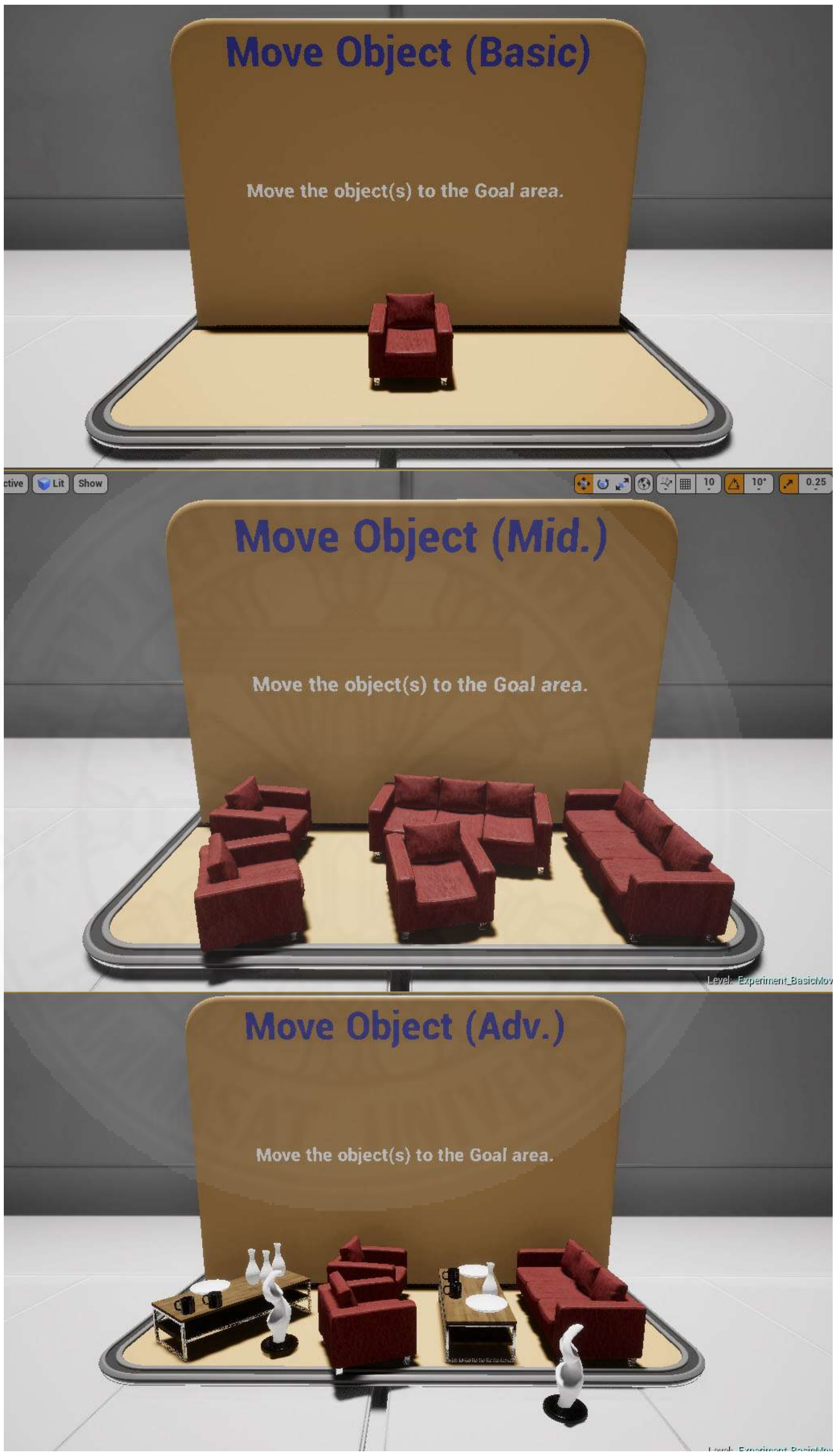
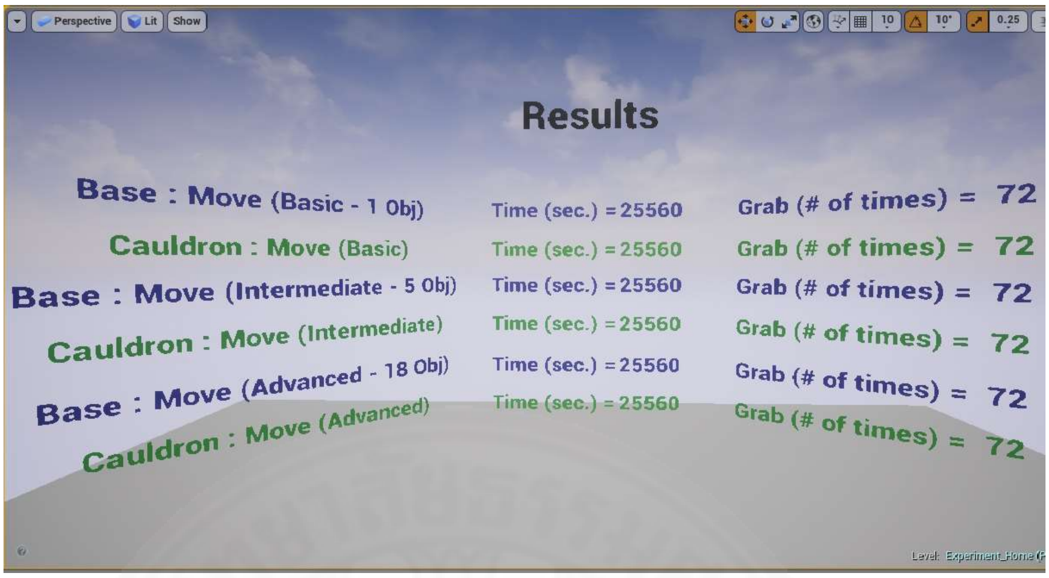

Move Object (Basic)
Move the object(s) to the Goal area.
Move Object (Mid.)
Move the object(s) to the Goal area.
Move Object (Adv.)
Move the object(s) to the Goal area.

Fig.4.7 Different difficulty levels



Results

Base : Move (Basic - 1 Obj)	Time (sec.) = 25560	Grab (# of times) = 72
Cauldron : Move (Basic)	Time (sec.) = 25560	Grab (# of times) = 72
Base : Move (Intermediate - 5 Obj)	Time (sec.) = 25560	Grab (# of times) = 72
Cauldron : Move (Intermediate)	Time (sec.) = 25560	Grab (# of times) = 72
Base : Move (Advanced - 18 Obj)	Time (sec.) = 25560	Grab (# of times) = 72
Cauldron : Move (Advanced)	Time (sec.) = 25560	Grab (# of times) = 72

Fig.4.8 Example of captured result screen

4.4.4 Results and Discussion

We collect two data from the test for evaluation purposes: task completion time, which represents efficiency; and number of **Grab** button presses to represent the amount of repetitive action the user needed to perform. During the test, the first problem we ran into was the complexity of the program. Since our test subjects came from various backgrounds and have different levels of experience with video games, some of the testers had trouble to perform basic tasks, such as moving around or picking up and dropping objects. We needed to compensate by allowing more time for each user to become familiarized with the software, which ended up resulting in each subject only taking the test once, instead of the intended multiple trials to measure learning curves. Also, due to the limitation of current VR hardware, a couple of testers had trouble getting the display to look sufficiently sharp, while some cannot wear glasses inside the VR headset due to the size of their frames, which effected their ability to judge distance correctly. Individual results show that some testers took significantly longer to complete the task and required many more attempts to grab an object.

Task Completion Time

The average task completion time for both techniques are shown in Table 4.1. The standard deviations of the results are shown in Table 4.2. The time starts to count from the moment the user enters the level and stops when the correct number of object(s) is placed into the goal area. Since we did the test of both techniques back-to-back, some users experienced confusion when we changed the level between direct manipulation and command-based technique, resulting in a few seconds of idle time before they realized the proper technique to use, especially in the first level.

Table 4.1 Average Task Completion Time (in seconds)

	1 Object	**5 Objects**	**18 Objects**
Command-based (proposed)	15.96	43.87	35.87
Direct manipulation	9.93	58.81	172.31

Table 4.2 Standard Deviation for Task Completion Time (in seconds)

	1 Object	**5 Objects**	**18 Objects**
Command-based (proposed)	13.94	30.22	18.52
Direct manipulation	4.18	24.60	75.04

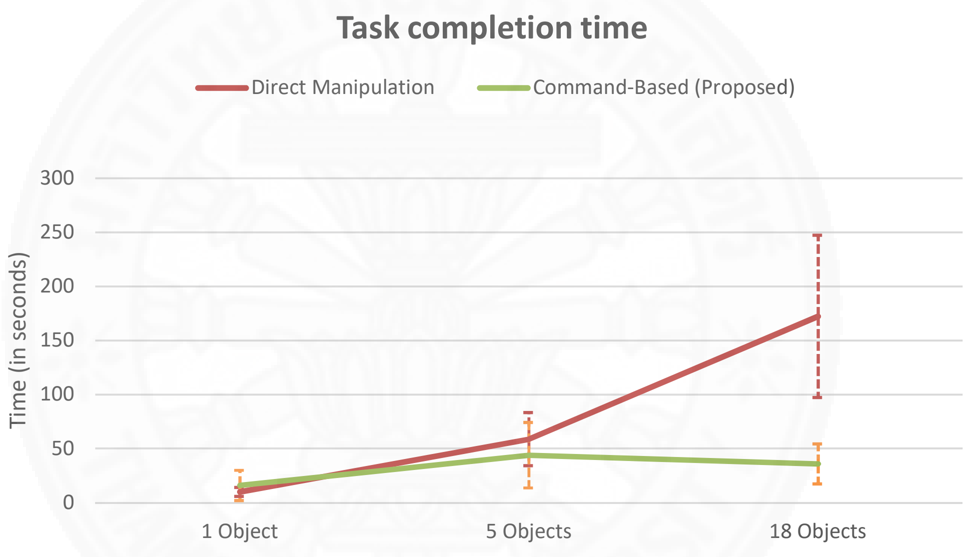


Fig.4.9 Graph for task completion time (in seconds)

Looking at the graph of task completion time in Fig. 4.9, when there is a small number of objects, the results are similar between both techniques. However, when the number of objects increases, the graph for direct manipulation is also increased in a steep slope, while for command-based technique the time stays relatively the same. For our particular tests, moving eighteen objects actually took less time on average than five objects because the actions that users need to perform are exactly the same, so they are learning and getting better in terms of both the average time as well as the standard deviation of the result, represented by the vertical dot lines in the graph. For direct manipulation technique, the different levels of proficiency of the user will be most noticeable, especially in the last scenario where some can complete the task in less than 100 seconds while others took well over 300 seconds.

Table 4.3 Average Number of **Grab** Button Presses

	1 Object	**5 Objects**	**18 Objects**
Command-based (proposed)	1.56	2.90	4.10
Direct manipulation	1.10	7.53	29.23

Table 4.4 Standard Deviation for Number of **Grab** Button Presses

	1 Object	**5 Objects**	**18 Objects**
Command-based (proposed)	0.91	2.74	5.08
Direct Manipulation	0.30	2.09	13.50

Number of **Grab** Button Presses

The number of **Grab** button presses are counted every time the user presses the physical button of the controller, shown in Table 4.3, not the amount of time the user successfully grabs the object. Table 4.4 shows the standard deviation of the result. Some users prefer to swap the grabbed object to their dominant hand, for example grabbing an object with the left hand then moving it to right hand, which the system also counted as two distinct actions. Several users are observed to rapidly pressing the grab button in a frantic manner when they missed grabbing the object the first time. This can also result in noticeably high action count.

The number of **Grab** button presses shown in Fig. 4.10, demonstrates a similar trend where for a small number of objects, the amount is comparable for both techniques. However, as soon as the number of objects increases, we can see a much steeper slope for direct manipulation versus command-based which required fewer actions throughout. For command-based technique during the last test, the increase in standard deviation might be a result from the way the test objects were set up in Fig. 4.7, since one white object is positioned outside the start area. When the user locks them together and move them to goal area, many of the users leave the white object just barely outside of the goal line, meaning they need to spend extra move to grab the object again.

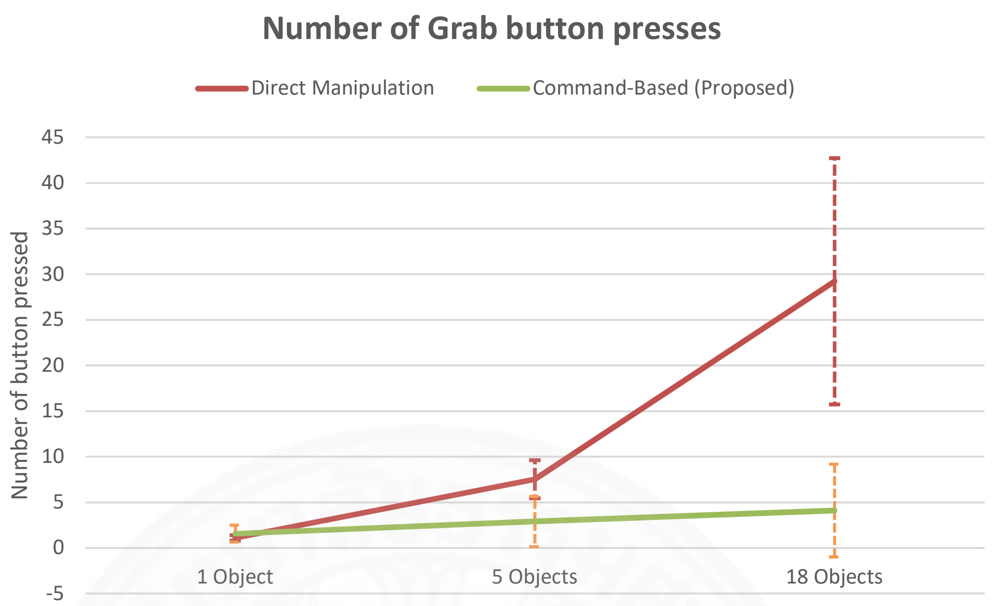


Fig.4.10 Graph for number of button pressed (times)

We used the Kennedy simulator sickness questionnaire (SSQ) to measure the side effects of VR on the users. The results in Fig. 4.11 show that most of the user have little to no problem using the prototype. The most noticeable issue was the clarity of vision due to low resolution displays inside the HMD, resulting in eye strain and slight dizziness, and general discomfort from having the device(s) strapped to their head and hands for an extended period of time.

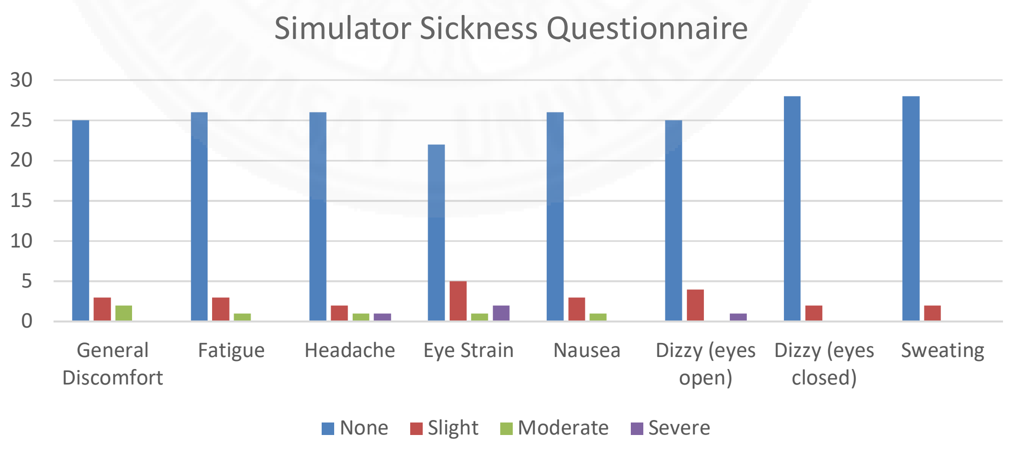


Fig. 4.11 SSQ results

In addition, the Likert scale questionnaire in Table 4.5 was used to gauge overall experience with the prototype. The result is shown in Fig. 4.12. We found mild to favorable feedback from the testers. However, we should also be mindful that there is some bias, as most of them never tried VR. In fact, some users never even played video games before, so the excitement might also come from the fact that they never knew such technology existed. Also, many of them still struggled with the controllers, even after warm-up and the test period ended, so the usability and UI of the platform clearly needs to be improved.

Table 4.5. Likert Scale Questions

Question 1:	I found the interface easy to learn.
Question 2:	Once I learned the interface, I found the **navigation** to be easy and intuitive to use.
Question 3:	Once I learned the interface, I found **object manipulation** easy and intuitive to use.
Question 4:	Once I learned the interface, I found that I could focus on being creative, instead of on the technicalities of the interface.
Question 5:	I prefer the new interface to a mouse and keyboard interface.
Question 6:	I was able to create the scenes that I wanted.
Question 7:	I would use the system myself in my home.
Question 8:	I would recommend the system to friends.
Question 9:	Overall, I enjoyed the experience.

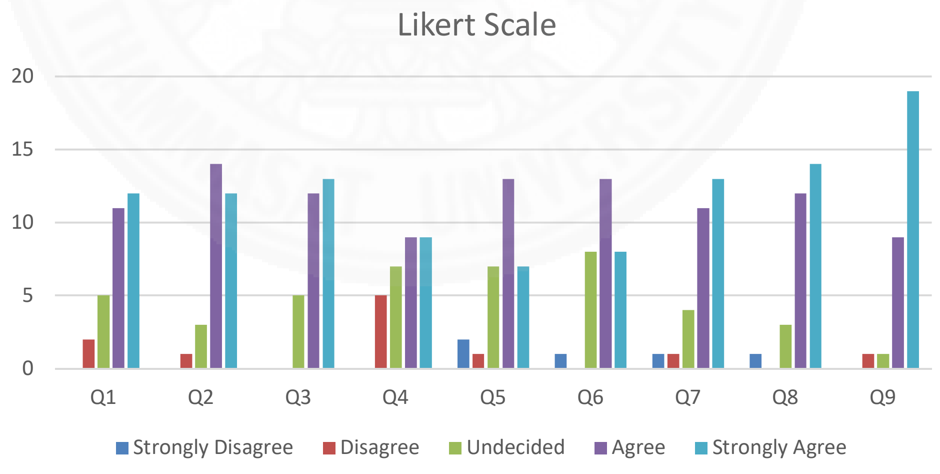


Fig. 4.12 Likert scale results

Chapter 5
Conclusions and Recommendations

Artists use various tools and technique such as sketches, drawings, clay models, photographs, and toys as references to help create their work. These references can also be used to plan and communicate idea with members of the team to make sure the project goes as smoothly as possible. This process of preparing and gathering references can be costly and time consuming. Therefore, we propose to use real-time rendering and simulation technology of modern videogames and VR to help off-load the technical aspects, and in turn let artists focus more on creativity. Our goal is to build an immersive VR prototype application (Cauldron) that lets artists quickly visualize and tests their pictorial design ideas in virtual environments.

5.1 Design Consideration and System Design

Cauldron is intended for visual artists who may or may not be proficient in using computer tools. The prototype allows artists to try their ideas out in the virtual world in a similar manner to real-life situations, while being easy to understand and use with minimal amount of buttons and panels. The user interfaces are intuitive to use and allow for a full range of controls that the artist needs to realize the design. The prototype also lets the artist test out different versions of ideas quickly, and swap or compare between ideas to make creative decisions right within the system, as well as provide convincing visual feedback that can be used to make reasonably accurate decisions. Oculus Rift is employed as the main VR device using either an Xbox360 controller or Oculus Touch as the input device. We also implemented a version for 2D monitors with keyboard and mouse controls for comparison. The prototype was implemented in UE4.

To use Cauldron, the user first creates a profile and selects between control devices options. The next step is to select the map or background of the scene that one will be working in. Then the user moves to the selected virtual space, where objects, characters, lighting setup, etc. can be add, remove, and manipulated to realize the design. After that, a virtual camera allows the user to take a photo of the scene and that photo will be exported to the hard drive.

5.2 Framework Implementation and Preliminary Evaluation

The main part of Cauldron is **CauldronCharacter**, a user-controlled character that consists of many subsystems each intended to do a very specific task. This was done so the user-controlled characters can be modular and easily customize to suit different input devices. The modularity could also be beneficial if we wish to expand in the future, for example, an eyes-only character without the ability to manipulate the completed design. The subsystem **World Interactor** manipulates physical objects in the scene. In the case of keyboard and mouse setup, a character will have only one **World Interactor** at the mouse crosshair, while VR characters with touch controllers will have one on each hand, allowing for more fidelity of interaction. Another notable subsystem is **Mannequin Interactor** that allow the user to adjust and move a virtual character in the scene to the desired pose freely. There are also many helper subsystems such as **Widget Interactor** to interact with the 3D diegetic UI, **Camera** to take the output photo, **Time-Of-Day** to change the time (and subsequently, overall lighting) of the scene, as well as **Save and Load** system to allows for quick comparison of different design versions of the scene. To test the usability of the prototype, we performed a test with six subjects, both males and females. The goal was to see if they can grasp the concept of Cauldron as well as determine the preferred locomotion and interaction methods.

It quickly became clear that traditional locomotion methods used in video games does not work in VR. The mouse and keyboard, as well as Xbox360 controller, caused simulation sickness to varying degrees for all of our test subjects within a few minutes, even though some are very used to it and think they prefer it over teleportation, which we ultimately chose as the main locomotion scheme for Cauldron. The limited degree of freedom for these controller schemes was also an issue. Two object interaction methods were implemented, a version of HOMER and fixed hands in front of camera. Both were unsatisfactory according to results from the test. However, besides getting simulation sickness, the test subjects were satisfied and have no trouble understanding the concept and usefulness of the system.

5.3 Virtual Object Manipulation

To solve the issue of manipulating objects in VR in the context of design, we developed a new interaction technique with the goal of minimizing repetitive actions, to help reduce discomfort and physical fatigue. We found that the most common task that the artist needs to perform is to move object(s) either to a target location or to follow certain objects. In real life, there are people (assistants) assigned to do this task specifically. Therefore, we recreated the same interaction in our software, but with the objects themselves acting according to our commands directly. The benefit is that the user does not need to micromanage every movement of the objects by him/herself. The idea is to group objects together, assign command(s) and let them keep execute said command automatically, while the user focuses on the bigger picture.

We implemented the interaction technique using Blueprint system in UE4. The object has built-in functions to execute tasks such as move to, follow (target) and rotate to (target). The user uses UI panel to issue commands to the selected object(s). The command will tell the selected object which object is the current target, calculate distance, speed and checks for validity, then it will execute the commands until the complete condition is met. In our prototype, the object moves by slowly interpolating its position towards the target object, ignoring any physics calculation, but other methods such as teleportation can be used as well, depending on the requirement of the application. The commands also can be nested together.

5.4 User Evaluation

We created a virtual environment to test the usability of the interaction technique. The task was to move the objects into a goal area as fast as possible. The test consists of three difficulty levels, depending on the number of objects. Our proposed technique is compared to direct hand manipulation, which is currently the most common method found in VR games and applications. The total level each tester have to complete is six. The testers consisted of thirty university students from information technology (10), multimedia technology and animation (10) and fine arts (10). Most of them have little to no gaming experience and none of them have ever used VR before.

The data that we collected include task completion time and number of grab button pressed. The results show that while our proposed technique performed similarly to the direct hand manipulation in the case of small number of objects, the improvement is significant when moving many objects. The testers initially needed some time to understand our technique, as it requires opening and using UI menu and selecting objects, compared to directly reaching out and grabbing with their hands. However, direct hand manipulation results indicate that gaming experience significantly effects the user efficiency, while our technique can be used by all testers regardless of their background. We also used SSQ to get subjective feedback from the testers. We found that while most testers have favorable feelings to the prototype, usability and control methods still need to be improved.

5.5 Recommendations

To develop application for VR, it is better to have a focus and directly target the full VR setup from the start, instead of trying to create a hybrid application that can be used on both 2D monitor and HMD. We spent significant time and effort to have implement a system that compensates for the lack of motion controller in the beginning, only to find that the experience is not even comparable. The level of freedom of movements, as well as immersion, that motion controller gives to VR is much greater that traditional gamepad. We eventually dropped the support of 2D monitor and gamepad setup in the end. It is also easier to choose locomotion and interaction methods if we were to target a specific device.

Due to time limitations when we performed our user test, we were not able conduct multiple tests per subject in order to find average values or analyze learning behaviors. The attention span of the students was also an issue. By the end, some students managed to sneak out, and the rest completely lost interest altogether. We would suggest to have multiple test days, if possible, as well as to perform the test in smaller, more manageable groups, instead of a single, large group.

References

Books and Book Articles

Gurney, J. (2009). *Imaginative Realism: How to Paint What Doesn't Exist*. Kansas City, Missouri: Andrews McMeel Publishing, LLC.

Jerald, J. (2016). *The VR Book: Human-Centered Design for Virtual Reality.* New York, NY: Association for Computing Machinery and Morgan & Claypool.

Articles

Arns, L.L., Cerney, M.M., (2005). The Relationship between Age and Incidence of Cybersickness among Immersive Environment Users. *Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality, VR '05*. 267-268.

Azhar, M., Ullah, S., Rahman, I. U., Otmane, S., (2015). Multi-Layered Hierarchical Bounding Box Based Interaction in Virtual Environments. *In Proceedings of the 2015 Virtual Reality International Conference, VRIC '15*.

Barreau, J.P., Gaugne, R., Bernard, Y., Le Cloirec, G., Gouranton, V. (2014). Virtual reality tools for the West Digital Conservatory of Archaeological Heritage. *Proceedings of the 2014 Virtual Reality International Conference, VRIC '14*. Laval, France.

Bigoin, N., Porte, J., Kartiko, I., Kavakli, M., (2007). Effects of depth cues on simulator sickness. *Proceedings of the First International Conference on Immersive Telecommunications, ImmersCom '07*.

Bolt, R.A. (1980). "Put –That-There": Voice and Gesture at the Graphics Interface. In *SIGGRAPH*. 262-270.

Bowman, D., Davis, E., Badre, A., Hodges, L. (1999). Maintaining Spatial Orientation during Travel in an Immersive Virtual Environment. *Presence: Teleoperators and Virtual Environments, vol. 8, no. 6*. 618-631.

Bowman, D., Hodges, L. (1997). An Evaluation of Techniques for Grabbing and Manipulating Remote Objects in Immersive Virtual Environments. *ACM Symposium on Interactive 3D Graphics 2*, 35-38

Bowman, D., Koler, D., Hodges, L. (1998). A Methodology for the Evaluation of Travel Techniques for Immersive Virtual Environments. *Virtual Reality: Research, Development, and Applications, vol. 3, no. 2*. 120-131.

Cakmak, T., Hager, H., (2014). Cyberith virtualizer: a locomotion device for virtual reality. *Proceedings of the ACM SIGGRAPH 2014 Emerging Technologies, SIGGRAPH '14*.

Clark, A., Moodley, D., (2016). A System for a Hand Gesture-Manipulated Virtual Reality Environment. *In Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists, SAICSIT '16*.

Dongsik J., Ungyeon, Y., and Wookho, S., (2007). Design Evaluation using Virtual Reality Based Prototypes: Towards Realistic Visualization and Operations. *In Proceedings of the 9th international conference on Human computer interaction with mobile devices and services, Mobile HCI '07*, 246-258.

Fischer, M. H., (2016). Inception: a Creative Coding Environment for Virtual Reality, in Virtual Reality. *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology, VRST '16*.

Gugenheimer, J., Dobbelstein, D., Winkler, C., Haas, G., Rukzio, E., (2016). FaceTouch: Touch Interaction for Mobile Virtual Reality. *In Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems, CHI EA '16*.

Hervy, B., Laroche, F., Kerouanton, J.L., Bernard, A., Courtin, C., D'haene, L., Guillet, B., Waels, A. (2014). Augmented historical scale model for museums: from curation to multi-modal promotion. *Proceedings of the 2014 Virtual Reality International Conference, VRIC '14*. Laval, France.

Ibayashi, H., Sugiura, Y., Sakamoto, D., Miyata, N., Tada, M., Okuma, T., Kurata, T., Mochimaru, M., Igarashi, T., (2015). Dollhouse VR: a Multi-view, Multiuser Collaborative Design Workspace with VR Technology. *SIGGRAPH Asia 2015 Emerging Technologies*.

Kaul, O.B., Rohs, M., (2017). HapticHead: A Spherical Vibrotactile Grid around the Head for 3D Guidance in Virtual and Augmented Reality. *Proceedings of* *the 2017 CHI Conference on Human Factors in Computing Systems, CHI ’07*. 3729-3740.

Kennedy, R.S., Lane, N.E., Berbaum, K.S., & Lilienthal, M.G. (1993). Simulator Sickness Questionnaire: An enhanced method for quantifying simulator sickness. *International Journal of Aviation Psychology*, 3(3), 203-220.

Kim, T., Park, J., (2014). 3D Object Manipulation Using Virtual Handles with a Grabbing Metaphor. *IEEE Computer Graphics and Applications*, vol. 34, no. 3, 30-38.

Kirner, T. G., Martins, V. F., (2000). Development of an Information Visualization Tool Using Virtual Reality. *Proceedings of the 2000 ACM symposium on Applied Computing*, 604-606.

LaViola, J.J. (2000). A Discussion of Cybersickness in Virtual Environments. *SIGCHI Bulletin* Volume 32/1, January 2000, 47-56.

Lee, J., Kim, B., Suh, B., Koh, E., (2016). Exploring the Front Touch Interface for Virtual Reality Headsets. *In Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems. CHI EA '16*.

Likert, R., (1932). A Technique for the Measurement of Attitudes. *Archives of Psychology*. 140: 1–55.

Linderman, R.W., Sibert, J.L., Hahn, J.K., (1999). Hand-Held Windows: Towards Effective 2D Interaction in Immersive Virtual Environments. *In IEEE Virtual Reality*. 205-212.

Llorach, G., Evans, A., Blat, J., (2014). Simulator sickness and presence using HMDs: comparing use of a game controller and a position estimation system. *Proceedings of the 20th ACM Symposium on Virtual Reality Software and Technology, VRST '14*. 137-140.

Marks, S., Estevez, J. E., Connor, A. M., (2014) Towards the Holodeck: Fully Immersive Virtual Reality Visualization of Scientific and Engineering Data. *Proceedings of the 29th International Conference on Image and Vision Computing,* New Zealand.

Mateevitsi, V., Sfakianos, M., Lepouras, G., Vassilakis, C., (2008). A Game-Engine Based Virtual Museum Authoring and Presentation System. *Proceedings* *of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts, DIMEA '08*.

Mazalek, A., and Nitsche, M., (2007). Tangible Interfaces for Real-time 3D Virtual Environments. *In Proceedings of the International Conference on Advances in Computer Entertainment Technology, ACE '07*.

Mine, M. Brooks, F.P. Jr, Sequin, C.H., (1997). Moving objects in space: exploiting proprioception in virtual-environment interaction. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques, SIGGRAPH '97*. 19-26.

Mine, M., Yoganandan, A., Coffey, D., (2014). Making VR Work: Building a Real-World Immersive Modeling Application in the Virtual World. *Proceedings of the 2nd ACM symposium on Spatial User Interaction, SUI '14*.

Mlyniec, P., Jerald, J., Yoganandan, A., Seagull, J., Toledo, F., Schultheis, U. (2011). iMedic: a Two-Handed Immersive Medical Environment for Distributed Interactive Consultation. In *Studies in Health Technology and Informatics*. Vol. 163. 372-378.

Nittala, S., Li, N., Cartwright, S., Takashima, K., Sharlin, E., Sousa, M. C., (2015). PLANWELL: Spatial User Interface for Collaborative Petroleum Well-Planning. *Proceedings of the SIGGRAPH Asia 2015 Mobile Graphics and Interactive Applications*, *SA '15*.

Pierce, J.S., Stearns, B.C., Pausch, R. (1999). Voodoo Dolls: Seamless Interaction at Multiple Scales in Virtual Environments. *In Symposium on Interactive 3D Graphics*. 141-145.

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*. 79-80.

Roedl, D., Stolterman, E. (2013). Design Research at CHI and Its Applicability to Design Practice. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '13*.

Stolterman, E., (2008). The Nature of Design Practice and Implications for Interaction Design Research. *International Journal of Design*, 2(1), 55-65.

Stolterman, E., Pierce, J., (2012). Design Tools in Practice: Studying the Designer-Tool Relationship in Interaction Design. *Proceedings of the Designing Interactive Systems Conference, DIS '12*.

Vinson, N., Lapointe, J.F., Parush, A., Roberts, S., (2012). Cybersickness induced by desktop virtual reality. *Proceedings of Graphics Interface 2012, GI '12*. 69-75.

Electronic Media

Benson, J. (2013). *Omni Treadmill sprints past Kickstarter goal in 4 hours; immersive VR now one step closer*. Retrieved July 2018, from https://www.pcgamesn.com/omni-treadmill-sprints-past-kickstarter-goal-4-hours-immersive-vr-now-one-step-closer

Billinghurst, M. (2017). *3D User Interfaces for VR*. Retrieved from July 2018, from https://www.slideshare.net/marknb00/comp-4010-lecture-4-3d-user-interfaces-for-vr

Cerga, R. (2018). *What Is The "Screen-Door" Effect & Why Does It Happen?* Retrieved July 2018, from https://vrvisiongroup.com/what-is-the-screen-door-effect-why-does-it-happen/

Epic Games. (2018). *Epic Games Homepage*. Retrieved July 2018, from https://www.epicgames.com/

GitHub. (2018). *UnrealEngine repository*. Retrieved July 2018, from https://github.com/EpicGames/UnrealEngine

Google. (2018). *Designing for Google Cardboard*. Retrieved July 2018, from https://designguidelines.withgoogle.com/cardboard/

Hill, A., Johnson, A. (2008). *Withindows: A Framework for Transitional Desktop and Immersive User Interfaces, Figure 2.* Retrieved from July 2018, from https://www.researchgate.net/publication/4324508_Withindows_A_Framework_for_Transitional_Desktop_and_Immersive_User_Interfaces

Intel. (2018). *Guidelines for Immersive Virtual Reality Experiences*. Retrieved July 2018, from https://software.intel.com/en-us/articles/guidelines-for-immersive-virtual-reality-experiences

Jerald, J. (2). (2016). *VR Interactions*. Retrieved July 2018, from https://www.slideshare.net/LearnWTB/vr-interactions-jason-jerald

Lipp M., (2017). GIS2VR: From CityEngine via Unity to HTC Vive. Retrieved July 2018, from https://www.esri.com/arcgis-blog/products/city-engine/3d-gis/gis2vr-from-cityengine-via-unity-to-htc-vive/

Lyon, H. (2014). *Process to the People: Part 1*. Retrieved July 2018, from http://www.muddycolors.com/2014/04/process-to-the-people-part-1/

Merriam-Webster. (2018). *Virtual Reality*. Retrieved July 2018, from https://www.merriam-webster.com/dictionary/virtual%20reality

Metaclassofnil. (2016). *Comparing the seated Vive and Rift CV1 experience in Radial-G*. Retrieved from July 2018, from http://blog.metaclassofnil.com/?p=837

Newegg. (2018). *Oculus Rift + Touch Virtual Reality System*. Retrieved from July 2018, from https://www.newegg.com/Product/Product.aspx?Item=N82E16826910005

Oculus (1). (2018). *Introduction to Best Practices*. Retrieved July 2018, from https://developer.oculus.com/design/latest/concepts/book-bp/

Oculus (2). (2018). *Guidelines for VR Performance Optimization*. Retrieved July 2018, from https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-performance-guidelines/

Oculus (3). (2018). *What are the minimum and recommended system specifications needed to power Oculus Rift?* Retrieved July 2018, from https://support.oculus.com/170128916778795/

OLED-Info. (2018). *Pentile OLEDs: introduction and market status*. Retrieved July 2018, from https://www.oled-info.com/pentile

Quora. (2018). *How much time does it takes to learn autodesk maya animation and to work as pro?* Retrieved July 2018, from https://www.quora.com/How-much-time-does-it-takes-to-learn-autodesk-maya-animation-and-to-work-as-pro

Steinicke, F., Bruder, G., Hinrichs, K., Jerald, J., Frenz H., Lappe, M. (2009). *Real Walking through Virtual Environments by Redirection Techniques*. Retrieved July 2018, from https://www.jvrb.org/past-issues/6.2009/1747

Sylights. (2018). *Lighting Diagrams*. Retrieved July 2018, from http://www.sylights.com/lighting-diagrams/editor

Unreal Engine Documentation. (2018). *Virtual Reality Best Practices*. Retrieved July 2018, from https://docs.unrealengine.com/en-us/Platforms/VR/ContentSetup

UQO Cyberpsychology Lab. (2013). *French-Canadian validated version of the SSQ*. Retrieved July 2018, from http://w3.uqo.ca/cyberpsy/docs/qaires/ssq/SSQ_va.pdf

Wikipedia (1). (2018). *Unreal Engine*. Retrieved July 2018, from https://en.wikipedia.org/wiki/Unreal_Engine

Wikipedia (2). (2018). *Oculus Rift*. Retrieved July 2018, from https://en.wikipedia.org/wiki/Oculus_Rift