NAVIGATING IMMERSIVE AND INTERACTIVE VR ENVIRONMENTS WITH CONNECTED 360° PANORAMAS

by

SAMUEL COSGROVE JR
B.S. University of Central Florida, 2014

A thesis submitted in partial fulfilment of the requirements
for the degree of Master of Science
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2020

ABSTRACT

Emerging research is expanding the idea of using 360-degree spherical panoramas of real-world environments for use in "360 VR" experiences beyond video and image viewing. However, most of these experiences are strictly guided, with few opportunities for interaction or exploration. There is a desire to develop experiences with cohesive virtual environments created with 360 VR that allow for choice in navigation, versus scripted experiences with limited interaction. Unlike standard VR with the freedom of synthetic graphics, there are challenges in designing appropriate user interfaces (UIs) for 360 VR navigation within the limitations of fixed assets. To tackle this gap, we designed RealNodes, a software system that presents an interactive and explorable 360 VR environment. We also developed four visual guidance UIs for 360 VR navigation. The results of a pilot study showed that choice of UI had a significant effect on task completion times, showing one of our methods, Arrow, was best. Arrow also exhibited positive but non-significant trends in average measures with preference, user engagement, and simulator-sickness. RealNodes, the UI designs, and the pilot study results contribute preliminary information that inspire future investigation of how to design effective explorable scenarios in 360 VR and visual guidance metaphors for navigation in applications using 360 VR environments.

Dedicated to my friends and family who have supported me in the pursuit of my dreams.

ACKNOWLEDGMENTS

Thanks to my thesis advisory committee – Dr. Joseph LaViola, Dr. Hassan Foroosh, and Dr. Charlie Hughes. Additional thanks to the members of UCF's Interactive Sytems and User Experience Lab, whose help and resources were vital to the completion of this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

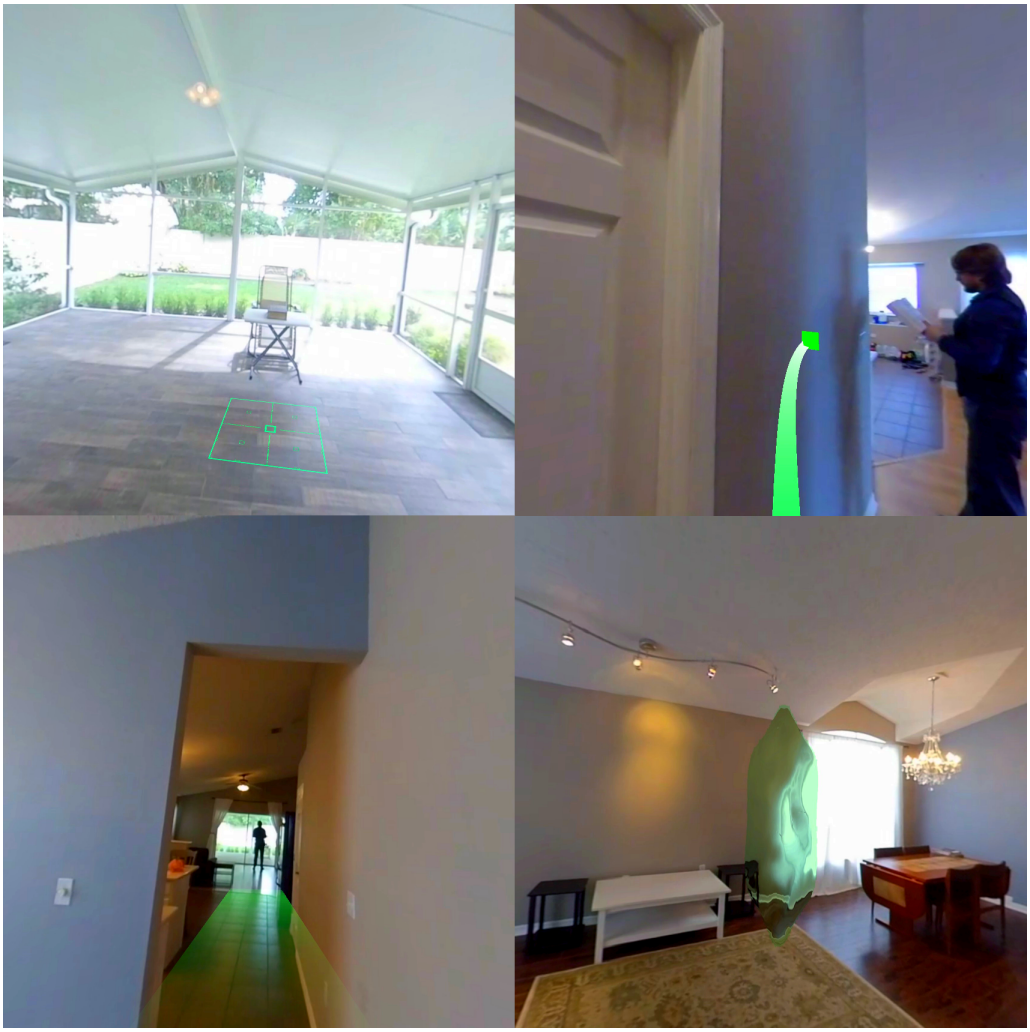

Figure 1.1: Images from the RealNodes software displaying the four types of visual guidance UI. (Top-left) Target; (Top-right) Arrow; (Bottom-left) Path; (Bottom-right) Ripple.

Recent virtual reality (VR) research has reignited interest in producing 360-degree spherical panoramas of real-world environments to incorporate into "360 VR" experiences. Modern commodity 360-video camera technology makes it easier to capture real-word data

for use in VR experiences. However, using images mapped as panoramas for a 360-degree VR experience dates back decades to technologies like QuickTime VR [8]. In the late 90's and early 2000s, commercial video games like Myst III and The Journeyman Project 3: Legacy of Time experimented with making worlds with synthetic 360-degree environment maps, occasionally composited with actors or real props, organized such that if a user clicked in certain directions, they would teleport to another location through either a fade transition or a prerendered transition video[4, 36]. Only recently with affordable consumer 360-video cameras can we combine these past ideas with modern technologies to research new kinds of 360 VR experiences with explorable, interactive worlds made with real-world data. Rather than simple video and image viewing, research is expanding beyond simple applications, exploring technological and human factors challenges of enhancing immersion beyond guided experiences.

Prior research in traditional VR applies to any exploration of 360 VR, such as navigation, wayfinding, and improved visualization. Some unique research has begun to explore the unique possibilities and challenges of 360 VR compared to traditional VR, like applying Walking-In-Place (WIP) locomotion to control 360-video playback [27], focus assistance in viewing 360-video [21], and visual transitions when navigating 360 VR environments [24].

However, there remains a gap with emerging applications in 360 VR being unable to combine these methods into a realistic approximation of an explorable and interactive holistic virtual environments (VEs). The feeling of truly walking within a 360 VR environment compared to one fixed video, or the ability to interact directly with the environment rather than it being a distant video background, and the ability to have integrated synthetic objects and alterations of the environment, would benefit emerging 360 VR applications. Training systems with navigation-based decision-making would be possible. Virtual tours can have the freedom of someone taking branching paths in a natural locomotive way, rather than a menu selection. Improved fitness applications can provide multiple routes and backtracking during

runtime, instead of a menu. New kinds of commercial games are possible that combine real-world 360-video and real actors with synthetic objects and characters for unique experiences.

This gap motivated us to develop RealNodes, a software engine for creating and presenting scenarios combining pre-filmed 360-video and virtual assets to create 360 VR location waypoints connected in a cohesive interactive environment. RealNodes is a fully functional system combining WIP locomotion for multi-path navigation between different video locations, real-time composited 360-video rendering, interactivity with cut-out sections of the video for animation, and support for integration of interactive virtual objects accurately lit with Image Based Lighting (IBL) based on the 360-video background.

Additionally, there remains a problem with emerging applications in 360 VR needing better designs for navigation user interface (UI). Applying traditional VR UIs to 360 VR does not account for unique limitations. Synthetic environments have the freedom of granular user position and geometry, while 360 VR is limited by fixed assets for environments. There is a lack of comparative evaluation of navigation UIs in 360 VR research and how they can affect the user experience. This leaves a gap for best practices in designing appropriate UI for visual guidance, wayfinding, and navigation for 360 VR. We felt this was necessary to explore alongside the development of RealNodes, as otherwise taking a na ïve approach to UI did not appropriately meet the goals of RealNodes being an engaging user experience.

This gap additionally motivated us to create four visual guidance UI designs for indicating waypoints in 360 VR: Target, Ripple, Path, and Arrow (pictured in Figure 1.1). We developed and integrated these designs into RealNodes. Additionally, we performed a pilot study on these four UIs to test the efficacy of the system and the strengths and weaknesses of the UI types on user experience. Preliminary results of the study show one of our methods, Arrow, had a statistically significant difference in scenario completion times.

In summary, our contributions are the following. We present the RealNodes system for creating and presenting 360 VR scenarios. We present four visual guidance UI designs.

We additionally present the results of a pilot study of the UIs. These contributions provide feedback for research into creating interactive and explorable 360 VR environments with effective navigation UIs.

Reader's Guide

Chapter 2 - A review of the relevant literature from the fields of VR, computer graphics, and human computer interaction (HCI). We categorize the relevant literature into several thrust categories relevant to the emerging exploration of experiences incorporating 360-degree images and video.

Chapter 3 - Technical description of the prototype software RealNodes, including implementation and feature details. The four visual guidance UI techniques we developed are also described.

Chapter 4 - Description of a pilot user study comparing the four visual guidance UI methods. This section provides details on the demographics of participants, technical setup, experimental procedure, and statistical design and analysis methods used. The preliminary results and discussion of the user study are presented with statistical analysis described.

Chapter 5 - Description of plans for refinement, further investigation, and future work.

Chapter 6 - The conclusion for our work.

Appendix A - Examples of all the questionnaires presented to participants for collection of demographic information and qualitative metrics.

Appendix B - IRB Approval Letter

CHAPTER 2: LITERATURE REVIEW

Past and emerging research in traditional VR regarding topics covering navigation, wayfinding, improved visualizations, and narrative experience applies to the evolving research of 360 VR. RealNodes iterates on this work to create a fully functional system combining 360-video, WIP multi-path navigation, virtual object integration, and composited 360-video rendering in a way we have not seen in prior literature. Additionally, we were unable to identify previous studies evaluating visual guidance UI for 360 VR. The context that has led us to our research is described below.

Navigation Techniques

Navigation in VR has been widely researched to tackle the challenge of providing effective options for users to explore virtual environments (VEs). An earlier example of research in interaction and navigation techniques is from Tomozoe et al. [39] who developed a gesture-based interaction system where the same gestures can be performed on objects that are strictly defined as moveable (interactable objects) and unmovable (navigation targets), such that performing the same basic gestures have different behavior. This is as opposed to strict gestures strictly for interaction and strictly for navigation, or mode switching. For example, a "waving-away" or "beckoning" gesture will move a moveable object further or closer from the player, while the same gestures on an unmovable object will move the player closer or farther to that object. Additionally, they proposed a "tunnel-window" tool for creating portals in the environment that show a remote area out of normal field of view (FOV). This allows both interaction with long range moveable objects, and teleportation to the scene in the window by interacting with unmovable objects. RealNodes also has gesture-based interaction and navigation actions as part of its system, though the control

mechanisms for them are distinct from each other compared to this system. Although a remote view like the "tunnel-window" proposed here is not implemented in RealNodes, future work could explore using alternate views like that or World-In-Miniature for alternative navigation modes.

Machuca et al. [6] devised a user interface called Fluid VR to perform navigation and selection task controls with a seamless transition between modes. They compare their system to traditional VR applications that have explicit button or gesture toggles for mode switching. They claim that unconstrained interaction and navigation modes have the propensity for mode errors, which can frustrate users. Fluid VR presents three constraint-based interaction techniques: manipulation, pre-programmed interaction-based navigation, and a flying method. By having these constrained systems, they claim a high accuracy and ease of use, allowing for fluid changes between modes. Manipulation of moveable objects is constrained by an extension of the Shift-Sliding algorithm, where the object moves along the surface of the object it is touching based on virtual cursor motion. For selection and use of repetitive action objects (example: ladder climbing), Fluid VR prevent false inputs by waiting until the controller is stationary before allowing further actions. For navigation tasks, a button press explicitly selects the object to navigate to, and a gesture stroke determines distance and direction of either a "fly-to" or orbiting method based on the kind of object. Though there is still an aspect of switching between modes and points of interest through button and gesture press, they take advantage of constraints for each point of interest to switch between modes fluidly, rather than an explicit mode switch. RealNodes has two modes: navigation and interaction. This does have an explicit button press toggle. However, there is a clear visual indication with the Visual Guidance UI. This helps prevent false mode input and makes clear what mode the user is in.

Mas et al. [25] developed Indy, a VR training software system for workers in industrial facilities to train them on navigation and searching tasks in their job. It is a multi-player

game with asynchronous collaboration. An instructor/trainer manages the scenario, while trainees are split into two groups: "Hunters" and "Radios". Hunters explore the environment in VR to find the objective, while Radios use desktop software to see floor maps and 360-degree photographs to guide the Hunters. The design of Indy is geared toward developing environment familiarity as part of navigation. They discuss the challenge of developing spatial understanding of large complex environments such as an industrial building are constrained by not just design decisions that need to instill learning, but also time and schedule constraints. They propose that active navigation and decision-making leads to forming a mental model of a place, instilling better spatial knowledge. Indy proposes to develop these spatial skills for navigating the industrial building being trained against. This occurs by having Hunters actively explore 3D environments while verbally collaborating with Radios who inspect the maps and 360-images to aid in the search task. Indy is unique in that it presents two different networked applications simultaneously as part of its system, where one is a synthetic VE for VR exploration, while the other includes 360-degree images of the real environments from select locations. This is a unique two-pronged way to present data enhanced by 360-degree panoramic content. RealNodes does not implement a map system but could benefit from a map tool in the future to aid in wayfinding and searching tasks.

A study developed by Powell et al. [33] evaluated three methods to navigate VR using low-cost mobile VR: Continuous forward motion with only steering control, a magnetic toggle switch directly on the head mounted display (HMD) where travel can be stopped and resumed, and a Bluetooth controller with direct control of forward and backward motion. Results showed that direct control with the controller was a faster task and was preferred technique. Participants disliked the lack of control of movement and the inability to tightly turn. Compared to our research and the capabilities or RealNodes, their research was concerned with a purely synthetic environment with motion primarily on one axis. RealNodes presents 360 VR with a capability for multiple path choices and freedom to backtrack.

Lorenz et al. [23] developed two navigation methods for natural walking (Wii Balance Board and Kinect) targeting two different VE displays (CAVE style system called Powerwall, and Mixed Reality glasses). A comparative user study was performed on the four possible conditions between the walking method and the VE display. Results indicated that Kinect and Powerwall had benefits compared to the other systems. Kinect felt more realistic to move in the environment compared to Wii Balance Board. This seems to indicate that real locomotion is preferred over stationary locomotion on a step sensing board. Users of Powerwall felt the experience was more enjoyable, felt more like parts of the environment were responding to them, and that they were participating in the environment. Though the authors claim that the Kinect and Powerwall combination is promising, the same combination was significantly more nausea inducing. RealNodes uses a VR HMD for VE display, compared to either of the methods used in the work of Lorenz et al. While RealNodes uses limited real-world locomotion within waypoint locations, navigation between locations is primarily done with WIP. Compared to the methods here, it lies between full locomotion and requiring a peripheral under the feet like the Wii Balance Board. This is part of why we chose it as a compromise to the feeling of walking while working in a small physical space.

Tanaka et al. [38] devised a large-scale experiment to test a novel input interface for navigating 360 VR on a mobile touch device compared to a conventional on-screen button-based method. Their "Motive Compass" was a slide pad interface. surrounded by arrows indicating the direction of a branching path and allowing for velocity control. It was compared in a "large-scale demonstration experiment in a real exhibition" with a more standard "Ground Arrow" method, where arrow icons would only be visible over a visible pathway with only one choice of velocity. Their results show that their interface better presents accessible directions compared to conventional methods. RealNodes has taken inspiration from both their visual guidance metaphors as well as the ability to control speed of navigation.

Paris et al. [30] developed two user studies comparing three methods of exploring

large virtual worlds in the constraint of a smaller physical space and their effect on presence and spatial memory. Two of the methods were reorientation techniques, which more directly influences the user to face a direction that allows for continued walking. One is reorientation using artificial rotational gain, which alters the speed of the virtual rotation and makes it look like the user turned too much. The user is instructed to turn around until they are facing a direction that allows for continued travel. The other method is reorientation with distractors. When reorientation is required, the user is instructed to follow a distractor object while induced rotation of the world is occurring. The third method is a redirected walking system, where the user's rotation is subtly changed from the real-world rotation speed, forcing them to correct for perceived misalignment. This results in them being repositioned appropriately. The first user study compared the three methods and their effect on presence and spatial memory. Participants performed a navigation task where they were specifically directed to objects, then asked to memorize a location and identify targets out of their FOV. The results indicated that scenarios with the distractor reorientation gave greater presence than those with the redirected walking. Their reorientation method had issues in the first experiment and pilots for the second, causing enough simulator sickness for participants to drop out. The second study was reformatted to compare only the two reorientation methods, this time allowing free exploration. Results indicated a significant effect on learning the environment, with scenarios using rotational gain reorientation taking less time than the distractor method. RealNodes has discrete, user-centered locations that don't require much real-world locomotion. And the navigation system uses a WIP method. So redirected walking or reorientation doesn't have direct use in the current version of the system. However, a similar system could be developed that ties navigation transition videos to real locomotion for use in a larger physical space, at which point reorientation or redirection should be considered.

RealNodes is concerned with a fuller HMD-based 360 VR experience and therefore

navigation methods appropriate for different physical interfaces. The WIP technique has been investigated in the literature for both traditional VR and 360 VR. One method requiring only an HMD with an accelerometer is described by Lee et al. [20], based on work by Zhao on 3-axis pedometer design [42].

Muhammad et al. [27] explored using WIP to navigate 360 VR by actively controlling playback of a video in 360 VR on a smartphone HMD. They found that compared to passive interfaces that merely played back the video, simulator sickness was reduced. One problem the authors had was that rotation caused false positives inn their step algorithm. We overcame that problem in RealNodes implementation by allowing steps only when rotated towards a navigable path. Another limitation that Muhammad et al. had was that the application was limited to linear playback, compared to RealNodes which allows for choice in direction.

Similarly, Tregillus and Folmer [40] developed a WIP interface for navigating a synthetic VR environment on a smartphone HMD. They ran a comparative study with WIP against a gaze-direction based Look Down to Move (LDTM) method. They found that WIP was more immersive and intuitive. This strengthened the idea for RealNodes incorporate WIP compared to other physical interfaces.

Though not exactly a WIP implementation, Lopez-Gulliver et al. [22] developed a related 360 VR experience paired with a treadmill to provide interactive environment for exercise. Based on the user's speed on the treadmill, key frames of the video are synthesized and interpolated to smoothly slow down or speed up the video displayed. Navigation tasks in 360 VR can benefit from a smoother visual transition between locations. The information from this research was relevant to our solution of developing an image-based playback and interpolation of transition videos based on user speed when performing WIP. However, the "navigation" task in their software is a one-axis movement on a treadmill, compared to RealNodes which adds a second dimension of navigation with multiple paths the user can

choose based on orientation.

Habgood et al. [17] performed an evaluation of three locomotion techniques in VR: arc-based teleport, free movement with an analog stick, and a rapid node-based shift activated by gazing at a waypoint icon and pressing a button. Results indicate that rapid node-based locomotion reduced motion sickness compared to free motion. Additionally it took less time to navigate a scenario with node-based compared to the other methods. RealNodes also presents scenarios in a node-based manner with active movement. However, our system presents a video transition timed to WIP, rather than rapid shift transition or free movement independent from real motion. Additionally, our study explored multiple wayfinding UIs in a node-based system instead of locomotion modes.

Assisted Focus and Guidance for Wayfinding

In the realm of 360-video, a constant challenge is encouraging focus on intended targets in the video and not letting the viewer feel like they missed something important. This has led to work on UIs that either directly change focus or more gradually guide viewers to points of interest in the video. To do this effectively, we need to look at prior literature for wayfinding in VEs. Smith and Hart [37] developed a model of distributed resources in a human-computer cognitive system for navigation tasks. Out of this, they produced several wayfinding aids fundamental for HCI design. They discuss the concept of distributed cognition, a hybrid approach to the study of cognition involving people and artifacts versus just the individual. They discuss thinking about design of HCI applications with distributed cognition. This facilitates reduction of cognitive load by externalizing representations and concepts and building them into the user interface or the environment.

For human-computer interaction, the authors propose a resource model with two components: abstract information structures and interaction strategies. Abstract informa-

tion structures can be divided into six items to classify information that informs interaction: plans, goals, current state, interaction history, action-effect model, and affordances. Interaction strategies can be divided into four categories: plan following, plan construction, goal matching, and history-based selection/elimination. These strategies and resources interrelate with each other and should be considered as part of HCI design. Smith and Hart make a distinction between motion/travel, navigation, and wayfinding. They pull from the literature to form a definition of navigation as an aggregate of the strict action of moving through the environment (motion/travel) and the cognitive process of defining a path through an environment by using acquired knowledge and environment cues (wayfinding). Five wayfinding aids are presented by the authors as being fundamental components to easing the cognitive load by separately fulfilling some of the abstract information structures for informing navigation. Those are a simple distance metric (goal, action-effect), compass (goal, affordance, action-effect), dead reckoning directional instructions (goal, affordance, action-effect), maps for route planning (goal, plan, action-effect), and plan following tools such as a graphical path rendered in the environment (plan).

A user study was performed by Smith and Hart to evaluate the five proposed wayfinding aids for use in a desktop VE. Participants were tasked with finding all landmarks in the environment. The results indicated that dead reckoning was the only method with low cognitive load, while distance only had medium load, and the rest had high load. This contradicted the expected cognitive load of all methods except for dead reckoning. Besides the limited amount of participants in the study, the authors indicate three other factors were determined to contribute to the mismatch between expected and actual results: impact of resource allocation on artifacts was optimistic; implementation of the aids had notable problems; and that the only measure of cognitive load used was usability problems rather than a combination of that and other measures (ex. Completion time for task, spatial knowledge of environment, etc.).

RealNodes presents multiple visual guidance UIs for wayfinding in a search task like this example of early research on wayfinding in VEs. Our designs needed to follow the fundamentals of designing for cognitive load in VEs to be successful. However, by nature of the design of RealNodes, the user must stop and evaluate the new waypoint they have reached. This makes the cognitive load situation somewhat different from a traditional free-movement VE with more continuous evaluation of navigation.

Freitag, Weyers, and Kuhlen [11] devised an interactive assistance interface that extends any free exploration system with ground-based travel. The system guides the user to unexplored areas on request, factoring in analysis of what the user has already seen, where they have already explored, and what places will be interesting based on viewpoint quality. They propose from the literature that designing a scene to support good wayfinding contributes to "exploration success", which they define as seeing all important areas in an environment. Traditional wayfinding aids like compasses or maps they say are merely support tools that do not necessarily ensure that all important areas are seen in an environment. Virtual tours with automated paths and camera positions are referenced as an example of exploration success. Their criticism of virtual tours is the automated nature with limited to no interaction, and that most tour systems do not consider where the user has already explored. Their method is purely computational and automatic for a scene. When requested, three suggestions are computed and presented to the user. A photo of each location is presented along with the distance to the location, and colored tube paths on the ground are shown for each location. When paths followed the same waypoints, the paths were laid side-by-side, which combined with different colorization makes clear which split off points are for which path. These paths are computed on a navigation mesh for the environment, prioritizing viewpoint quality based on a visibility histogram computed across all objects. Viewpoint quality on this histogram utilizes an object uniqueness metric to favor new and unique objects in a scene, reducing weight based on how well that object is seen at a large

visual size. Based on prior literature claiming that users often want to visit close locations before new locations, the three locations suggested are computed based on the two closest locations with high quality, followed by the globally highest quality viewpoint as the third. A user study was performed to determine if the proposed method added a benefit compared to free exploration without the interface. They found an improvement in knowledge of the environment and higher exploration success, and participants experienced the interface as helpful and easy to use.

The solution of Freitag et al. was designed for arbitrary scenes where they don't have the ability to modify a synthetic scene with landmarks. Meanwhile RealNodes has a stricter but similar restriction by only having real-world video data for the environment and no geometry. RealNodes is concerned with exploring effective graphical assisted guidance for exploration success like the work of Freitag et al. Our Arrow method is in part inspired by their method of always indicating direction of close by waypoints.

Lin et al. developed focus assistance user interfaces and ran an experiment to determine methods for visually guiding users better while they watch a 360-video [21]. The challenge of continuous focus and refocus on intended targets was what they wanted to tackle with their two techniques: "Auto Pilot", which directly changes the view to the target, and "Visual Guidance", which indicates the direction of the target in the UI. They found that their focus assistance techniques improved ease of focus overall, but any other effects focus assistance has depended on both the video content and the goals for watching the video. This content was developed for standard 360-video streaming, but lessons learned from this can benefit navigation and interaction tasks in 360 VR. This inspired our development of RealNodes to improve navigation and interaction tasks in 360 VR with our pilot study on methods targeted at a multi-path navigation and searching task. Exploration in RealNodes scenarios can happen at the user's pace, with indicated waypoints encouraging focus and preventing the feeling of missing something important.

YouTube and Facebook have some tools for their 360-video distribution such as Facebook Guide, which allows for setting hotspots of interests in a video [31]. This provides a method of visual guidance to viewers in the content but does not give the viewer ways to interact or engage with the content. Even more advanced commercial software packages for developing 360 VR experience such as Pixvana [35] are limited toolkits that produce what is still plain video content with just a few dialogue choices, again with few opportunities for creating the feeling of direct interaction or exploration.

Effective Visual Transitions

One of the challenges of creating any kind of VR world is having effective methods of visually presenting transitions between locations. 360 VR adds the additional challenge of not being able to rely on a fully available virtual world with all its geometry.

Moghadam and Ragan [26] conducted a comparative study on three visual transition types for movement in VR: Teleportation, Animated interpolation, and Pulsed interpolation. Teleportation is an instant change in position/rotation with three sub-types: Instant Change, Fade to Black, and Blurred Fade. Animated interpolation is a smooth viewpoint change with three speeds. Pulsed interpolation is a series of intermediary teleportations leading up to the final state, with three choices of intermediate points. The results show the faster transitions are preferred by participants with more VR experience, while slower transitions are preferred by those who do not. Based on their findings, we made considerations about speed and interpolation for RealNodes system of visual transitions that blends 360-video of the current location with a manually filmed transition to the next waypoint.

MacQuarrie and Steed [24] developed VEs made from 360-degree images connected into different locations that could be navigated through by pointing with a wand controller at a waypoint sphere and pressing a button. The authors conducted a comparative study

on visual transitions types for navigating this kind of environment: instantaneous teleport, a linear movement through a 3D reconstruction of the real-world scene, and an image based "Möbius" transformation. Results indicate that there was no significant difference in spatial awareness between transition types, while significant differences were found in other measures. Movement profiles were longer for teleport than other transitions. 3D model and Möbius transition were better for feeling of motion. Model and teleport had a higher preference compared to Möbius. These indicate that different visual transitions have a variety of trade-offs relying on what the goal of the experience being made is. Taking inspiration from this work, RealNodes implements manually filmed transition videos in lieu of a 3D model to aid in better feeling of motion. While their system only had one kind of waypoint object metaphor (a floating sphere), we implemented four visual guidance UIs.

Cho et al. [9] developed a novel method to integrate multiple 360-images into a VE. They combine simple image interpolation and warping into a method to simulate movement from different image locations as a player moves using controller input. They organized their images into a virtual data map using commercial photogrammetry pipeline software. They performed synthetic experiments with their novel image synthesis method compared to ground truth images. Results showed that as the synthesized image location gets closer to a real image location, the interpolation is more accurate. Our method differs in having a VE made from videos connected with ground truth filmed transitions, compared to many images mapped together in their environment.

Narrative Experience

360 VR experiences are being developed to convey narrative in a unique way compared to traditional film and VR, with its own unique challenges. Aitamurto et al. [5] presented 'Uturn", a novel 360-video about gender inequality from a male and female perspective,

simultaneously presented as two halves of a 360-video in a HMD. A comparative user study showed that the split view increased the feeling of presence, embodiment, and understanding of the characters more than a traditional view. The role of the viewer becoming an implied actor in the experience has been explored by Bindman et al. [7] in a CGI VR film called "Invasion" about interacting with a bunny dealing with alien antagonists. Their study found that viewers felt more like they were a character (often another bunny) when viewing in 360 VR version compared to a traditional view. Pope et al. [32] tackled the opposite topic: how actors behave differently when filming for 360 VR. They presented a play in two configurations: immersive theater in which an audience member sits in a swivel chair in the center of the stage, and 360-video where the camera is in the center. They found actors gave less personal space to the camera compared to a human, indicating that personal space needs to be accounted for when filming for 360-video.

More generalized research exists for the presentation of "interactive films" that can feed into creating 360 VR experiences. Verdugo et al. [41] proposed a novel design framework for interactive storytelling called "Coconstruction". Compared to common frameworks from video games, Coconstruction is a "detour framework" with decision points that are either on the "Backbone" of the story or are "detours" that, instead of creating more branches, are designed to return right back to the "Backbone". Rather than many branching paths or infinite choice within a limited scope, an author lays out pieces for the viewer to "co-construct" with them into a story, defined by both the planning of the author and the viewer's perception. Two studies are discussed. The first is an exercise where four pictures were given to participants without context, and participants were asked to write stories out of them. It was found that despite many possible combinations, most stories fit into three similar main themes. The second study involved developing an interactive movie, "The Crime or Revenge of Fernando Moreno", and presenting it to participants. The film was designed so the backbone would not change, regardless of what detours were made. Instead,

the study showed that any difference in perception of events in the backbone were influenced by the viewer's choices.

Though the scenarios developed for our study do not have an overt narrative, we do have a non-player character (NPC) present in many locations within the environment to aid in immersion and engagement in a unique way. RealNodes is additionally designed to present videos based on event triggers, making it a prime target for future work making scenarios with branching storylines and decision making.

Integration of Interactive Elements

Authoring 360 panoramic images and video content together with virtual elements into a cohesive 360 VR experience is an emerging research challenge. Understanding methods to realistically merge interactive virtual elements with 360 panoramic content is relevant to creating options for interaction in 360 VR.

Felinto et al. [10] developed a production framework for combining captured 360 images with synthetic elements. They incorporated IBL to light the synthetic objects, including incorporating reflections and shadows, to integrate the objects. "Support" objects are created to match the exact position of a real-world object in the panorama, texture mapped to blend into the scene, and augmented to add effects like deformation when colliding with a synthetic object (ex. purely synthetic object deforms a couch augmented with Support geometry). They developed a Blender plugin to augment the authoring process of placing synthetic and Support objects in the environment, properly aligned with the 360-degree panorama.

Rhee et al. developed a software system called MR360 to demonstrate real-time integration of accurately lit interactive virtual objects in a live stream of a 360-video [34]. It uses the input video to apply IBL, including realistic soft shadows from perceived generated

light sources, to the virtual objects. A user study was conducted showing an improved sense of presence in MR360 compared to conventional 360-videos without interaction.

RealNodes employs IBL to light any static and interactive virtual objects like MR360 to match the environment. MR360 was designed to generate VEs with live 360-videos, compared to the RealNodes system which uses prerecorded images and video. Though RealNodes requires manual authoring of scenarios compared to the framework made by Felinto et al., it does so with video and image data compared to just panoramic image data.

Stereo/3D Reconstruction Techniques

Stereo 3D 360 cameras exist that allow for a brute-force method of creating stereo 360-degree images and video. Systems like the Samsung 360 Round [13] and Insta360 Pro series [28] are similarly designed. An array of multiple cameras is arranged in a circle at a known separation, sometimes with a camera pointing up to compensate for missing information above the camera. The knowledge of the camera properties and orientation allows for the use of known 3D computer vision methods for stereo matching. This allows for two panoramas to be stitched together, one for each eye, that are stereo matched. These systems target professional filmmakers and are considerably more expensive than standard consumer 360 cameras, with prices ranging from around $5,000-15,000 (as of early 2020). They also tend to rely on offline server-based computation methods for the stitching, often a proprietary step done by the camera manufacturer.

Some work has been done to perform 3D reconstruction of an environment solely with 360-degree panoramas. Huang et al. [15] developed a novel view algorithm that warps a monoscopic 360-degree panorama view into a stereo view. This is enabled by offline calculation of Structure from Motion (SfM) from a 360-video, then warping the image based on data from the dense reconstruction (as opposed to displaying the dense reconstruction

directly). This warping algorithm had a low computational footprint on their tested GPUs, enough for greater than 120 frames per second (FPS) performance. Based on feedback from a user study performed, the stereo view did not cause discomfort compared to the monoscopic view when looking at close objects but did cause discomfort when moving far away from objects due to higher distortion.

Work by Im et al. [16] proposed a method using Small SfM on monoscopic 360-video frames to generate a dense depth map for a location. Their solution uses a bundle adjustment method tailored to the spherical camera model of a consumer 360 camera. They developed a "sphere sweeping" method, a dense matching method that looks at overlapping regions on the two fisheye lenses of the camera system to assist in matching. The video only required small jittery movements of the camera in place for the reconstruction to work. Their resulting algorithm on both synthetic and real-world environments resulted in low reprojection error compared to traditional bundle adjustment methods and ground truth.

RealNodes can use a panoramic depth map to perform a shader-based geometric displacement of the panorama centered on the user. The depth map produced used an attempted version of the method described by Im et al. This is different from the method proposed by Huang et al. which involves distortion of the left and right eye view separately based on an acquired point cloud. Though implementing a structure-from-motion 3D reconstruction for RealNodes was out of scope of this Thesis work, we acknowledge a need to explore it for future research.

CHAPTER 3: SOFTWARE DESIGN - REALNODES

RealNodes is a software system for creating immersive and interactive 360 VR scenarios, developed to explore the challenge of effective navigation in 360 VR. It presents separate locations as their own 360-video, each logically connected with 360-video transitions facilitating multi-path, bidirectional traversal. By allowing scenarios of real-world pre-recorded location and transition videos, we balance realism and immersion with lower cost and time to create than synthetic environments or photogrammetry/reconstruction.

Our system allows for additional virtual assets, all lit with correct IBL. These can be static or dynamic, adding interactivity otherwise absent from simple 360-video. A video manager system performs real-time dynamic loading, playback, and compositing of 360-video. This allows for a variety of locations, transitions, and interactivity with animated cut-out sections of background. Additionally, our WIP system is tied to playback of transition videos for locomotion. This adds additional realism, comfort, and engagement with the environment by simulating truly walking in the environment. The visual guidance UI system clearly indicates what waypoints a user can navigate to based on facing direction. This completes the capability we sought to develop for enabling scenarios with free, user controlled, multi-path navigation in 360 VR.

This chapter is organized into the following sections:

Content Capture - Describes the hardware used for capturing 360-degree video and the specifications of the data produced.

Input and Display - Describes the hardware used for RealNodes software control and display, and the basic controls of the software.

Software Design Overview – Describes the high-level design of the RealNodes software, including scene hierarchy and scripts/shaders.

Video Manager – Detailed description of the video manager system, which controls

the state management and rendering of the 360-degree video environment.

Walking-in-Place Implementation – Detailed description of the WIP implementation.

Visual Guidance User Interface Methods – Detailed description of each of the Visual Guidance UIs we developed for RealNodes.

Interaction Objects – Detailed description of the interaction systems in RealNodes.

Metrics Recording – Detailed description of the metrics recording system built into RealNodes for data collection purposes.

Scenario Development – Detailed procedural description of the process of authoring a scenario for the RealNodes system.

Content Capture

Commodity 360-degree multi-camera systems incorporate two synchronized wide-angle lenses that capture raw images and video. These camera systems usually have automated image stitching software built into the camera or provided external to the device. The resulting data is projected to a flat equirectangular format image, a 2:1 ratio image where the pixel horizontal and vertical axes map to spherical coordinates.

For our work, we utilized the Insta360 One X (See Figure 3.1) [1]. It is a two-lens multi camera system in a small form factor. Each wide-angle lens captures 200 degrees FOV, providing a 10-degree buffer of overlap around each lens to aid in stitching. It has onboard buttons for configuration and recording, but we used the official Android app to allow remote start/stop of recording, which avoids having the cameraperson in the videos.

The highest 360-image resolution the camera supports is 6080 * 3040. Several video recording options are available for varying resolution and FPS: 5760 * 2880 @ 30 FPS, 5760 * 2880 @ 25 FPS, 5760 * 2880 @ 24 FPS, 3840 * 1920 @ 50 FPS, 3840 * 1920 @ 30 FPS, 3008 * 1504 @100 FPS. After several test recordings of our scenario environment, we chose to

record all our video with the 3840 * 1920 @ 50 FPS (“4K”) mode, balancing high resolution with relatively smooth playback in VR.

Figure 3.1: Side and top view of Insta360 One X. Illustrates the viewing angle for each lens.

In lieu of a standard tripod, we made our own monopod from the Insta360 One X selfie stick and a ruggedized mini tripod intended for large professional desktop cameras. This minimized the camera appearance in the nadir of the 360-degree video, all while standing as stable as a conventional tripod. Though such monopods for this purpose are sold commercially, this setup worked for our needs.

Input and Display

RealNodes was made for deployment on SteamVR platforms, minimally targeting the HTC Vive VR system (see Figure 3.2 [3]). The Vive utilizes an HMD and wand style controllers that use an outside-in tracking system. The Vive has Lighthouse base stations which sweeps infrared light across a room synced with sensors on the HMD and controllers. Multiple base stations can define a 3D tracking space for triangulating the HMD and controller positions. The HMD contains a 3.6” diagonal screen with a 2160 * 1200 resolution (1080 *

1200 for each eye) with a 90 Hz refresh rate.

Figure 3.2: HTC Vive HMD and Vive Controllers.

The Vive controllers are primarily designed for hand position/rotation tracking for gesture and interaction tasks with VR. They have a multifunction trackpad, trigger, grip buttons, and system/menu buttons. Because our software was developed for SteamVR, it can work with other SteamVR systems with little/no changes. Notionally, development was primarily tested with the Vive Pro, a higher end version of the system with a higher resolution screen (2880 * 1600) [2].

RealNodes can be controlled with one controller. Only two buttons are used: *Trigger* and *Grip* (see Figure 3.3). Using the *Trigger* simulates a grabbing action. This is used to either grab objects or to manipulate switches in the environment (described in section Interaction Objects). The *Grip* button is used as a mode toggle to turn $NavigationMode$ on/off. This controls both the display of visual guidance UIs (described in Visual Guidance User Interface Methods) as well as the ability to use WIP (described in section Walking-in-Place Implementation).

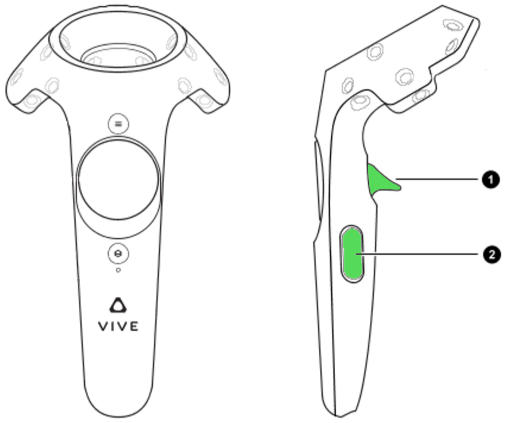

Figure 3.3: Simple instructions image produced for participants in our study, illustrating how to use the Vive controller in the software.

Software Design Overview

RealNodes was developed with Unity3D targeting Windows 10 desktops. It was made for SteamVR, targeting the HTC Vive. RealNodes minimally requires one controller for grabbing and manipulation tasks, as well as toggling *NavigationMode* On/Off. This controls both the display of visual guidance UIs as well as the ability to use WIP. Since most of the scene is defined from 360-videos, there are fewer synthetic objects in the scene hierarchy compared to most VEs. Instead, these are dense with scripting logic to define the visuals and state based on user action. Major components include a video manager system for video playback and rendering, a waypoint state machine system, sensor data manager for interaction and WIP, and a simple file writer to produce experimental logs (timestamped event data, sensor data, etc.)

Scene Hierarchy

RealNodes organizes its VE scene graph hierarchy into several objects. These objects are defined by a mixture of 3D geometry, scripts with assets assigned, exposed configuration variables, and nested children objects. See Figure 3.4 for an image of the RealNodes software loaded in the Unity3D development environment. Since most of the scene is image-defined from the 360-videos, there are fewer objects defined in the scene compared to other VEs. However, these objects are dense with scripting logic to define the scene and control the state transformations that occur from player action. We describe this hierarchy below.

CoreNode is the target object for the final processed 360 VR environment and lighting for 3D objects. It primarily contains a large sphere acting as a render surface for the 360 VR environment. The player always interacts with the world inside this sphere. The sphere is large enough for the surface to appear far from the player, while close enough to allow for the viewpoint to change with minor player motion. Normally, we would use a skybox at infinite distance to do environment map rendering like this. However, if we used the skybox alone, the background would not realistically shift when the player moves. Thus, the sphere object was added for that small addition to realism. Both the sphere and skybox contribute IBL to any interaction objects contained in it.

[*SteamVR*] is the application programming interface (API) core rendering object to support the VE to run on SteamVR hardware. It runs the default *SteamVR_Render* script as part of the SteamVR API.

Player is the core collection of objects that define the user avatar. To support SteamVR, this includes objects for the HMD and left/right hand controllers for tracking. Controller inputs and tracking data are collected by scripts attached to the VR related objects and sent to the *SensorDataManager*. Additionally, contained in the *Player* are helper objects used for the Path and Arrow visual guidance UI, respectively (described in

the section Visual Guidance User Interface Methods).

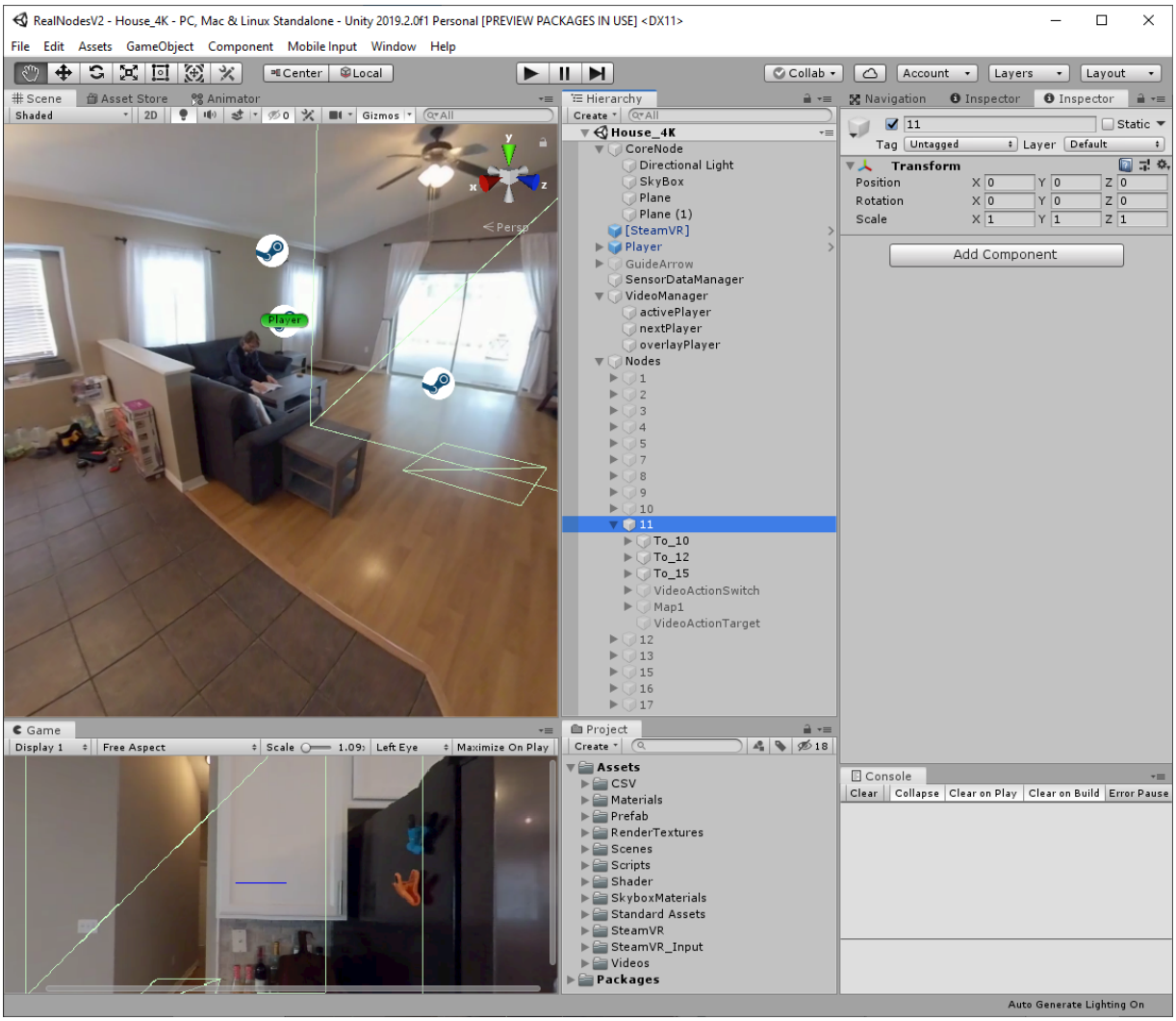

Figure 3.4: View of the RealNodes software project development environment in the Unity3D game engine.

GuideArrow is the core environment object that defines the appearance and behavior of the Arrow UI (explained in section Visual Guidance User Interface Methods).

SensorDataManager is the central collector of tracking data from the HMD and left/right controllers. This data is then sent to appropriate objects and/or scripts in the

environment for state tracking and change.

VideoManager is the central controller for the state management and rendering of the 360 VR environment. Based on where the player is, whether they are performing WIP, and whether they trigger an environment animation, the render pipeline changes video states to influence the final environment map rendering (described in the VideoManager section).

The *Nodes* object is the collection of waypoint node objects in the environment that define the interaction and navigation in each location of the 360 VR environment. Rather than having these locations physically laid out in the VE, they are overlapped over each other and are made active only when the waypoint state has changed for the player. At minimum, each waypoint has objects for neighbor waypoints placed in accordance to where they are in the corresponding 360-video for that waypoint. These have scripts that dictate the transition states sent to the VideoManager, waypoint objects additionally contain core visual guidance UI objects for the Target and Ripple UIs respectively that are only active when either of those UIs are enabled. In addition, a waypoint can contain an *ActionTool* and *Page* object, which are interaction objects for use as part of the search task (described further in the section Interaction Objects).

Assets

RealNodes includes a mixture of custom developed assets and leveraged/extended standard assets from the SteamVR library and Unity3D game engine. Custom scenes, scripts, shaders, render materials, and video assets are organized in their own folders for the scene hierarchy to reference. Prefab objects, or scene graph objects with a specific definition intended for reuse and instancing, are also organized here. Note that other than test scenes, the operation of RealNodes takes place in only one scene definition.

Scripts and Shaders

Unity3D's primary programming language for its API is C#. Most of these scripts inherit Unity3D's *MonoBehaviour* API class, which allows them to be attached to objects in the scene graphs, communicate with other scripts through message passing, and trigger periodic update threads every frame. Unity additionally uses shaders in a custom language called ShaderLab, though the syntax is similar enough to common shader language formats like GLSL to allow for implementing common algorithms from the literature.Figure 3.5 illustrates the high-level architecture of RealNodes, showing the relationship between major scripts as well as their inputs and outputs, showing a principle of closely linked systems. Below are high-level descriptions of the custom scripts and shaders we developed.

MultiVideoManager is the densest class definition in the project. Due to the importance of the logic for video playback and rendering, it acts as the focal point of almost all software logic, receiving inputs from many other scripts in the system. Based on the current state of the script, and inputs received from the rest of the system, it sends new inputs to the *SkyboxPanoramicBlended* shader. This shader synthesizes the state inputs and currently active 360-video data into the final 360 VR environment map. *CommonEnums* holds the enumeration definitions for all the videos for location, transitions, and action videos, as well as the visual transition mode. *PathHotspotV2* is the state machine attached to each waypoint object, storing configuration data on which waypoint it is departing and which one it is going to. It also controls render logic based on *NavigationMode* state and which visual guidance UI is set. *ViveCursor* controls ray casting logic for the HMD, which indicates which waypoint the player is facing. Based on this data, different states are triggered for the corresponding *PathHotspotV2* script and the *MultiVideoManager*. The above scripts and shader are detailed further in the section Video Manager.

SensorDataManager is the focal point of sensor tracking data for the HMD and

left/right controllers. It additionally receives button states for the controllers captured from *SimpleInput*. This data is sent to other scripts in the system that require tracking and controller input. Primarily, the *WalkInPlaceScript* get raw HMD tracking data and processes it to determine whether WIP is being performed. This calculated step data is sent to *MultiVideoManager* to trigger transition playback.

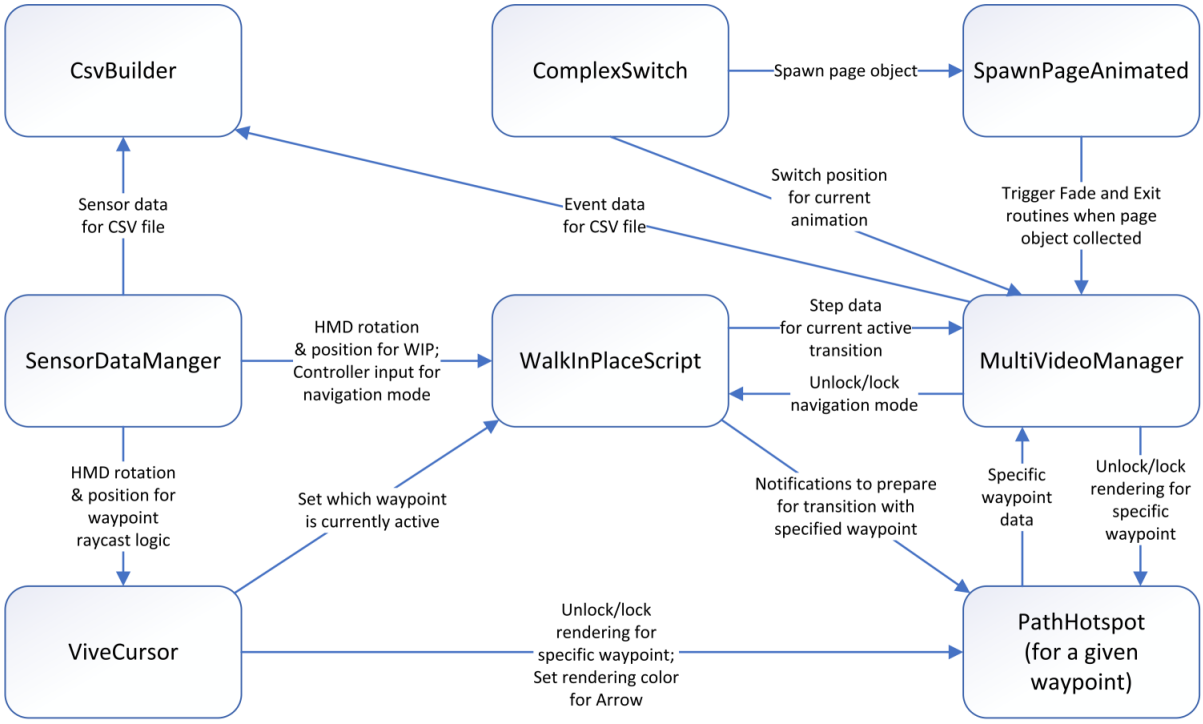

Figure 3.5: High-level architecture diagram of the different script modules in RealNodes and their inputs/outputs to each other.

SimpleTextureScroll is used to control the *MaskedRipple* shader, which controls the animation of the Ripple visual guidance UI. *SimpleFixRotation* dictates the behavior of the Path visual guidance UI. *ArrowColor*, *PointArrow*, and *BezierCurve* all dictate the behavior of the Arrow visual guidance UI. The above scripts and shader are detailed further in the section Visual Guidance User Interface Methods.

ComplexSwitch tracks the state of the *ActionTool* that it is attached to. *SpawnPageAnimated* handles both the spawning and pickup behavior of the *Page* object that it is attached to. The above scripts are detailed further in the section Interaction Objects.

CsvBuilder is used to write timestamped metrics for a specified logging file (described in the section Metrics Recording).

Video Manager

A major feature of RealNodes is a multi-video management system for changing the active video based on event triggers and providing layered video effects. Unity3D Render Textures are used as targets for our videos. A Render Texture can be modified and updated at run time. This allows for receiving dynamic data like video frames while being treated as a standard texture by our *SkyboxPanoramicBlended* shader and merged with graphical effects that are finally sent to the user's view. *MultiVideoManager* holds references to Unity3D Video Player objects, all the video files, and all the Render Textures. Our system allows for smoothly fading between two videos as they are simultaneously playing, such as the current location video and a transition video. It also supports preprocessed videos with a green "alpha" mask color to place an animation on top of the background. This allows for animating partial regions of video based on event triggers, such as opening a closet on command. Additionally, the *VideoManager* has a CsvBuilder script attached to handle writing the event data file (described further in Metrics Recording).

Render Pipeline

Figure 3.6 illustrates the high-level render pipeline for the VideoManager, which dictates the appearance of the 360 VR environment around the player. At any given time, there is up to three videos actively loaded into memory. These videos are tied to one of

three Video Player objects: *activePlayer*, *nextPlayer*, and *overlayPlayer*. Every frame of playback of the Video Player objects is sent to a different Render Texture, which acts as input to the *SkyboxPanoramicBlended* shader. See Figure 3.7 for an example of a location video frame in equirectangular format.

Figure 3.6: The Video Manager render pipeline, illustrating the high-level process of how the 360 VR environment is made.

The image data combined with instructions from *MultiVideoManager* dictates the final environment background. *MultiVideoManager* can intervene at each level of this process: which video clips are loaded and prepared, which videos are playing or paused, and how much of each video is visible. The *VideoManager* leverages the Video Player object

and the Render Texture asset type from the Unity3D API. The Video Player object controls playback of a provided video clip. Video can be played, paused, and even set to a specific frame. It supports several render modes to allow the frame data to be output to a specified render target. Since we needed flexibility to perform custom operations with the frame data, we used Render Textures. A Render Texture is a special kind of texture that can be modified and updated at run time, allowing it to receive dynamic data like video frames while being treated as a standard texture by a shader.

The *activePlayer* plays the primary video for the current situation. Usually, this is the location video for the current waypoint node the player is in. Whenever a new video is needed, it is swapped from the *nextPlayer* into the *activePlayer*, at which point playback of the new video can occur.

The *nextPlayer* contains the next planned video. Rather than load in video right when a change is needed, RealNodes is configured to dynamically load the next possible transition video every time the player looks at a new waypoint. This allows the likely chance that once WIP is initiated, the transition video will already be loaded and prepared.

The *overlayPlayer* is loaded with the effect animation for the current waypoint the player is in. Unlike the other Video Player objects, the playback of the video is not automatic, but is instead dictated by the *ActionTool* (described in Interaction Objects). When there is no animation in the waypoint node, this is empty and any functions regarding its use are skipped.

MultiVideoManager Script

MultiVideoManager controls all behavior for the Video Manager. Its three most important functions are the *Update* thread and the two coroutines *Blend* and *PlayOneShotVideo*. They are fundamental to performing the playback and combination of video data.

The main *Update* thread runs every frame of the RealNodes application (90 FPS).

Figure 3.7: Example of an equirectangular image from one of the location videos. This defines the background of the 360 VR environment.

Its primary function is to wait for signals to start a new instance of the Blend coroutine, as well as trigger playback of animations.

Blend is the core coroutine thread that facilitates switching between two videos. It performs a fade transition between the videos in *activePlayer* and *nextPlayer*, until *nextPlayer* is the dominant video being played. At which point the clips are swapped and the next video has now been set to the primary video.

PlayOneShotVideo is the main coroutine thread that performs a transition from one waypoint location to another, primarily when WIP is executed. First, it calls *Blend* to fade between the current location video to the transition video. Then it blocks until the script receives step data from the *WalkInPlaceScript*, at which point it progresses playback of the transition video if it receives step data. Playback of the transition stops and the coroutine continues to block if step data stops being received. Once the playback of the transition has

finished, the next location clip is prepared ending the coroutine. The script is now in a state where the next time *Update* is called, it will trigger a new instance of *Blend* that will fade from the finished transition video into the newly prepared location video.

There are several publicly available functions in *MultiVideoManager* allowing other objects to trigger state changes (refer to Figure 3.5). These include receiving step data from the WIP system, receiving triggers to prepare the next video clip, executing the next transition, manipulating the animation overlay, and performing a fade to black and exiting the application. *MultiVideoManager* contains a state machine of which locations map to each possible transition video. Based on the inputs received, the correct transition and location videos will be loaded into the appropriate Video Player objects.

MultiVideoManager also contains references to several environment objects to trigger state changes outside of itself (again, refer to Figure 3.5). Primarily, it toggles *NavigationMode* and rendering of visual guidance UI to avoid player control conflicts during video transitions. Additionally, it writes data to an event log every time a transition occurs, or when the application ends.

Transitions

One of the most important capabilities of *MultiVideoManager* is performing transitions between multiple videos. The script allows for smoothly fading between two videos as they are both simultaneously playing, such as the current location video and the transition video to be played back. See Figure 3.8 for an example of a transition video frame.

As described above, the *Update*, *Blend*, and *PlayOneShotVideo* functions, combined with external state change commands, are key to this occurring. Below is pseudocode of the primary functions for performing the video transition sequence:

```
/*
    Main update thread. Checks if a) Not playing transition, b) currently switching video, c) next
```

Figure 3.8: Example of an equirectangular image from one of the transition videos. Note the masked out black region removes the cameraperson from view.

```
        ↪ video prepared, d) donePreparing = false. If true, start a new Blend.
*/
void Update ()
{
    if (!playingTransition && switching_video && nextPlayer.isPrepared && !donePreparing)
    {
        donePreparing = true;
        StartCoroutine("Blend");
    }
    CheckOverlayPlayback();
}

// Externally called to prepare next transition clip for current location and next location.
public void PrepareNextClip(LOCATIONS next_index)
{
    if ((nextLocation != next_index || firstPrepare) && canPrepare)
```

```
    {
        firstPrepare = false;
        canPrepare = false;
        nextLocation = next_index;
        currentTransition = GetNextTransition();
        nextClip = transitionPlaylist[(int)currentTransition];
        nextPlayer.clip = nextClip;
        nextPlayer.Prepare();
        donePreparing = false;

        // Trigger cooldown routine to temporarily block new clips from being prepared.
        StartCoroutine("PrepareCooldown");
    }
}

// Prepares next location clip based on nextLocation, after a transition.
public void PrepareNextClip()
{
    if (nextClip != playlist[(int)nextLocation])
    {
        nextClip = playlist[(int)nextLocation];
        nextPlayer.clip = nextClip;
        nextPlayer.Prepare();
        donePreparing = false;
    }
}

// Perform a fade transition between the videos in activePlayer and nextPlayer.
Blend()
{
    nextPlayer.Play();

    // Changes Blend coefficient in a loop for fade transition from activePlayer to nextPlayer.
    if (currentBlend == 0.0f)
```

```
{
    for (float f = 1.0f; f >= 0; f = (f - incr))
    {
        skyboxMat.SetFloat("Blend", 1.0f - f);
        sleep(wait);
    }
    currentBlend = 1.0f;
}

else
{
    for (float f = 0.0f; f < 1.0f; f = (f + incr))
    {
        skyboxMat.SetFloat("Blend", 1.0f - f);
        sleep(wait);
    }
    currentBlend = 0.0f;
}
skyboxMat.SetFloat("Blend", currentBlend);

// Swap the current activePlayer and the nextPlayer objects.
activePlayer.Pause();
activePlayer.frame = 0;
VideoPlayer tempplayer = activePlayer;
activePlayer = nextPlayer;
nextPlayer = tempplayer;

// Change current location only if we are not doing a transition.
if (!playingTransition)
    currentLocation = nextLocation;

// Send messages to WalkInPlaceScript and Waypoints to allow new transitions.
if (endingTransition)
{
```

```
        walkScript.UnlockWaypoint();
        PathHotspotV2.UnlockWaypointRender();
        endingTransition = false;
    }
    CheckIfAddOverlay();
    switching_video = false;
}

// Start a full transition from one Node location to another when WIP is executed.
PlayOneShotVideo()
{
    playingTransition = true;

    // Wait for next video to be prepared.
    while (!nextPlayer.isPrepared)
        sleep(wait);

    switching_video = true;
    nextPlayer.Play();
    while (nextPlayer.frame < 1)
           sleep();

    // Call Blend to fade between current location video to the loaded transition video.
    StartCoroutine("Blend");
    while (switching_video)
       sleep(wait);

    // Stop transition video from playing
    activePlayer.Pause();
    // While the transition video has not completed yet, we will continue to check if step data is
        ↪ received.
    while (activePlayer.frame < activePlayer.frameCount)
    {
        sleep(wait);
```

```
        /*
            If localSpeed is positive, we have step data. If we have step data and are not currently
                ↪ playing transition, we should trigger playback of a few frames.
        */
        if (localSpeed != -1 && !playingWithStop)
        {
            long nextFrame = activePlayer.frame + (localSpeed * framesPerSpeed);

            // Only change target next frame if it is larger than the current frame count.
            if (nextFrame > activePlayer.frameCount)
            nextFrame = activePlayer.frameCount;

            // Start new instance of the coroutine to play transition video until target frame
            StartCoroutine(StopAtSpecificFrame(nextFrame,activePlayer));
            localSpeed = -1;
        }
    }

    ClearOverlayRenderTexture();

    // Prepare the new current location clip.
    PrepareNextClip();
    endingTransition = true;
    playingTransition = false;
    switching_video = true;
}

// Plays video in provided Video Player object until the provided end frame.
StopAtSpecificFrame(long endFrame, VideoPlayer player)
{
    player.Play();
    playingWithStop = true;
```

```
    // Continue playback until either the player frame exceeds target frame or video ended.
    while (player.frame < endFrame && player.isPlaying)
        sleep();

    player.Pause();
    playingWithStop = false;
}

// Called externally to indicate a WIP step occurred, indicating transition should continue.
public void SendStepData(float speed)
{
    if(localSpeed == -1 && playingTransition && !playingWithStop)
        localSpeed = speed;
}
```

Animations

RealNodes supports using preprocessed videos with a green "alpha" mask color to place the exposed video as a layer over the background (see Figure 3.9). This allows for animating partial regions of the video based on event triggers. For example, we can have a location with a closet that opens on command. $MultiVideoManager$ handles how these animations overlay on the other videos. When there is no animation for a waypoint, the Render Texture for *overlayPlayer* is cleared and set to pure green. This allows for the system to fully ignore the image data for animation and any commands to set the playback of the *overlayPlayer*. When there is an animation for the current waypoint, that animation video is added and prepared in the *overlayPlayer*. Below is pseudocode of the primary functions for handling the triggering and playback of an animation video:

```
// Main update thread. Checks if there is an overlay video to play.
void Update()
{
```

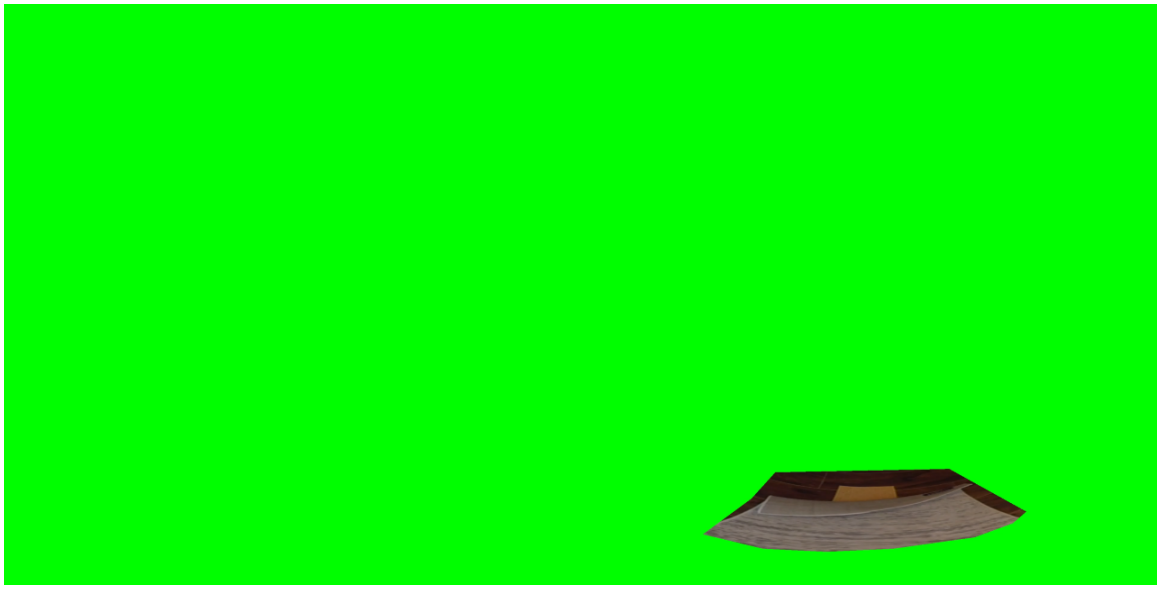

Figure 3.9: Example of an equirectangular image from one of the animation videos. The green mask covers everything except the desired animating feature of the environment.

```
    if (!playingTransition && switching_video && nextPlayer.isPrepared && !donePreparing)
    {
        donePreparing = true;
        StartCoroutine("Blend");
    }
    CheckOverlayPlayback();
}

/*
    Checks if a) not playing another overlay, b) current overlayPlayer frame is not the same as
        ↪ target frame. If true start a new PlayOverlay.
*/
private void CheckOverlayPlayback()
{
    if (!playingOverlay && overlayPlayer.frame != overlayTargetFrame)
    {
```

```
        StartCoroutine("PlayOverlay");
    }
}

// Plays the animation video up until a the target frame is reached or exceeded.
Coroutine PlayOverlay()
{
    playingOverlay = true;

    while(overlayPlayer.frame is negative)
        sleep();
    overlayPlayer.Play();
    while (overlayTargetFrame > overlayPlayer.frame)
    {
        sleep(frameRate);
    }
    overlayPlayer.Pause();
    playingOverlay = false;
}

// Called externally to set current animation frame, calculating based on a percentage progress.
public void SetOverlayVideoFrame(float progress)
{
    float targetVideoFrame = overlayPlayer.frameCount * progress;
    if(overlayTargetFrame != targetVideoFrame)
        overlayTargetFrame = targetVideoFrame;
}
```

SkyboxPanoramicBlended Shader

The *SkyboxPanoramicBlended* shader is an extension of Unity3D's standard *Skybox−Panoramic* shader. Both shaders take an equirectangular image and render it as a skybox

environment map around the player, adding an optional global color tint and exposure. It uses a standard algorithm for calculating spherical coordinates from an equirectangular image texture coordinates. Below is pseudocode for this calculation:

```
// Normalize coordinates
float3 normalizedCoordinates = normalize(textureCoordinates);
// Get latitude angle from coordinates
float latitude = acos(normalizedCoordinates.y);
// Get longitude angle from coordinates
float longitude = atan2(normalizedCoordinates.z, normalizedCoordinates.x);
// Calculate spherical coordinates from the two angles
float u = (longitude + PI) / (2.0 * PI);
float v = (latitude / PI);
return float2(u,v);
```

There are several additions in *SkyboxPanoramicBlended*. Rather than just one equirectangular image, it loads in three texture: two standard environment maps, and one overlay to the environment map. In addition to this, it takes in a blending coefficient. The fragment shader portion of *SkyboxPanoramicBlended* combines this data into the final environment map, allowing for both the fade and overlay effects described earlier in this section. The pseudocode describing this modified fragment shader is shown below:

```
fixed4 frag (v2f i)
{
    // Get spherical coordinates from vertex data.
    float2 textureCoordinates = ToSphericalCoordinates(i.textureCoordinates);

    // Load fragment colors from texture data.
    half3 c1 = tex2D(SphereTex1, textureCoordinates).rgb;
    half3 c2 = tex2D(SphereTex2, textureCoordinates).rgb;
    half3 c3 = tex2D(SphereTexOverlay, textureCoordinates).rgb;

    // Combine tint and exposure to use after other calculations are done.
```

```
    half3 mixColor = Tint.rgb * unity_ColorSpaceDouble.rgb * Exposure;

    // Subtract from the overlay image the mask color (hard-coded to pure green).
    half3 maskCheck = (c3 - half3(0.0, 1.0, 0.0));

    /*
        c3 is the overlay for the current view c1. It should blend against c2 the same way as c1. If
            ↪  the below check succeeds, then fragment of c3 is not green masked part, but data we
            ↪  want to use instead of c1. Hard-coded threshold chosen to allow for tolerance with
            ↪ fragments along the edge of the mask and the anilmation content in SphereTexOverlay.
    */
    if (maskCheck.g < -0.58)
    {
        c1 = c3;
    }

    /*
        Equivalent to "half3 c = c1 * (1.0 - Blend) + c2 * Blend;". Blends percentage of c1 and
            ↪ inverse percentage of c2 to fade between the two images.
    */
    half3 c = c1 + Blend * (c2 - c1);

    // Combine with the mixColor to get the final fragment color
    c *= mixColor;
    return half4(c, 1);
}
```

Walking-in-Place Implementation

The user navigates with a WIP system. In RealNodes when the user faces a waypoint and begins to make steps, the transition video for that path begins to play. At this point, playback only progresses with each step. This continues until the transition video ends and

the user is now shown the video for the new waypoint (see Figure 3.10). Visual guidance UI and WIP can be enabled/disabled by toggling *NavigationMode*, done by pressing the *Grip* button on the controller. The feature of allowing WIP only when the mode is activated and facing a navigable path was intended to reduce false-positive steps.

Figure 3.10: Performing WIP controls playback of transition videos, creating the illusion of navigating the environment.

SensorDataManager

The *SensorDataManager* script is the central collector and provider of tracking data for the HMD and the left and right controllers, as well as button input states for both controllers. It contains a reference to the position transform of these three objects, such that any call to *SensorDataManager* for the position and rotation data of these objects will always contain the most accurate information. it primarily provides sensor data to two scripts. *ViveCursor* uses the HMD rotation data as part of its ray casting logic to detect

which waypoint is being faced. *WalkInPlaceScript* then uses both the HMD position data for the step detection calculation, as well as the *Grip* button state to toggle *NavigationMode* on and off.

WalkInPlaceScript

Our *WalkInPlaceScript* is inspired by the process described by Lee et al. [20]. It tracks sinusoidal motion of the HMD over an input window. It counts steps when a lower peak occurs within a recognition range, correlating with a step. *MultiVideoManager* converts step data into playback frames for the current transition video. This allows WIP to realistically match playback, creating the illusion of walking in 360 VR.

The main *Update* thread runs every frame of the RealNodes application (90 FPS). On each loop, it checks if the player should be moved, based on whether there have been any steps taken. Then it captures the current raw HMD y axis position and processes it through the Recognition step. *WalkInPlaceScript* primarily provides step data to the *MultiVideoManager* to control transition video playback. Additionally, it sends notifications to waypoint objects through *PathHotspotV2* to indicate a waypoint should prepare for transition. Inputs to *WalkInPlaceScript* are primarily raw HMD position data and *Grip* button input data from *SensorDataManager* to toggle *NavigationMode*. Additionally, it receives messages from *ViveCursor* and *MultiVideoManager* to control *NavigationMode* related states.

There are two phases of the WIP process: the Calibration step and Recognition loop. The Calibration consists of two major steps: calculating the central axis of the HMD WIP cycle based on the user's default position and applying a recognition range based on that central axis for allowing the WIP to occur. The central axis is a computation based on the Y axis height of the HMD and the X axis rotation to account for variance in head pitch. Below is pseudocode for the algorithm and functions related to the calibration calculation:

```
// Calculate central axis of sinusoidal motion and recognition range for WIP.
void Calibrate()
{
    isCalibrated = false;
    centralAxis = ComputeCentralAxis();
    recognitionRange = ComputeRecognitionRange();
    isCalibrated = true;
}

// Computes "centralAxis = centralAxis_initial + C * sin(xRotation * PI/ 180)".
float ComputeCentralAxis()
{
    float xRotation =GetHmdRotation().x;

    // Get initial centralAxis, which is initial y position.
    float yPosition = GetHmdPosition().y;

    // Check pitch of X rotation to decide which constant C to use for the central axis formula.
    float c = cDown;
    if (0 <= xRotation && xRotation < 90)
        c = cUp;

    // Return central axis computation based on xRot bounds.
    return yPos + c * sin(xRotation * PI / 180);
}

/*
    Returns recognition range. Currently hard-coded to a delta assuming an approximately 1.78 meter
        ↪  person as described in paper by Lee et al. Future plan to calcuate difference between
        ↪ top peak of yPos and H for accurate calibration.
*/
float ComputeRecognitionRange()
{
```

```
        return delta * cSpacing;
}
```

Once Calibration is calculated, we can start the Recognition loop. The Recognition loop collects raw y axis position data from the HMD, which is then averaged with a sliding k-size window filter. The results are inserted into an n-size queue. The n and k values are such that $n > k$, $k \leq 1$, and $n \leq 3$. The oldest value of either queue is discarded when the queue is too full for new data. When the filtered queue is full, it is checked to determine whether the bottom peak of a sinusoidal motion has been achieved, indicating a step has occurred. This happens when the middle position is the smallest value and the absolute value of the difference between smallest and largest value is more than the recognition range. Once a step is detected, we increment steps and increase a movement time value to aid in calculating how long WIP will occur for that step. On the next call to *Update* we check if WIP time has completed. If not, *MultiVideoManager* will be sent step information. Below is the pseudocode describing the algorithm and functions related to the recognition process:

```
public int n;
public int k;

// Make raw queue size big enough so k size window can filter enough to produce up to n results.
int rawSize = n + k - 1;
int midIndex = nHalf + 1;

/*
    Main update thread. Performs calibration if not done yet. Primarily performs step recognition.
*/
void Update()
{
    CheckInput();
    MovePlayer();
```

```
    /*
        If calibrated, feed current yPosition into step recognizer. If not, do calibration routines.
    */
    if (isCalibrated && showWaypoints)
        StepRecognizer(GetHmdPosition().y);
    else
        Calibrate();
}


/*
    Check Grip input to see if Navigation Mode should be enabled/disabled.
*/
void CheckInput()
{
    if (LeftInput().gripButtonDown || RightInput().gripButtonDown && canGrip && !lockWaypoint)
    {
        showWaypoints = !showWaypoints;
        if (showWaypoints)
            PathHotspotV2.UnlockWaypointRender();
        else
            PathHotspotV2.LockWaypointRender();


       // Cooldown to temporarily block toggling Navigation Mode
        StartCoroutine("GripCooldown");
    }
}


/*
    Core step recognizer function.
*/
void StepRecognizer(float yPosition)
{
    // Take new input yPosition and insert into continuous recognizer queue
    rawStepQueue.Enqueue(yPosition);
```

```
// Dequeue items out of raw queue if needed
while (rawStepQueue.Count > rawSize)
    rawStepQueue.Dequeue();

// Filter the current queue with the k-size window and insert into the filtered queue

if(rawStepQueue.Count >= kWindowSize)
{
    // Move window through all elements of the signal.
    int maxSize = n - nHalf;
    if (rawStepQueue.Count < n)
        maxSize = rawStepQueue.Count - nHalf;

    for (int i = kHalf; i < maxSize; ++i)
    {
        float result = 0f;
        for (int j = i - kHalf; j < i + kHalf; ++j)
        {
            result += rawStepQueue[i + j];
        }

        // average out result
        result *= oneOverK;
        filteredStepQueue.Enqueue(result);
    }
}

// Dequeue items out of filtered queue if needed
while (filteredStepQueue.Count > n)
    filteredStepQueue.Dequeue();

if (filteredStepQueue.Count == n)
{
```

```
        float midItem = filteredStepQueue[midIndex];


               // Start to iterate through the current filtered queue
        float min = recognitionRange;
        float max = -recognitionRange;

        for (int i = 0; i < n; ++i)
        {
            float val = filteredStepQueue[i];

            if (val < min)
                min = val;
            else if (val > max)
                max = val;
        }

        // Check if center is min and difference between max and min larger than recognition range.
        if (min == midItem && max - min > -recognitionRange && max - min < recognitionRange &&
            ↪ isMovementAllowed)
        {
        // If it is, increment step and trigger movement of player
         IncrStepCount();

         // Set a movement time to start moving in a given direction
            endTime = Time.time + journeyTime;
        }
    }
}

/*
    Send step data if before end time. Disable Waypoints for WIP if not already done.
*/
void MovePlayer()
{
```

```
    if (Time.time < endTime)
    {
        manager.SendStepData(speed);

        if (currWaypoint != null && !lockWaypoint && isMovementAllowed)
        {
            lockWaypoint = true;
            nodes.BroadcastMessage("DisableWaypointRender");
            currWaypoint.SendMessage("GoToPath");
        }
    }
}

/*
    Called externally to set current active Waypoint object.
*/
public void setWaypointObj(GameObject obj)
{
    if (!lockWaypoint)
    {
        currWaypoint = obj;
        if(currWaypoint != null)
            currWaypoint.SendMessage("PreparePathClip");
    }
}

/*
    Called externally to clear currently set active waypoint.
*/
public void UnlockWaypoint()
{
    // Cooldown to temporarily block WIP
    StartCoroutine("MoveAllowedCooldown");
    currWaypoint = null;
```

```
    lockWaypoint = false;
}
```

Visual Guidance User Interface Methods

We implemented four different visual guidance UIs into RealNodes. The purpose of the visual guidance UI is to indicate where waypoint objects are in the environment and when a user can perform WIP to navigate to those locations. In the current version of RealNodes, only one of the visual guidance UI is available at runtime. The visual guidance UI is chosen through project configuration settings prior to runtime. This dictates which support objects are active in the scene depending on the current chosen UI. We implemented four different visual guidance UIs into RealNodes. The purpose of the UI is to indicate the location off waypoint objects in the environment. Depending on what visual guidance UI is configured for runtime dictates which support objects are active in the scene. Visual guidance UI display logic is primarily dictated by the waypoint objects surrounding the user, and what direction the HMD is facing. Based on the angle the HMD is facing, UI graphics clearly indicate the visible waypoints (based on the selected UI type). This facing behavior controls rendering of the UI and signals $MultiVideoManager$ to prepare and execute a video transition sequence. This is performed by signaling the $MultiVideoManager$ to load the specific transition video. Then, if $NavigationMode$ is on and WIP is detected, $MultiVideoManager$ executes the transition. When $NavigationMode$ is toggled on or off, callback messages are broadcast to waypoint objects to toggle rendering of visual guidance graphics. The WIP system sends notifications to prepare for transition with that waypoint when WIP is occurring. $MultiVideoManager$ turns off rendering of visual guidance graphics based on whether video transition playback is occurring. The remaining part of this section presents detailed explanations of each UI and how they were implemented.

Target

The Target method indicates waypoints with a square shaped, semi-transparent target aligned with the ground plane. The floor Target is inspired by waypoints in conventional VR, often used with a teleport-style method of navigation [19]. Each waypoint has a Target object that renders in the Target UI mode. The waypoint neighobor objects are placed in the environment such that they match with the location of the next waypoint that will be traveled to if WIP is performed. See Figure 3.11 for an example of this placement. In this picture, the next waypoint is in the hallway in front of the door. This allow the Target to act as an absolute location indication for where a player can go.

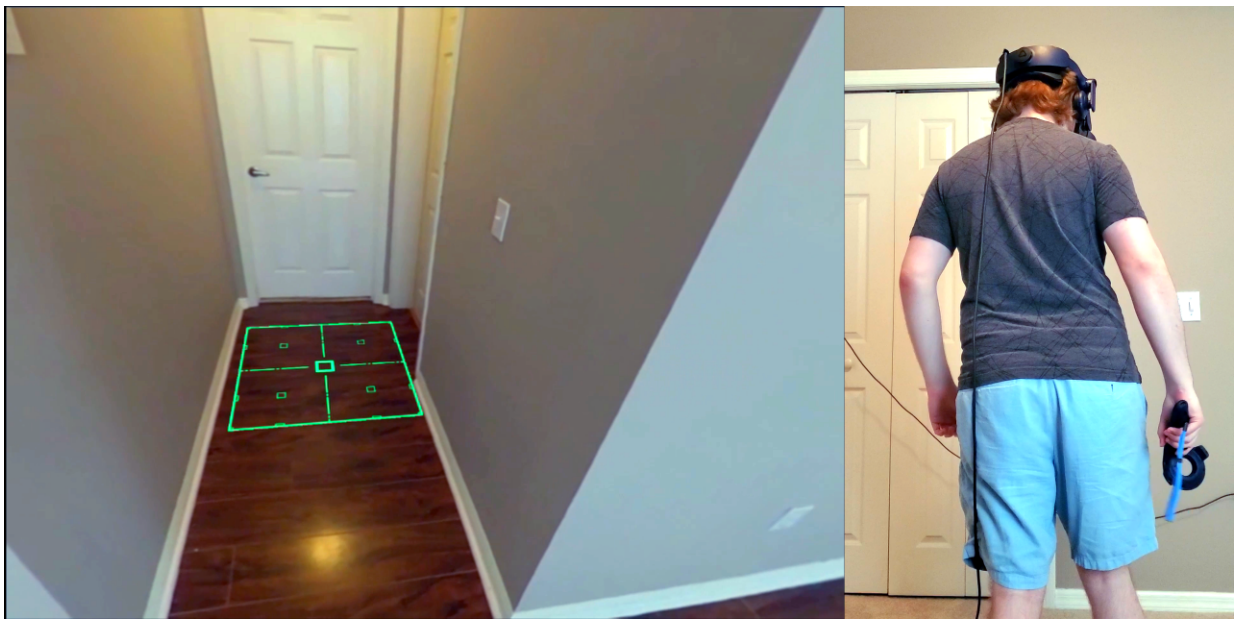

Figure 3.11: Example of Target UI.

Ripple

The Ripple method indicates waypoints with a diamond shaped floating marker exhibiting a semi-transparent "ripple" visual effect. We were curious about how a guidance method with a "distortion" effect would affect engagement in a positive way, or if it would negatively impact usability. Each waypoint object has a Ripple object that renders in the Ripple UI mode. The Ripple objects are placed in the environment such that they match with the location of the next waypoint that will be traveled to if WIP is performed. See Figure 3.12 for an example of this placement. In this picture, the next waypoint is located on the carpet next to the table. The Ripple acts as an absolute waypoint marker in this way.

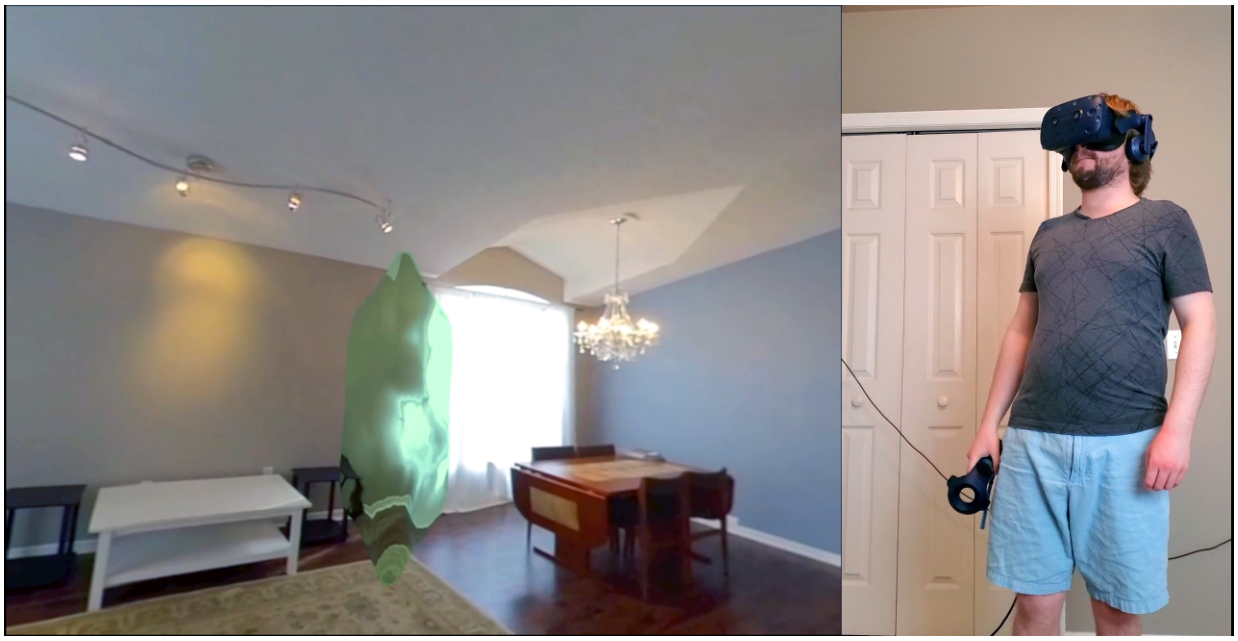

Figure 3.12: Example of Ripple UI.

The Ripple UI is animated with a combination of *MaskedRipple* shader and *SimpleTextureScroll* script. See Figure 3.13 for a high-level illustration of the pipeline of the Ripple UI rendering. *MaskedRipple* is an extension of Unity3D's standard *GlassStained-*

BumpDistort shader. Both shaders take a normal map texture and a distortion coefficient as input, which controls a transparent distorted effect. We add a black and white culling mask texture, which acts as a stencil on the render. Any fragments where the culling mask is black are not rendered, while those that are white on the culling mask are rendered normally by the shader. This gives the Ripple UI its hexagonal shape. The *SimpleTextureScroll* is what causes the animation effect. On every frame, it updates the x and y texture offsets of the normal map texture in the *MaskedRipple* shader. It changes these offsets based on public variable scrolling coefficients, multiplied by a Unity3D API call to a frame time call. This allows for a smooth scrolling effect of the normal map texture, creating an effect like rippling water floating in the air. By default, the x and y coefficients are set to 0.5, which makes the texture scroll diagonally down and left.

Path

The Path method indicates the direction of a waypoint with a semi-transparent "lane" aligned with the ground plane and originating from the user. The Path was inspired by the idea of showing a direct line-of-sight path from the user's current location to the waypoint, partly inspired by visualizations from Tanaka et al. [38] of ground-plane based indicators of where a user can navigate. Attached to the virtual HMD object is a Path object. This is the core object for rendering the Path UI "lane". If the player is gazing at a waypoint, Path will render. See Figure 3.14 for an example of the Path UI. In this picture, it is pointing in the direction of the next waypoint, heading into the kitchen. Attached to the object is the *SimpleFixRotation* script, which rotates the Path in the same direction as the HMD along the Y axis but keeps its other rotation axes fixed such that the path is parallel to the ground plane.

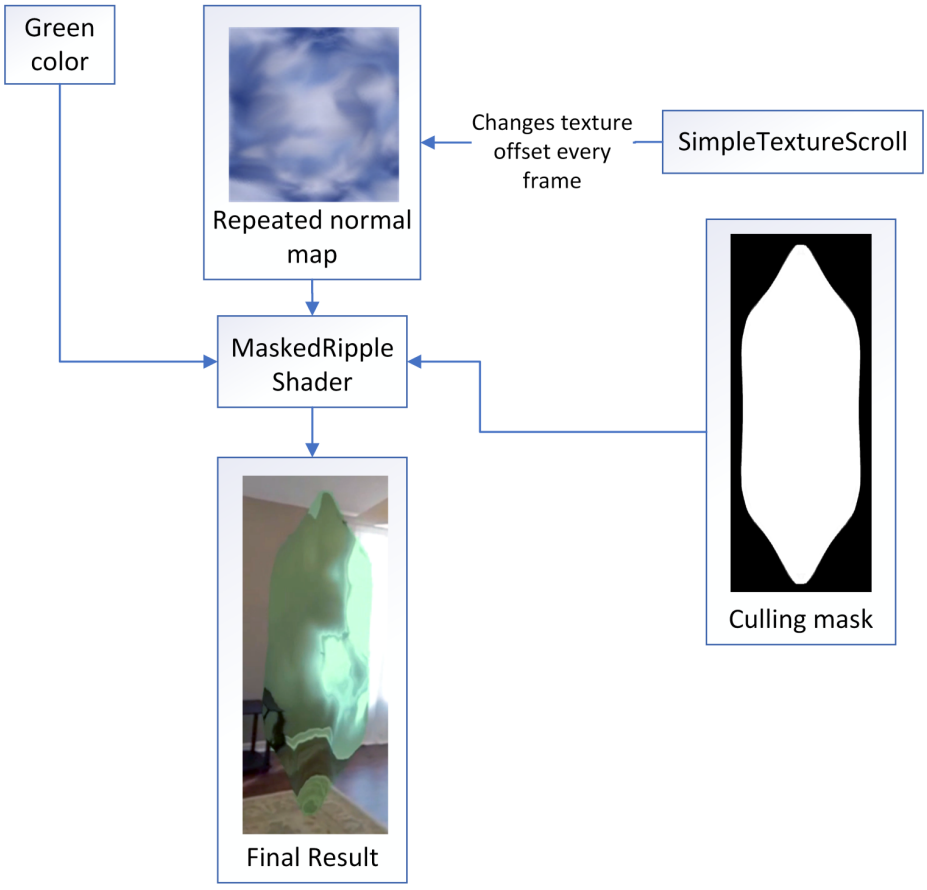

Figure 3.13: High-level pipeline illustrating the MaskedRipple shader pipeline, which creates the final animation of the Ripple visual guidance UI.

Arrow

The Arrow method indicates the directions of a waypoint with an arrow formed with a Bezier curve and an arrowhead. Both components actively and smoothly curve towards and point at the nearest waypoint based on shortest rotation distance from player to waypoint. See Figure 3.15 for an example of the Arrow UI, indicating in the picture the direction of the next waypoint is the carpet by the window. When *NavigationMode* is active, the Arrow is always being rendered. When user is facing a waypoint, the Arrow changes from blue to

green. An example of both the curving and color change behavior are shown in Figure 3.16. The implementation of Arrow is inspired by guidance methods explored in the work of Lin et al. for indicating points of interest in standard 360-video [21].

Figure 3.14: Example of Path UI.

The core Arrow object is made of several sub-objects to control its appearance and behavior. It contains two child objects: a Bezier curve object that uses a line renderer to draw the curved line, and an arrowhead object. A companion script changes the color of the Arrow UI based on whether the user is directly facing a waypoint. These are controlled by three scripts: *BezierCurve*, *PointArrow*, and *ArrowColor*. *BezierCurve* controls the position of the control points of the Line Renderer that draws the Bezier curve of the Arrow. To match the behavior of *BezierCurve*, *PointArrow* calculates the closest waypoint target by rotation angle between player and waypoint, then smoothly rotates and translates the arrowhead towards that waypoint. Finally, *ArrowColor* changes the color of the Arrow UI based on whether the player is facing a waypoint.

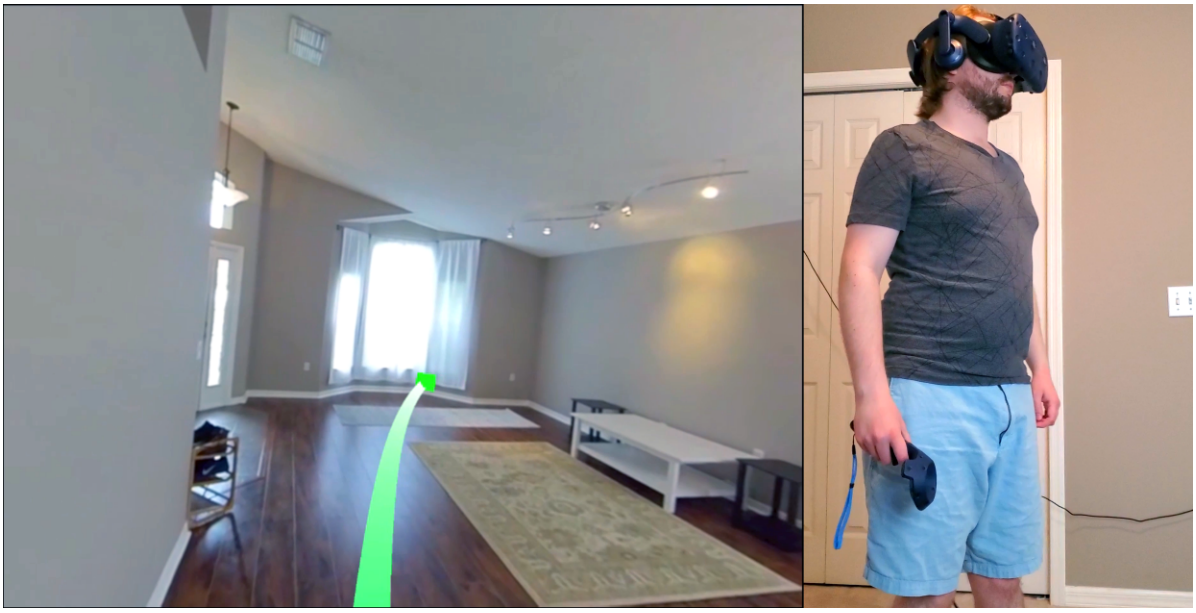

Figure 3.15: Example of Arrow visual guidance UI, indicating the direction of the closest waypoint in relation to rotation distance.

PointArrow smoothly changes both the position and rotation of the arrow such that, for a given target waypoint, it both rotates to point at it and it shifts its position in space so it is between the player and the waypoint. Every frame, it iterates through all active waypoint objects for the current waypoint and calculates the rotation angle between it and the player. When the smallest angle is found, the waypoint is set as the new target. Then, the desired target rotation and position are calculated, and a smooth interpolation is performed across multiple frames of time. This results in the arrowhead turning towards the waypoint and influencing the Bezier curve of the Arrow UI.

BezierCurve controls the Line Render drawing of a quadratic Bezier curve defined by three points. In RealNodes, two of these control points are attached to the virtual HMD object. One of these points is near the center of the user, while the other is a fixed distance in front of the user. This is such that no matter what direction the HMD is facing, the

second point is always directly in front of the HMD at a fixed distance. The last point is the arrowhead object. Because the arrowhead shifts left or right of the user based on its target waypoint, it has the strongest influence on the Bezier curve. This results in the Arrow UI smoothly and actively indicating where the nearest waypoint is for the user. Below is pseudocode for the Bezier curve calculations done in this script:

Figure 3.16: Arrow curving to nearest waypoint, changing color when nearly facing it.

```
// Called to calculate the quadratic Bezier curve points for the Line Renderer to use.
private void QuadraticBezierCurve()
{
    // Calculate points on a quadratic Bezier curve defined by the three control points.
    for (int i = 1; i <= pointCount; i++)
    {
        float t = i * 1 / pointCount;
        positions[i - 1] = QuadraticBezierPoint(t, p0, p1, p2);
    }
    lineRend.SetPositions(positions);
}

// Calculate point on quadratic Bezier curve defined by three control points at specified time.
private Vector3 QuadraticBezierPoint(float t, Vector3 p0, Vector3 p1, Vector3 p2)
{
```

```
    float x = 1 - t;
    float t_sq = t * t;
    float x_sq = x * x;

    return x_sq * p0 + 2 * x * t * p1 + t_sq * p2;
}
```

Interaction Objects

RealNodes is capable of presenting interaction objects in the environment. These span from virtual objects that are realistically lit based on the 360 VR environment, to support tools that manipulate the 360 VR environment. For the base interaction logic in RealNodes, we leverage the SteamVR interaction system scripts. This allows for events to occur based on whether the virtual controller object is hovered over the object and when the *Trigger* button is pressed to simulate grabbing an object.

Action Tool

As described in previous sections, RealNodes can animate individual portions of the 360 VR environment. To trigger these animations, we added an *ActionTool*, a purely virtual linear drive switch. It appears as a golden sphere on a green rail. The golden sphere part can be grabbed by the player by hovering the virtual controller over it and holding the *Trigger*. While the *Trigger* is held, the player can freely slide the sphere along the rail. These *ActionTool* objects are placed in locations where animations can be triggered. These animations can reveal hidden objects, such as the collectible *Page* object. This effect is used in the House Scenario developed for our experiment (explained in the section Scenario Development). It includes animations such as opening a drawer/book and lifting a portion of a rug. See Figure 3.17 and Figure 3.18 for examples.

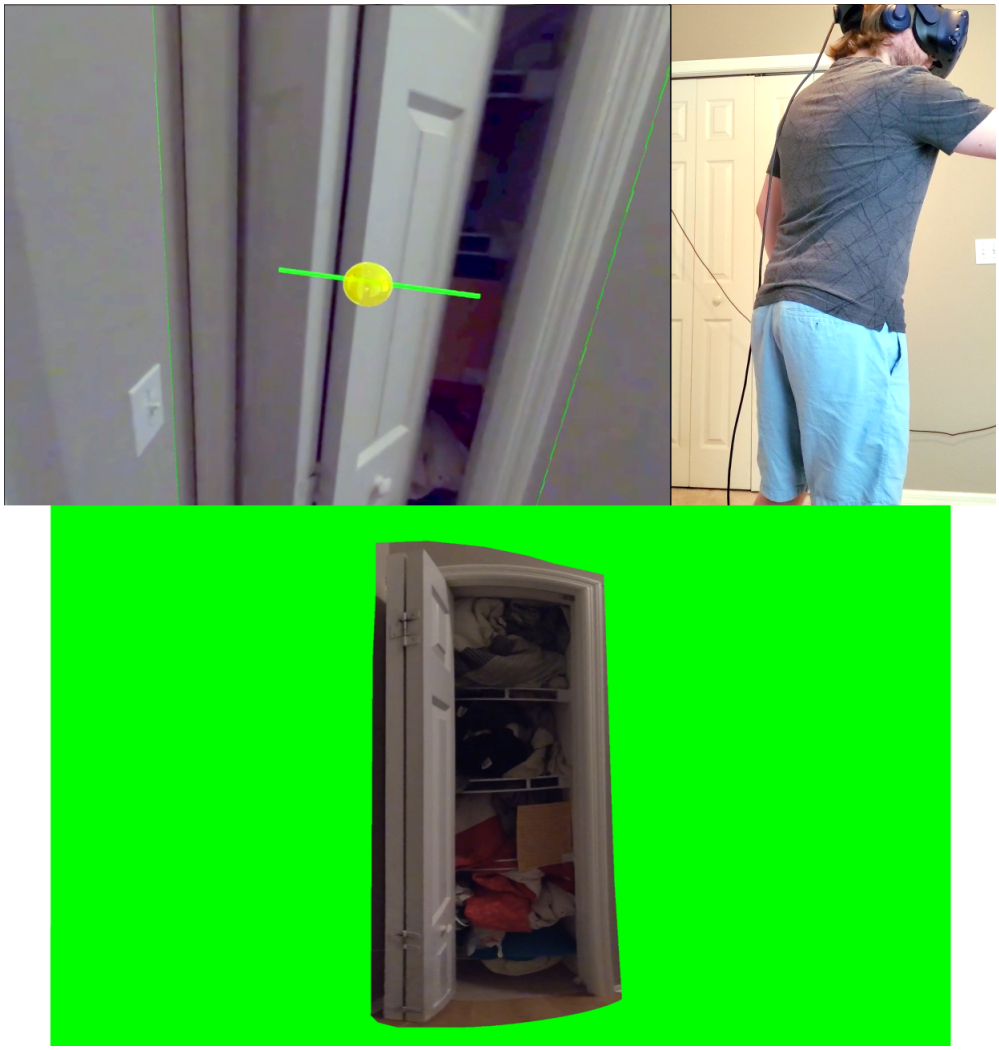

Figure 3.17: Footage of closet being opened with Action Tool, showing how the masked video shown below it is applied to the background and manipulated.

The script *ComplexSwitch* is attached to each *ActionTool* and controls the state logic of the switch. Its primary purpose is to track the position of the switch, knowing what percentage it is between the designated start and end positions on the rail. It signals the $MultVideoManager$ every time there is a change to this position, which indicates that frames of the animation should play until a specific stopping frame. The effect of this is

a smooth playback of the animation tied to how much or little the switch is moved. As a usability feature, it holds references to the player to control the position of the switch, so it is always between the location where the animation is and the player, closer to the player and near shoulder level height. This allows it to be reached by the player. Once the animation has been completed, it spawns the *Page* object associated with that animation.

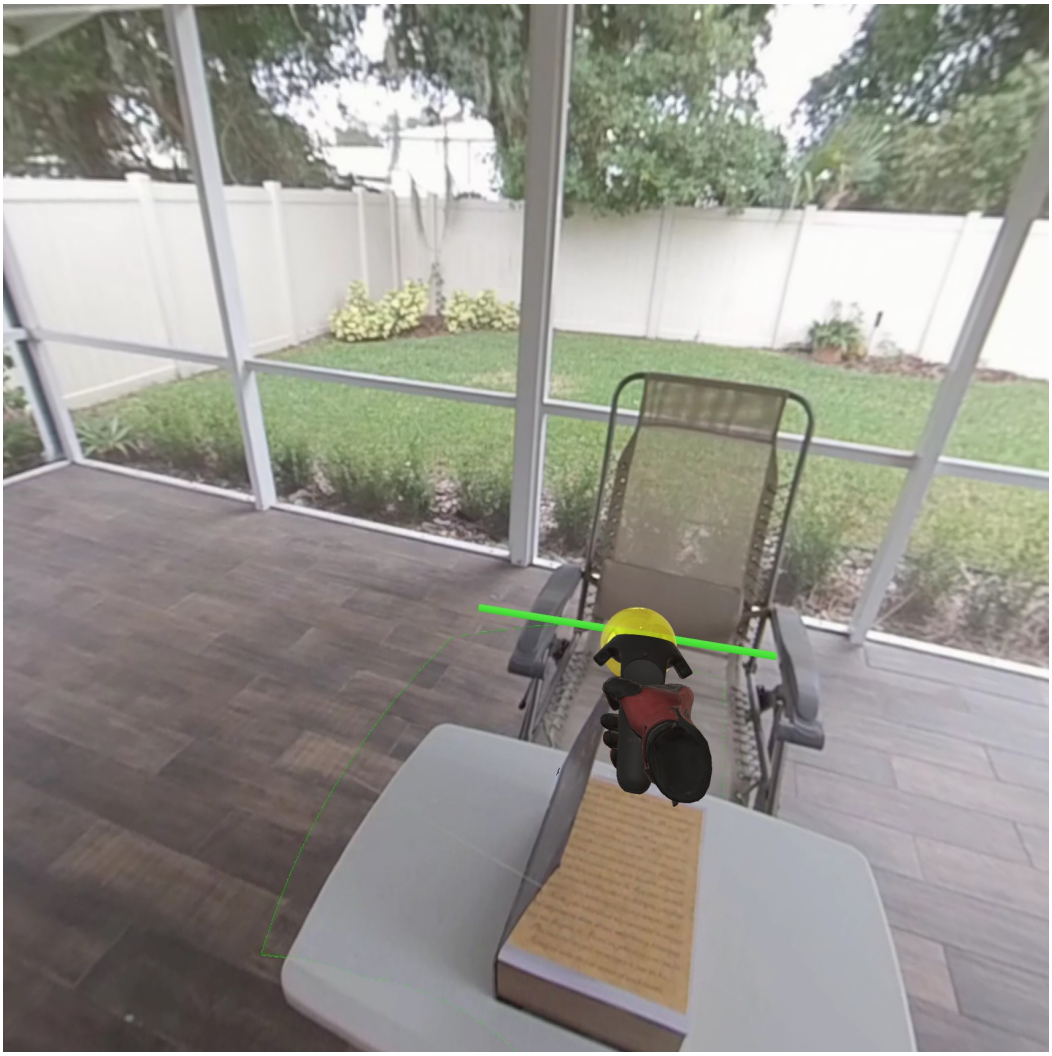

Figure 3.18: Sliding the switch on the Action Tool across the rail manipulates the 360 VR environment, such as opening a book.

Collectible Page Objects

Finding and collecting a *Page* object is the completion goal of the scenario. When an *ActionTool* is fully switched and a background animation has completed, it will appear in the environment close to the animation. Once it appears, it will float towards the player until it is close enough to be in arm's reach. It can then be collected by the player by hovering the virtual controller over it and pressing the $Trigger$. See Figure 3.19 for an example of a page appearing after an *ActionTool* is triggered.

Each *Page* object has a *SpawnPageAnimated* script. The two purposes of this are to trigger an animation for the page to float towards the player, and to execute the scenario exit conditions when grabbed by the player. On each update frame of the floating animation, it linearly interpolates a smooth trajectory towards the player. It does this until it is a specified distance from the player, at which point it stops animating. When the page is collected, it calls a *CollectPage* routine. This first destroys the *Page* object in the environment to simulate collecting it. It then sends a signal to $MultiVideoManager$ to call the function $FadeAndExit$. This causes the $MultiVideoManager$ to do a fade transition to a black screen and quit the application.

Metrics Recording

RealNodes records metrics to CSV log files for experimental data collection purposes. Two logs are produced for each scenario run: *SensorLog* and *EventLog*. The $MultiVideoManager$ script writes to the Event Log every time a new transition is performed. Every time a new transition is started, a new row is written to the Event Log. When the scenario is ended, the function *OnApplicationExit* adds an additional row to the Event Log and saves the file. *SensorDataManager* contains an update thread on a fixed schedule (around 50 FPS) that records the tracking and button state data to the *SensorLog*.

Figure 3.19: Left is original state of drawer and Action Tool; right is after activating Action Tool, with drawer open and Page object revealed.

Each row contains the rotation and position data of the HMD, and the rotation, position, and button states for both the left and right controller. When the scenario is ended, the function *OnApplicationExit* saves the file. Both logs reuse the same custom *CsvBuilder* script. This is provided externally with name and header information to define which log it is writing. It then passively receives new row data to write. It includes helper functions to create a current timestamp string. The file remains open in memory in the application until the *Save* function is called, which writes out the collected row data strings and writes them to the file.

Scenario Development

Developing a 360 VR scenario from start to finish is an interdisciplinary process that requires skills and expertise in film, video editing, emerging camera hardware knowledge,

computer graphics, and software development. It is a challenge to develop a scenario for RealNodes since many steps require both manual authoring and software modifications. In the future, more friendly automated authoring tools that do not require software modifications for the core system are desired for future work. Nevertheless, it is a feasible process to make a new scenario for RealNodes in relatively short time, especially when planned thoroughly ahead of time. The reusable scripts and Prefab objects we created for RealNodes additionally make it a manageable process.

Figure 3.20: Various scenes from the House Scenario developed for RealNodes.

For our pilot study, we developed the House Scenario. We captured video across several locations inside a house (see Figure 3.20. We imported these videos from their native proprietary formats to common MP4 format, then edited them as necessary. These video assets were then imported into the RealNodes project assets. Then a new scene was constructed with our custom objects manually created and placed in the scene. Finally, the

few scripts that control scenario-specific information were modified to define the final 360 VR scenario environment and its behavior. The total time to develop the House Scenario took about 3 days of labor, with preproduciton and filming taking one day, and the rest (editing, waypoint placement, minor code changes) taking the remaining time. A similarly sized environment would likely take a similar amount of time to construct a new scenario.

Planning and Filming

To efficiently make a 360 VR scenario involves a planning step that is like pre-production on a video or film. There are aspects of location scouting to determine an area that fits for the need. For example, we used an indoor environment freely available to us, allowing for flexibility in filming without concerns for time or weather. Once the location was chosen, the rough map of the scenario needed to be created. This map includes not just individual waypoint locations, but also available connecting edges for those waypoints, as well as planned animation locations. This defined the constraints that a node-graph based scenario would have to adhere to. Figure 3.21 shows a map of the environment locations. From the above data, we created a list of all necessary shots. There were three kinds of shots: Locations, Transitions, and Actions. Not counting test filming sessions, the final scenario shoot produced 68 shots in a day, of which 55 were used. In total, 13:38 minutes of video are included in the scenario.

Locations are longer videos that needed to be filmed so they could be edited to seamlessly loop. This is because they must infinitely play around the player. In instances where an actor was present in the background, their motions were planned so that they roughly matched a looping sequence (walking back and forth from a window, reading a book on the couch, looking around before returning to a neutral position, etc.). By using the Insta360 One X app, these recordings could be done remotely to avoid having a cameraperson in the shot, adding realism.

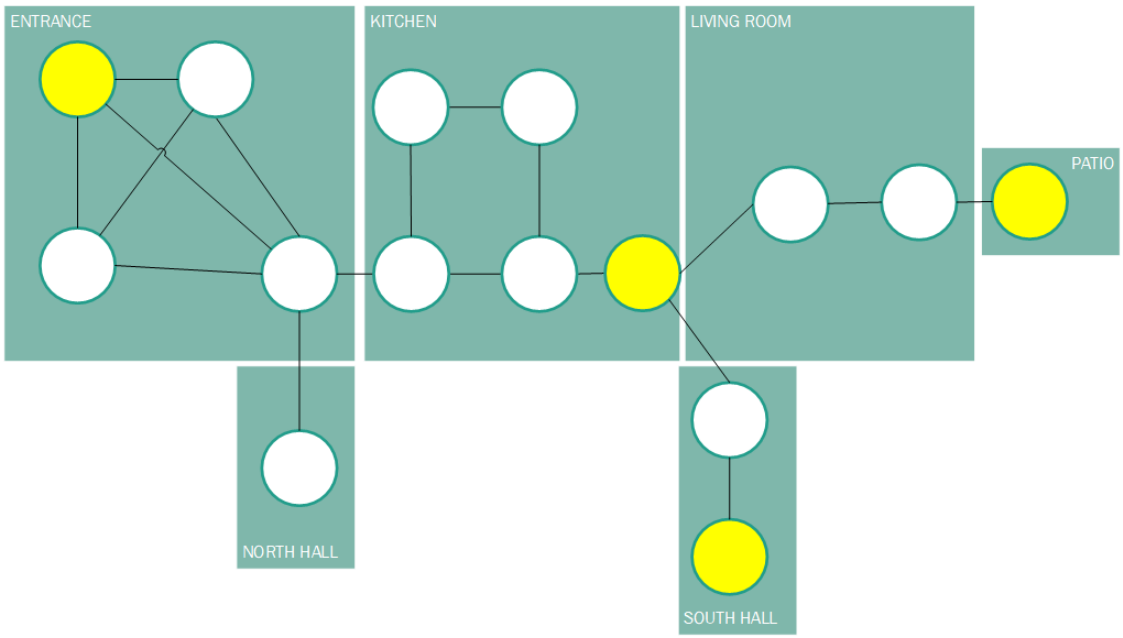

Figure 3.21: Simple map showing House Scenario environment. Circles in node graph indicate locations that could be navigated to. Yellow circles indicate locations of hiding spots.

Transitions are videos that show the camera being walked directly from one waypoint location to the other. We wanted to avoid having a cameraperson present in these shots, concerned it would detract from the intended realism of walking in the environment. To mitigate this, two separate transition videos were filmed for each connecting edge, one for each direction the player could walk on that edge. This allowed the cameraperson to remain behind the camera motion so the direction the player is likely looking in the 360 VR environment is unobstructed.

Actions are short videos that can be repurposed for our overlay animation system. These involved filming an object in the environment moving, lifting, or opening. Filming Action videos needed to be done right after filming the corresponding Location video to keep the tripod position the same for each, to make the eventual overlay accurate. To avoid

having a cameraperson in the planned final shot, we used the common practical effect trick of tying or hooking a string on the end of the object we wanted to animate. We recorded a video of the object being moved by someone pulling on the string from a distance.

Importing and Video Editing

Videos produced by the Insta360 One X camera are in a proprietary packaged format (.insv, or Insta360 Panorama video). These files need to be exported in the Insta360 Studio, a free software package (Figure 3.22). This software can perform simple edits on clips, apply stabilization algorithms, and calibrate stitching before finally encoding and exporting to MP4. We needed it to trim and export every clip before further edits could be made. This process was enough to finish editing for Location videos.

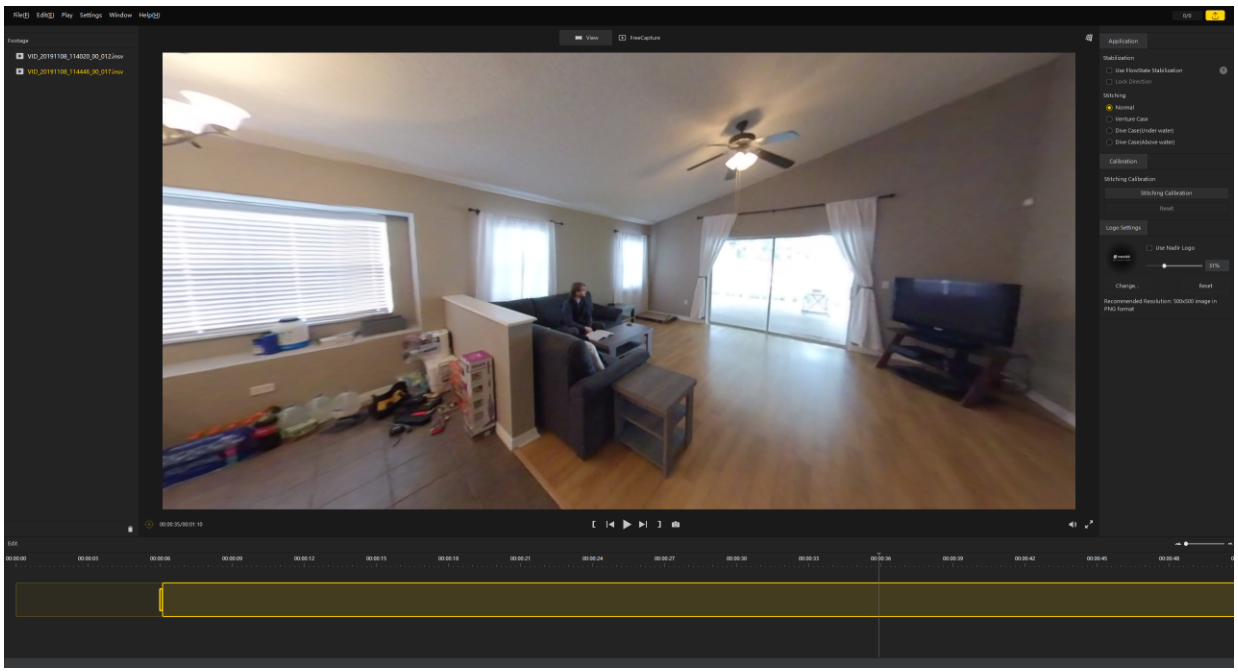

Figure 3.22: Insta360 Studo software, for exporting from proprietary video format to MP4.

All Transition and Action videos were edited in DaVinci Resolve video editing software

after being exported from Insta360 Studio (see Figure 3.23). Both video types required masking a region of the clip. This was done by using Resolve's Color menu, which besides color correction options allows for adding a custom mask shape over the shot. This mask can either be used to show only what is inside or outside the mask.

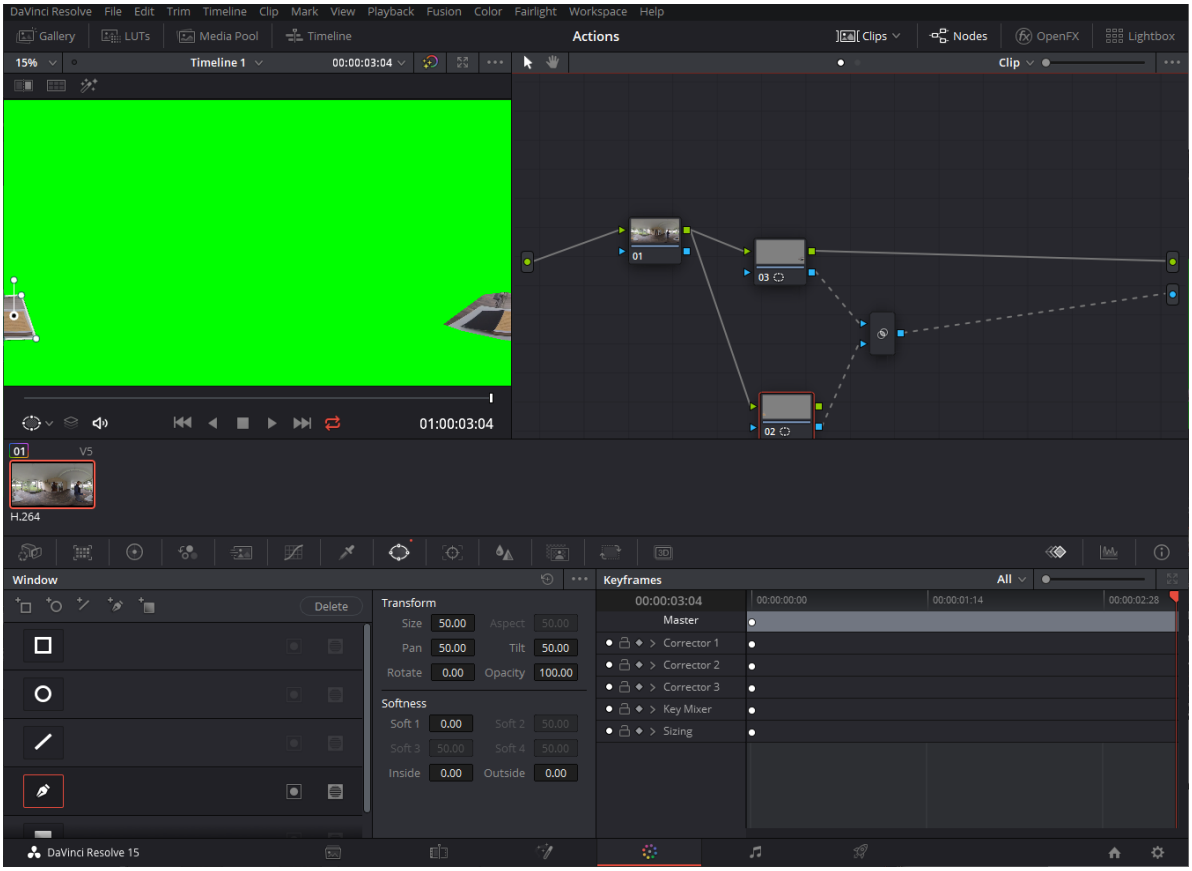

Figure 3.23: View of DaVinci Resolve video editing software. This image shows the editing pipeline for adding a mask over the video data.

For Transition videos, since the cameraperson appears in a portion of the video, they were manually edited out leaving a black region. This was done by creating a mask that covered the tight bounds of where the cameraperson was at any time in the shot. This mask leaves a black region in the video where the cameraperson would be (see Figure 3.24). It

was decided that this artifact was a reasonable compromise for realism compared to leaving the cameraperson in, under the assumption that most players will perform a WIP transition while facing forward, or away from where the black region will be.

For Action videos, we wanted to show only the region that was meant to be animated. We applied a mask that covered the tight bounds of where the moving object is. Everything else outside this mask was colored green, successfully hiding the person pulling the string (see Figure 3.25). This green region is treated as an "alpha" color by our shader to allow just the animating part to be overlaid on the corresponding Location videos.

After all editing is finished, we import the videos into the RealNodes Unity3D project. Though videos can be dragged-and-dropped into the asset hierarchy, we chose to transcode all videos to 75% of the resolution (2880 * 1440). This was to address performance on minimum spec hardware for running our user study.

Manual Creation of Node, Waypoint, and Interaction Objects

To create a new scenario, a new Scene is created in the project. In this case, we created $House\_4K$ to define the House Scenario. The amount of work to create a new RealNodes scenario is reduced in part since most scene hierarchy objects are the same between all RealNodes scenarios (refer to Scene Hierarchy subsection of the section Software Design Overview). The only unique object that needs to be populated is $Nodes$, the collection of waypoint objects in the environment. For each waypoint location to be represented in the scenario, a new object must be made. Each must contain neighbor waypoint objects representing where in the video environment waypoint locations are relative to current location. These neighbor objects are a reusable Prefab object that already contains the support structures and scripts for working with the WIP and visual guidance UI systems.

The custom steps that are needed to modify these are the physical placement in the environment and setting public references in the object for which waypoint it is traveling from

and to. Figure 3.26 shows an example of two waypoint objects that were manually placed in the scene to correspond with where they lead to. This development process is partially sped up because rather than requiring waypoints to be accurately placed in the scene, our system swaps out waypoints and their neighbor objects in the environment relative to player centered position. This means only waypoint objects need to be accurately manually positioned. After waypoints are set up, *ActionTool* and *Page* objects can be added (see Figure 3.27). The *ActionTool* should be placed in between where the animation is planned and where the player is, however this does not need to be perfect since $ComplexSwitch$ automatically moves the switch to an appropriate place near the player and between these two points. The *Page* however needs to be place manually near where an animation is planned to occur in the environment, to allow it to animate correctly away from that animation to the player.

Update State Transitions in Scripts

The final steps to make the scenario for RealNodes involves modifications to both external variable references for objects in the scene hierarchy and modifications to the few scripts that have scenario specific data. $MultiVideoManager$ has public definitions in the editor for its three video arrays: one for Location videos, one for Transitions, and one for Actions. All three of these need to be filled manually with reference to the imported videos.

This is also the place where the start location is set for the scenario. $CommonEnums$ is where the enumeration indexes are for all the Location, Transition, and Action videos are stored, for use by the other scripts. These indexes must be modified with labels and indexes that match the same indexes in the arrays in $MultiVideoManager$.

Finally, two functions need to be modified in $MultiVideoManger$. $GetNextTransition$ has a switch statement that acts as a state machine that, based on the current Location and next Location, will get the appropriate enumeration for the next appropriate Transition video. This needs to be modified to. $CheckIfAddOverlay$ has a switch statement that

acts as a state machine for determining if the current Location has an accompanying Action video. Both functions need to be modified to match the scenario node-graph of all possible transitions in the environment as well as which Location nodes have accompanying animations.

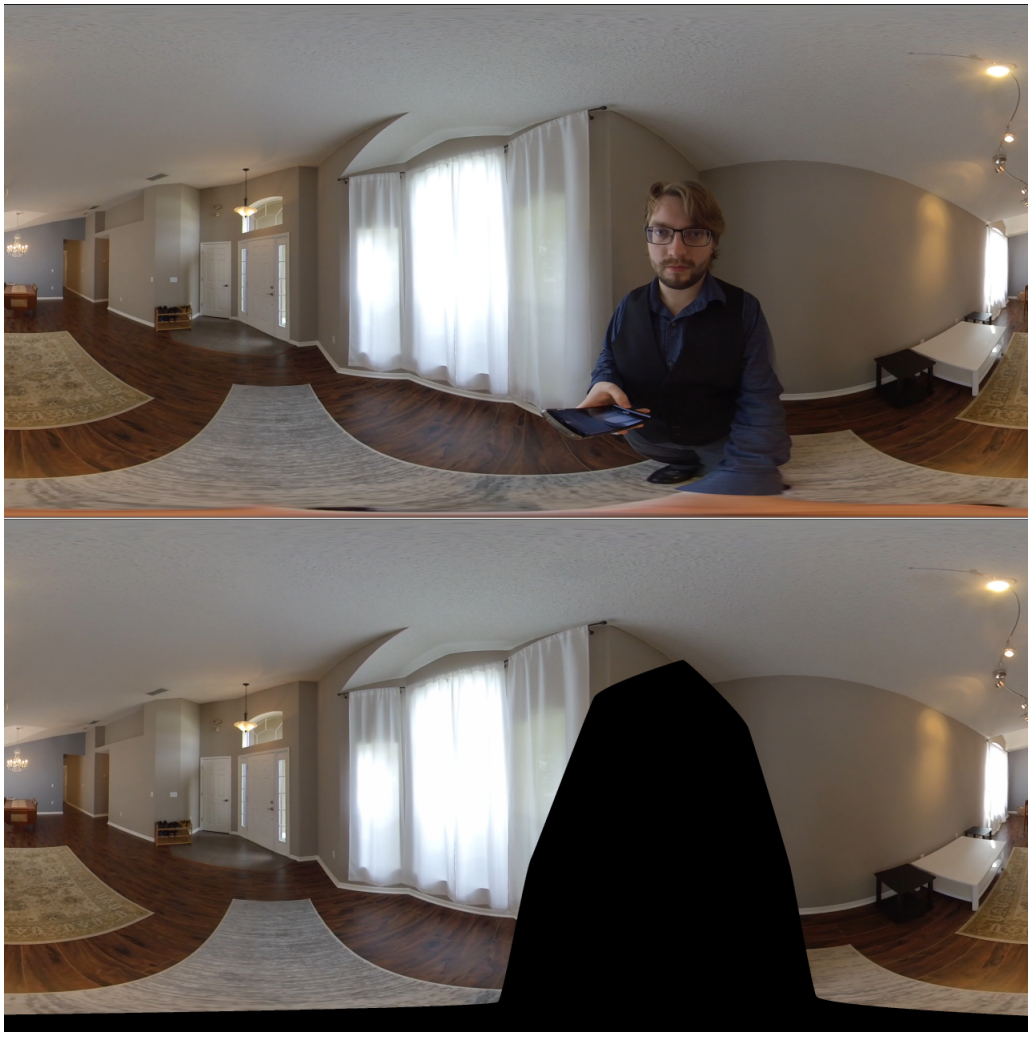

Figure 3.24: (Top) Equirectangular frame from transition video before editing showing the cameraperson. (Bottom) same frame after a mask was added to crop the cameraperson out.

Figure 3.25: (Top) Someone pulling a string to lift the carpet up. (Bottom) Same as above, except all but the lifted carpet is masked in green.

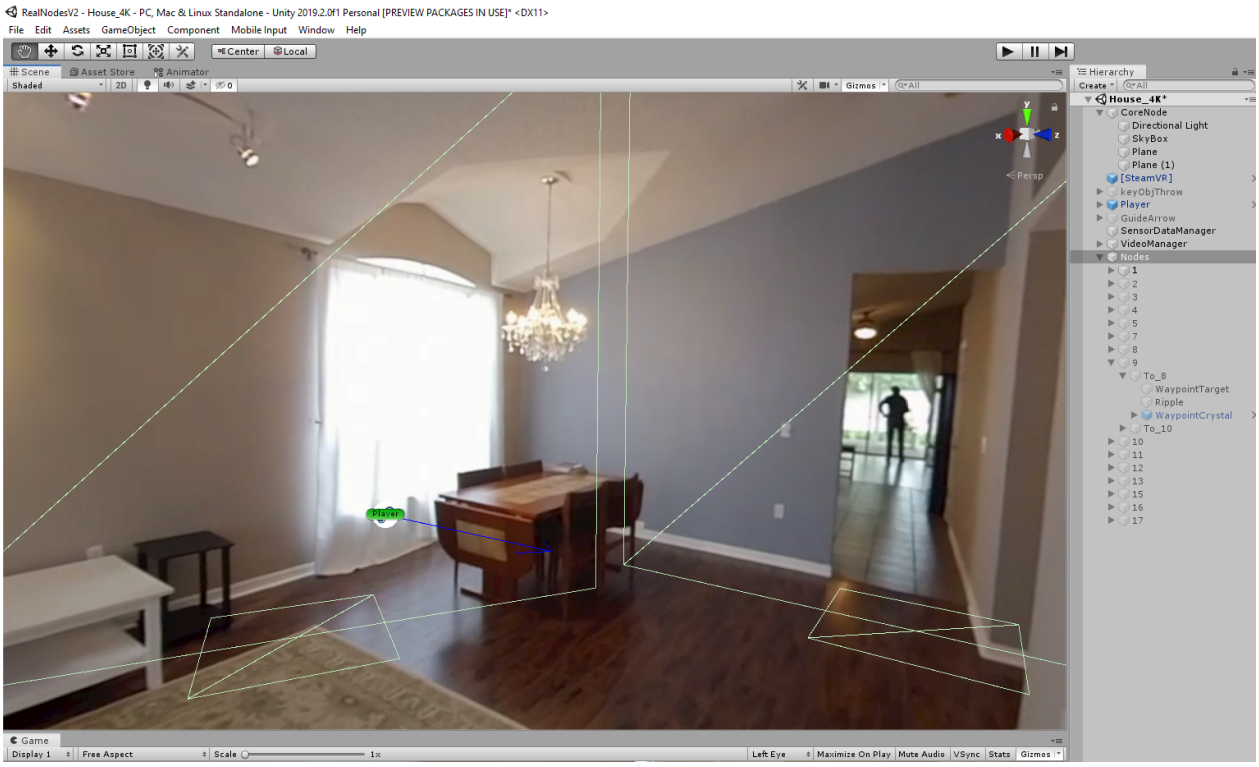

Figure 3.26: Wireframes of the Waypoint objects in Unity3D editor. Note the large quad perpendicular to the ground plane. This is the target surface for raycasting to detect which waypoint is being faced.

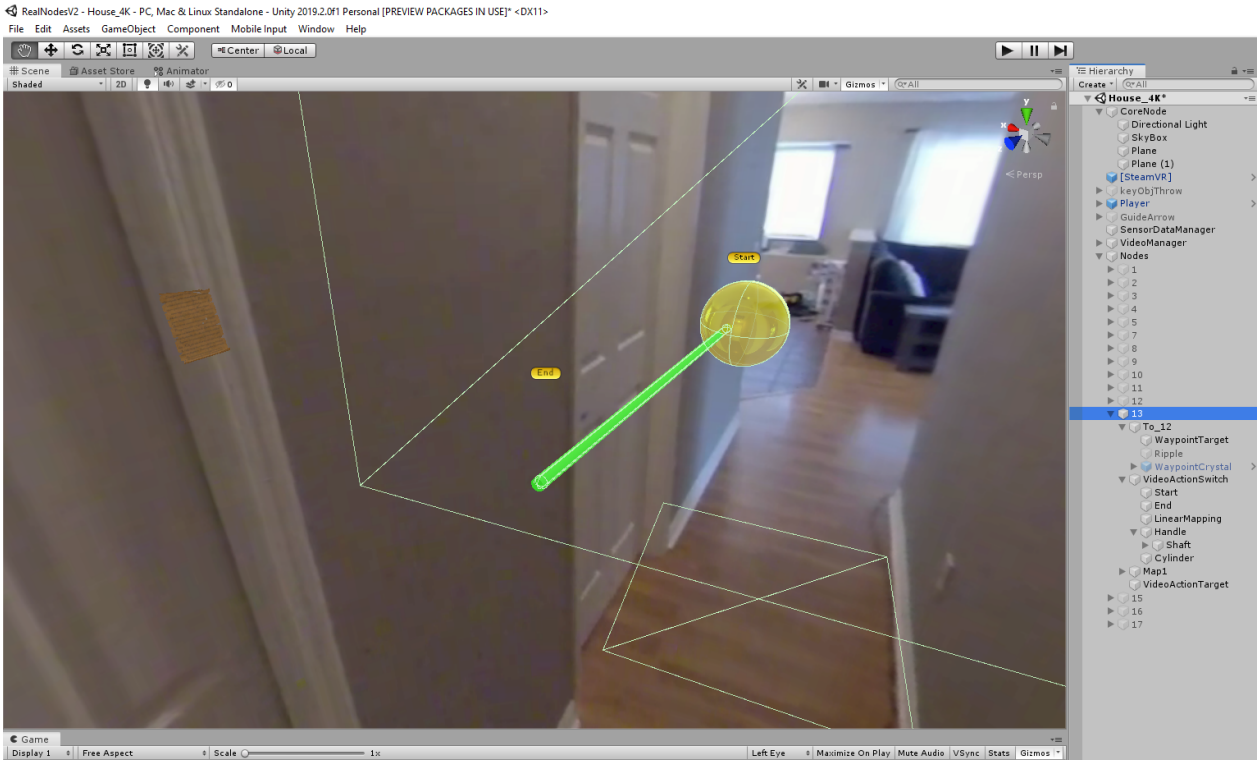

Figure 3.27: Action Tool and Page objects in the editor environment in front of a Waypoint object.

CHAPTER 4: PILOT USER STUDY

A pilot study was carried out using the House Scenario in RealNodes to determine the effect of different visual guidance UIs on people's user engagement measures, the presence or lack of simulator sickness symptoms, and user performance in a 360 VR experience. The study was performed with 24 people from a volunteer pool from a local university's students, faculty, and staff. The participants were only included if they were physically able to stand in place while wearing a virtual reality HMD and use a wand style game controller, and jog in place for a duration of a few seconds. The gender breakdown was 18 males and 6 females. Out of 24 participants, 17 wore glasses/contacts. The age breakdown was 9 people age 18-21 years old, 9 people age 22-25, 4 people age 26-29, and 2 people age 30-32. On a five point Likert scale of how often participants played virtual reality games, 7 said Never, 11 said Rarely, 6 said Sometimes, and none said Frequently or Always. The education breakdown was 12 with high school degree, 8 with an undergraduate degree, and 4 with a graduate degree. The RealNodes VR software was run on a Windows 10 desktop computer in a lab setting, equipped with an Intel Core i7-4790K CPU, an NVIDIA GeForce GTX 1080 GPU, and 16 Gigabytes (GB) of RAM. The user was equipped with an HTC Vive HMD and one Vive Controller.

Procedure, Design, and Analyisis

The procedure is as follows. The participant was asked to fill out a demographic survey to answer questions about experience with video games, VR, and 360 images or video. They were instructed on how to use the HTC Vive HMD and controller in RealNodes. The participant was instructed on the concept of WIP and how they can use it to navigate the environment. They were told how to do tasks with hand gestures like grabbing and pulling,

and how to switch *NavigationMode* on/off.

For each condition, they navigated the environment to find a hidden *Page*. They were told the only way to uncover it was to manipulate an *ActionTool*. Once the *Page* was grabbed the scenario would end. There were four scenarios, each with a unique visual guidance UI and unique hiding spot. After each condition, they filled out two questionnaires: Simulator Sickness Questionnaire to determine the absence or presence of simulator sickness, nausea, and ocular motor issues (SSQ) [18], and User Engagement Scale Short Form (UES-SF) to determine perception of user engagement and subscores for Aesthetic Appeal (AE), Perceived Usability (PU), Focused Attention (FA), and Reward (RW) [29]. After the last scenario, they filled out a UI preference questionnaire, where 1 = most preferred, and 4 = least preferred.

Two logs were produced for each scenario run: Sensor Log and Event Log. Sensor Log has position and rotation tracking data from the VR headset and controller, including button presses, recorded with timestamps at a rate of 50 frames per second. Event Log has timestamps of when a participant navigated to a new area, including a label for when the scenario ended.

The study was organized into four conditions presented to each participant in a within-subjects manner. Each condition had a unique visual guidance UI presented to the participant and a unique hiding spot for the participant to find. UI order was randomized across subjects in a counterbalanced manner such that we tested all 24 possible permutations.

A Shapiro-Wilk test was performed on the Preference, SSQ, UES SF, and completion time data to determine normality. Since the data was found to be non-normalized, it was decided to analyze the data using non-parametric tests. A Friedman test was performed on the SSQ, UES SF, and Completion times data to determine presence of statistically significant differences between conditions. When a Friedman test found an asymptotically significant result, we performed post-hoc analysis using a Wilcoxon Signed Ranks Test on

all possible pairs of conditions to determine where the significance was, followed by a Holm's sequential Bonferroni adjustment to correct for type I errors. For all our statistical measures, we used $\alpha = 0.05$.

Results

We can report that the different visual guidance UIs have a statistically significant effect on Completion Times, indicating that the Arrow was significantly better for users to complete the scenario quickly. We additionally found that the Arrow had the highest average scores in all the total scores and most of the sub scores in preference, simulator sickness, and user engagement, though they were not statistically significant differences.

Completion Times

We found a statistically significant difference in completion times after performing a Friedman test ($\chi^2(3, 24) = 12.75,\ p < 0.005$). After performing post-hoc analysis using Wilcoxon Signed Rank Test on all possible pairs, a significance was found with two of the pairwise tests: Arrow compared to the Path($Z = -3.686,\ p < 0.001$) and Ripple compared to Path ($Z = -2.029,\ p < 0.05$). Table 4.1 shows the test statistics for this.

Table 4.1: Results of Wilcoxon Signed Rank Test on all possible condition pairs.

Test	Result
Target-Ripple	$Z = -0.743,\ p = 0.458$
Target-Path	$Z = -1.914,\ p = 0.056$
Target-Arrow	$Z = -1.4,\ p = 0.162$
Ripple-Path	$Z = -2.029,\ p < 0.05$
Ripple-Arrow	$Z = -0.771,\ p = 0.44$
Path-Arrow	$Z = -3.686,\ p < 0.001$

After performing a Holm's sequential Bonferroni adjustment against actual signifi-

cance threshold, only the Arrow to Path pairwise was found to be significant and the Ripple to Path pairwise test was not, indicating that there was only a significant difference found between Arrow and Path ($Z = -3.686,\ p < 0.001$).

The mean and standard deviations for completion times (seen in Figure 4.1) for each condition match up with this data (time in seconds): ($Target : M = 189.729,\ SD = 100.887, Ripple : M = 194.6,\ SD = 105.929, Path : M = 312.878,\ SD = 206.606, Arrow : M = 152.825,\ SD = 82.377$). The completion time for Arrow is fastest on average while Path is slowest on average (taking more than twice as long). This seems to indicate that the Arrow is easier to get accustomed to and use to effectively search an environment, and Path is slower to understand and to use moment-to-moment.

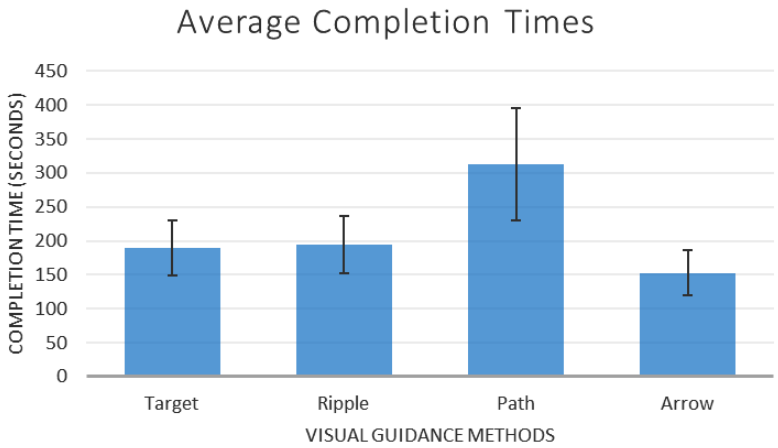

Figure 4.1: Average scenario completion times (in seconds) with 95% confidence error bars (lower is better). There is a significant difference between Arrow (fastest average), and Path (slowest average).

Preferences

We found no statistically significant difference in preference ($\chi^2(3, 24) = 3.65,\ p = 0.302$). We found the following average Preferences (as seen in Figure 4.2) (lower is better):

($Target : M = 2.29,\ SD = 0.999, Ripple : M = 2.88,\ SD = 1.076, Path : M = 2.58,\ SD = 1.1389, Arrow : M = 2.25,\ SD = 1.225$). The Arrow was on average preferred but was very close in preference to Target, possibly indicating a split between preference of a more direction/angle-based guidance in Arrow and an absolute location guidance in Target.

SSQ

We found no statistically significant difference in SSQ total score ($\chi^2(3, 24) = 2.404,\ p = 0.493$), Nausea sub score ($\chi^2(3, 24) = 1.451,\ p = 0.694$), or Oculo-motor sub score ($\chi^2(3, 24) = 4.274,\ p = 0.233$). Table 4.2 lists the mean and standard deviations for the SSQ scores, and Figure 4.3 shows a bar graph visualization. For simulator sickness, Arrow had the least average effect in general and for all subcategories.

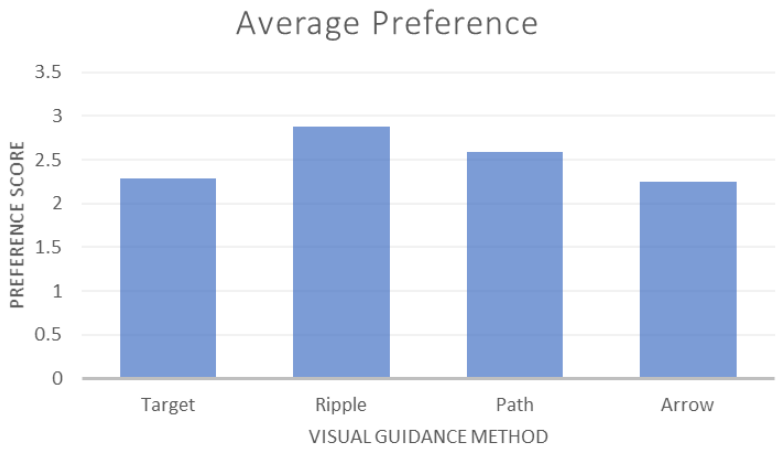

Figure 4.2: Mean Preference (lower is better). Arrow is lower than Target, but by only a small margin.

UES SF

We found no statistically significant difference in UES SF total score ($\chi^2(3, 24) = 3.967,\ p = 0.265$) and all sub scores: FA ($\chi^2(3, 24) = 6.745,\ p = 0.08$), AE ($\chi^2(3, 24) = 4.432,\ p = 0.218$), PU ($\chi^2(3, 24) = 2.86,\ p = 0.414$), and RW ($\chi^2(3, 24) = 2.814,\ p = 0.421$). Table 4.3 lists the mean and standard deviations for the UES SF scores, and Figure 4.4 shows a bar graph visualization.

Table 4.2: Mean and standard deviation of SSQ scores and sub scores. T=total score, N=Nausea, O=Oculo-motor (lower is better).

Score	Target	Ripple	Path	Arrow
N	M = 2.6 SD = 1.794	M = 2.65 SD = 2.621	M = 2.4 SD = 2.206	M = 2.35 SD = 2.263
O	M = 2.79 SD = 1.81	M = 2.48 SD = 1.351	M = 2.46 SD =1.501	M = 2.27 SD = 1.624
Total	M = 2.73 SD = 3.239	M = 2.58 SD = 3.707	M = 2.42 SD = 3.257	M = 2.27 SD = 3.77

Arrow had the highest average effect on overall user engagement, AE, PU, and RW. The only score Arrow was not the highest in was FA, with Path being the highest average effect. This possibly indicates a difference in conditions that allowed or required more attention in Path compared to the others.

Discussion

We developed RealNodes to investigate creating interactive and explorable 360 VR, and to perform a study to determine what effect choice of visual guidance had on user experience. We successfully implemented RealNodes and four visual guidance methods: Target, Ripple, Path, and Arrow. We completed a pilot study, providing a statistically significant finding that Arrow is faster for users to learn and use, contributing to faster task completion times.

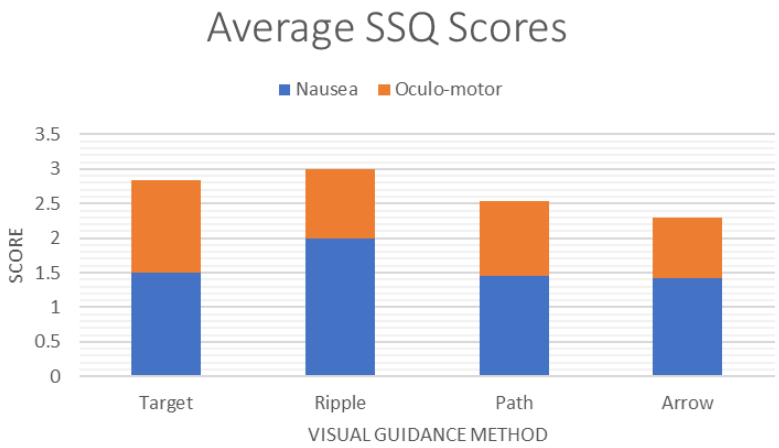

Figure 4.3: Average scores for SSQ, including sub scores (lower is better). Arrow has lowest scores in total and sub scores.

Table 4.3: Mean and standard deviation of UES SF total scores and sub scores (higher is better).

Score	Target	Ripple	Path	Arrow
FA	M = 2.23 SD = 1.054	M = 2.21 SD =0.861	M = 2.83 SD = 0.735	M = 2.73 SD = 0.927
AE	M = 2.35 SD = 0.948	M = 2.31 SD = 1.023	M = 2.46 SD = 0.832	M = 2.88 SD = 0.784
PU	M = 2.46 SD = 0.624	M = 2.23 SD = 0.917	M = 2.56 SD = 0.778	M = 2.75 SD = 0.581
RW	M = 2.5 SD = 1.054	M = 2.31 SD =0.809	M = 2.4 SD = 0.671	M = 2.79 SD = 0.72
Total	M = 2.42 SD = 0.795	M = 2.19 SD =0.711	M = 2.52 SD = 0.644	M = 2.88 SD = 0.607

Participants liked how Arrow smoothly curved towards the nearest waypoint, giving continuous feedback on waypoint locations. Additionally, participants liked how it changed color when they could do WIP. One participant described it as "feeling good for exploration". Another described it as "a combination" of showing direction and location compared to other UIs. Again, Arrow had best average scores in Preference, SSQ, and UES SF (with the exception of FA subscore, discussed below), which aligns with the positive feedback and completion

time measure. Though not statistically significant, there seems to be some possibility that Arrow is better out of the UIs for engagement and simulator sickness measures. Though future work is needed to understand this trend in the data.

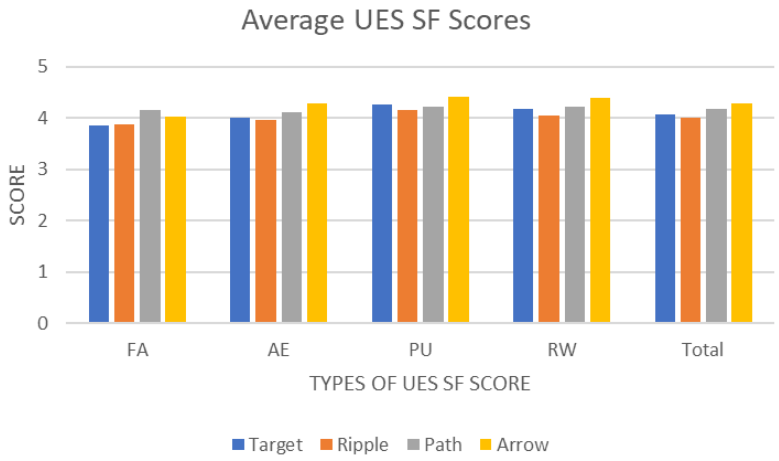

Figure 4.4: Average scores for UES SF sub scores and total scores (higher is better). Arrow had the highest scores in all categories except FA, where Path has the highest.

Meanwhile participants claimed Path was unclear for how they could navigate and gave less information compared to Arrow. An interesting trend was noticed for UES SF with Path beating Arrow at FA. FA is described by the authors of the UES SF test as indicating "cognitive absorption and flow" and losing track of time [29]. This could be interpreted negatively, meaning the difficulty of using Path required more focus.

Some participants liked Target over Ripple because it showed exact waypoint location. Target clearly indicated where on the floor the waypoint was, meaning it was easier for participants to tell where they would be standing next.

For some participants, Ripple looked less like a waypoint and more like a collectible item, or a portal which would lead to a whole new area outside of the home. This contributed to it feeling more distracting for these users.

As mentioned before, trends show Target and Arrow close in Preference, but notably ahead of the others. Target preference was higher than Ripple, and Arrow preference was higher than Path. This implies a split preference between subclasses of our UIs: direction/angle-based guidance in Arrow/Path and absolute location guidance in Target/Ripple. Participant testimonies did indicate that some preferred Target over Arrow because it clearly indicated the location where they would be going next, while others felt the smooth directional information was preferred over static Target markers. Our current measures and analysis do not clearly indicate what the reason is for this split preference, indicating other measures are needed to uncover what could result in this difference.

CHAPTER 5: FUTURE WORK

The implications of the results provide new work to be done. Many of the scenario development steps in our research remain manual and specialized. Further research into authoring tools is needed, especially asset management of videos and animations for ease of scenario creation. An authoring tool with drag-and-drop capability to combine video and virtual assets into scenarios equivalent to RealNodes House Scenario is ideal. Felinto et al. [10] developed a production framework for overlaying virtual objects over 360 images, showing some work has been done on this problem. But further investigation is needed.

Besides 3D reconstruction, 3D computer vision techniques can be leveraged for the authoring dilemma. For RealNodes, we manually created transition videos in each direction on a connecting path. This manual process is tedious and discourages creation of complex scenarios. There is a need to further investigate novel view synthesis that could use principles of 3D computer vision to generate novel transition videos using just two location videos. Methods of 3D reconstruction using epipolar geometry and computing a fundamental matrix for two standard perspective views has existed in the literature for some time [14, Chapters 9-10]. Fusiello and Irsara [12] describe a pipeline that starts with two or more uncalibrated images and through a series of steps (feature matching, calculation of epipolar geometry, stereo matching, and rectification) produces a "video" sequence of images between the two views. In theory this principle could work on two 360-videos to produce novel 360-video transitions. This would reduce the current need for manual transition videos for each possible path. However, the epipolar geometry calculation must be handled differently for a 360-degree panorama compared to a standard perspective image(different fundamental matrix calculations, etc.). An appropriate method needs further investigation.

Future studies are needed to make generalizable claims about the effect of navigation and visual guidance cues on user experience. Larger populations and deeper analysis of de-

mographic differences such as gender are needed. We can improve our navigation techniques based on refined hypotheses with more focused designs. Though our choice of WIP was motivated by existing state-of-the art for 360 VR navigation, we should also compare navigation techniques (Teleport, WIP, or automated scenario with no navigation) to determine if effects are due to UI guidance method or navigation method (or lack of it). Additionally, we can choose to deemphasize the current set of measures to instead focus on those missing from our current study, primarily cognitive load and accuracy measures. More scenario types are needed. Some participants indicated a desire to see different kinds of environments besides the House Scenario. Developing a variety of scenarios is worth investigating for richer experiences, as well as doing even more to avoid learning effects over time, aiding in generalization of findings.

Refinements of the UIs we developed should be explored. Based on observed absolute location and directional preferences, we can develop further iterations or hybrids of the visual guidance UIs from RealNodes. For example, there could be a user interface that uses multiple visual guidance UI elements at once, such as having Arrow to guide the player towards a Target to show exact location of waypoint. The benefits of the Arrow could be merged with the ideas from Path to make a hybrid "curving Path" that dynamically points at the next waypoint.

Our current system for indicating WIP start and end can be improved. Though there are exiting visual cues (UI disappears/reappears at the start and end of a WIP), this was not enough for some participants to know they were "done" with the WIP in a quick and clear way. A solution is a UI metaphor indicating progress during the walk, such as a progress indicator or an absolute waypoint getting closer as the cycle proceeds.

There is room to investigate immersive character interactions in 360 VR. Some participants in our study were interested in a "full game" like the House Scenario with a larger narrative. They reacted favorably to the background character in the videos, asking "Who

is he?" and saying he reminded them of horror/thriller games. RealNodes is designed to present videos based on event triggers, making future scenarios with branching storylines and decision making possible. However, best practices for integration of real actors into 360 VR to allow immersive interaction with characters need further investigation. Pope et al. [32] found that actors give less personal space to a 360-camera standing in for an actor compared to a real person when performing the same play. This indicates a need for more investigation of Proxemics (consideration of personal space) for designing natural feeling character interactions in 360 VR.

CHAPTER 6: CONCLUSION

We sought to develop a 360 VR system geared towards navigation and interaction, and to determine if the choice of visual guidance UI in 360 VR exhibited a significant difference in the user's experience. We successfully contribute to the literature RealNodes to facilitate development and presentation of 360 VR scenarios with explorable and interactive environments. We successfully contribute to the literature a set of four visual guidance UI elements for use in 360 VR to be refined and iterated on in future applications. We additionally provide the results of a pilot study of our four UIs, providing preliminary results indicating using our Arrow visual guidance UI in RealNodes scenarios provided a significant benefit to completion time compared to our other designs. Emerging 360 VR application types directly benefit from exploring these kinds of UIs. Virtual tours and occupational training software can have a wide variety of locations and paths with less concern of confusing the user. Exercise software can be made using either WIP or another locomotive technique in 360 VR allowing for branching paths that can be telegraphed as the participant moves. New kinds of 360 VR games can be designed, delivering an immersive environment that is easy to navigate compared to traditional methods. Our exploration of navigation in 360 VR experiences, and the resulting contribution of Realnodes and the pilot study, provides feedback for future research and design of navigation in 360 VR.

APPENDIX A: SAMPLE QUESTIONNAIRES

Demographics Survey Participant: _____

Age: _____

Gender: _____

Education Level: _____

How often do you play video games?

Never Rarely Sometimes Frequently Always

How often do you look at 360-degree panoramic or virtual reality images or videos?

Never Rarely Sometimes Frequently Always

How often do you create 360-degree panoramic or virtual reality images or videos?

Never Rarely Sometimes Frequently Always

How often do you play virtual reality video games?

Never Rarely Sometimes Frequently Always

How often do you use virtual reality in other applications?

Never Rarely Sometimes Frequently Always

Do you wear **glasses** or **contacts**? Yes Ns

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
9. Difficulty concentrating	None	Slight	Moderate	Severe
10. « Fullness of the Head »	None	Slight	Moderate	Severe
11. Blurred vision	None	Slight	Moderate	Severe
12. Dizziness with eyes open	None	Slight	Moderate	Severe
13. Dizziness with eyes closed	None	Slight	Moderate	Severe
14. *Vertigo	None	Slight	Moderate	Severe
15. **Stomach awareness	None	Slight	Moderate	Severe
16. Burping	None	Slight	Moderate	Severe

* Vertigo is experienced as loss of orientation with respect to vertical upright.

** Stomach awareness is usually used to indicate a feeling of discomfort which is just short of nausea.

Last version : March 2013

***Original version : 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.

User Engagement Scale Short Form Participant: _____

Instructions for respondents: The following statements ask you to reflect on your experience of engaging with this condition of the study. For each statement, please use the following scale to indicate what is most true for you.

I felt interested in this experience.				
Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5
I was absorbed in this experience.				
Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5
I lost myself in this experience.				
Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5
This video game was attractive.				
Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5
Using the video game was worthwhile.				
Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5
I found this video game confusing to use.				
Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5

The time I spent using this video game just slipped away.

Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5

This video game was aesthetically appealing.

Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5

My experience was rewarding.

Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5

I felt frustrated while using this video game.

Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5

This video game appealed to my senses.

Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5

Using this video game was taxing.

Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
1	2	3	4	5

Visual Guidance Preference Questionnaire Participant: _____

Instructions for respondents:
The following statements ask you to reflect on your experience of engaging with all the conditions of the study. Write a letter next to each number to put each condition in order of preference, where 1 = most preferred visual guidance element, and 4 = least preferred visual guidance element. Place only one condition label next to a number.

---

1 ____

2 ____

3 ____

4 ____

A: First (1st) Condition

B: Second (2nd) Condition

C: Third (3rd) Condition

D: Fourth (4th) Condition

APPENDIX B: IRB APPROVAL LETTER

UNIVERSITY OF CENTRAL FLORIDA

Institutional Review Board
FWA00000351
IRB00001138Office of Research
12201 Research Parkway
Orlando, FL 32826-3246

APPROVAL

November 5, 2019

Dear Samuel Cosgrove:

On 11/5/2019, the IRB reviewed the following submission:

Type of Review:	Initial Study
Title:	Visual Guidance Methods in Immersive and Interactive Virtual Reality Environments with Connected 360° Panoramas
Investigator:	Samuel Cosgrove
IRB ID:	STUDY00001080
Funding:	None
Grant ID:	None
IND, IDE, or HDE:	None
Documents Reviewed:	• Consent_Form, Category: Consent Form; • Demographic form, Category: Survey / Questionnaire; • Protocol, Category: IRB Protocol; • Recruitment.docx, Category: Recruitment Materials; • Representative video - arrow, Category: Other; • Representative video - floor targets, Category: Other; • Representative video - path, Category: Other; • Representative video - ripple, Category: Other; • Simulator Sickness Questionnaire (SSQ), Category: Survey / Questionnaire; • User Engagement Scale (UES) Short Form Questionnaire, Category: Survey / Questionnaire;

The IRB approved the protocol from 11/5/2019.

In conducting this protocol, you are required to follow the requirements listed in the Investigator Manual (HRP-103), which can be found by navigating to the IRB Library within the IRB system.

If you have any questions, please contact the UCF IRB at 407-823-2901 or irb@ucf.edu. Please include your project title and IRB number in all correspondence with this office.

Sincerely,

Adrienne Showman
Designated Reviewer

LIST OF REFERENCES

[1] Insta360 one x - own the moment. https://www.insta360.com/product/insta360-onex.

[2] Vive pro. https://www.vive.com/us/product/vive-pro/.

[3] Vive vr system. https://www.vive.com/us/product/vive-virtual-reality-system.

[4] Review: Myst III: Exile, 2001. IGN.

[5] T. Aitamurto, S. Zhou, S. Sakshuwong, J. Saldivar, Y. Sadeghi, and A. Tran. Sense of presence, attitude change, perspective-taking and usability in first-person split-sphere 360° video. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pp. 545:1–545:12. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574.3174119

[6] M. D. Barrera Machuca, J. Sun, D. Pham, and W. Stuerzlinger. Fluid vr: Extended object associations for automatic mode switching in virtual reality. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 846–847, March 2018. doi: 10.1109/VR.2018.8446437

[7] S. W. Bindman, L. M. Castaneda, M. Scanlon, and A. Cechony. Am i a bunny?: The impact of high and low immersion platforms and viewers' perceptions of role on presence, narrative engagement, and empathy during an animated 360° video. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pp. 457:1–457:11. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574.3174031

[8] S. E. Chen. Quicktime vr: An image-based approach to virtual environment navigation. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive*

Techniques, SIGGRAPH '95, pp. 29–38. ACM, New York, NY, USA, 1995. doi: 10.1145/218380.218395

[9] H. Cho, J. Kim, and W. Woo. Novel view synthesis with multiple 360 images for large-scale 6-dof virtual reality system. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 880–881, March 2019. doi: 10.1109/VR.2019.8798142

[10] D. Q. Felinto, A. R. Zang, and L. Velho. Production framework for full panoramic scenes with photorealistic augmented reality. *CLEI ELECTRONIC JOURNAL*, 16(3):1–10, 2013. doi: 10.1109/CLEI.2012.6427123

[11] S. Freitag, B. Weyers, and T. W. Kuhlen. Interactive exploration assistance for immersive virtual environments based on object visibility and viewpoint quality. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 355–362, March 2018. doi: 10.1109/VR.2018.8447553

[12] A. Fusiello and L. Irsara. An uncalibrated view-synthesis pipeline. In *14th International Conference on Image Analysis and Processing (ICIAP 2007)*, pp. 609–614, Sep. 2007. doi: 10.1109/ICIAP.2007.4362844

[13] H. K. Grigonis. Meet the Samsung 360 Round, a pro-level 360 with 3D and impressive live-streams, 2017. Digital Trends.

[14] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, USA, Second Edition ed., 2003.

[15] J. Huang, Z. Chen, D. Ceylan, and H. Jin. 6-dof vr videos with a single 360-camera. *IEEE Virtual Reality (VR)*, pp. 37–44, 2017. doi: 10.1109/VR.2017.7892229

[16] S. Im, H. Ha, F. Rameau, H.-G. Jeon, G. Choe, and I. S. Kweon. All-around depth from small motion with a spherical panoramic camera. vol. pt.III, pp. 156 – 72. Cham, Switzerland, 2016.

[17] M. P. Jacob Habgood, D. Moore, D. Wilson, and S. Alapont. Rapid, continuous movement between nodes as an accessible virtual reality locomotion technique. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 371–378, 2018.

[18] R. S. Kennedy, N. E. Lane, K. S. Berbaum, and M. G. Lilienthal. Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness. *The International Journal of Aviation Psychology*, 3(3):203–220, 1993.

[19] J. LaViola, E. Kruijff, R. McMahan, D. Bowman, , and I. Poupyrev. *3D User Interfaces: Theory and Practice*. Addison Wesley, 2017.

[20] J. Lee, S. C. Ahn, and J. I. Hwang. A walking-in-place method for virtual reality using position and orientation tracking. *Sensors (Basel)*, 18(9), 2018. doi: 10.3390/s18092832

[21] Y.-C. Lin, Y.-J. Chang, H.-N. Hu, H.-T. Cheng, C.-W. Huang, and M. Sun. Tell me where to look: Investigating ways for assisting focus in 360° video. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pp. 2535–2545. ACM, New York, NY, USA, 2017. doi: 10.1145/3025453.3025757

[22] R. Lopez-Gulliver, T. Hatamoto, K. Matsumura, and H. Noma. Synthesis of omnidirectional movie using a set of key frame panoramic images. *IEEE Virtual Reality Conference*, pp. 221–222, 2015.

[23] M. Lorenz, M. Busch, L. Rentzos, M. Tscheligi, P. Klimant, and P. Fröhlich. I'm there! the influence of virtual reality and mixed reality environments combined with

two different navigation methods on presence. In *2015 IEEE Virtual Reality (VR)*, pp. 223–224, March 2015. doi: 10.1109/VR.2015.7223376

[24] A. MacQuarrie and A. Steed. The effect of transition type in multi-view 360 degrees media. *IEEE Trans Vis Comput Graph*, 24(4):1564–1573, 2018. doi: 10.1109/TVCG.2018.2793561

[25] A. Mas, I. Ismaël, and N. Filliard. Indy: a virtual reality multi-player game for navigation skills training. In *2018 IEEE Fourth VR International Workshop on Collaborative Virtual Environments (3DCVE)*, pp. 1–4, March 2018. doi: 10.1109/3DCVE.2018.8637113

[26] K. R. Moghadam and E. D. Ragan. Towards understanding scene transition techniques in immersive 360 movies and cinematic experiences. *IEEE Virtual Reality (VR)*, pp. 375–376, 2017.

[27] A. S. Muhammad, S. C. Ahn, and J.-I. Hwang. Active panoramic vr video play using low latency step detection on smartphone. *IEEE International Conference on Consumer Electronics (ICCE)*, 2017.

[28] W. Muklashy. Review: Insta360 Pro 2 professional 360 camera, 2019. Digital Photography Review.

[29] H. L. O'Brien, P. Cairns, and M. Hall. A practical approach to measuring user engagement with the refined user engagement scale (ues) and new ues short form. *International Journal of Human-Computer Studies*, 112:28–39, 2018. doi: 10.1016/j.ijhcs.2018.01.004

[30] R. A. Paris, T. P. McNamara, J. J. Rieser, and B. Bodenheimer. A comparison of methods for navigation and wayfinding in large virtual environments using walking.

In *2017 IEEE Virtual Reality (VR)*, pp. 261–262, March 2017. doi: 10.1109/VR.2017.7892276

[31] A. Pavel, B. Hartmann, and M. Agrawala. Shot orientation controls for interactive cinematography with 360 video. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pp. 289–297. ACM, New York, NY, USA, 2017. doi: 10.1145/3126594.3126636

[32] V. C. Pope, R. Dawes, F. Schweiger, and A. Sheikh. The geometry of storytelling: Theatrical use of space for 360-degree videos and virtual reality. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pp. 4468–4478. ACM, New York, NY, USA, 2017. doi: 10.1145/3025453.3025581

[33] W. Powell, V. Powell, P. Brown, M. Cook, and J. Uddin. Getting around in google cardboard – exploring navigation preferences with low-cost mobile vr. *IEEE 2nd Workshop on Everyday Virtual Reality (WEVR)*, pp. 5–8, 2016.

[34] T. Rhee, L. Petikam, B. Allen, and A. Chalmers. Mr360: Mixed reality rendering for 360 degrees panoramic videos. *IEEE Trans Vis Comput Graph*, 23(4):1379–1388, 2017. doi: 10.1109/TVCG.2017.2657178

[35] K. Schlosser. Training workers in VR: Seattle startup Pixvana teams with cruise line operator to help train staff, 2019. GeekWire.

[36] P. Shaffer. Review: The Journeyman Project 3: Legacy of Time, 1998. Adventure Classic Gaming.

[37] S. P. Smith and J. Hart. Evaluating distributed cognitive resources for wayfinding in a desktop virtual environment. In *3D User Interfaces (3DUI'06)*, pp. 3–10, March 2006. doi: 10.1109/VR.2006.60

[38] R. Tanaka, T. Narumi, T. Tanikawa, and M. Hirose. Navigation interface for virtual environments constructed with spherical images. *IEEE Virtual Reality Conference*, pp. 291–292, 2016.

[39] Y. Tomozoe, T. Machida, K. Kiyokawa, and H. Takemura. Unified gesture-based interaction techniques for object manipulation and navigation in a large-scale virtual environment. In *IEEE Virtual Reality 2004*, pp. 259–260, March 2004. doi: 10.1109/VR .2004.1310098

[40] S. Tregillus and E. Folmer. Vr-step: Walking-in-place using inertial sensing for hands free navigation in mobile vr environments. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pp. 1250–1255. ACM, New York, NY, USA, 2016. doi: 10.1145/2858036.2858084

[41] R. Verdugo, M. Nussbaum, P. Corro, P. Nuñnez, and P. Navarrete. Interactive films and coconstruction. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 7(4):1–24, 2011. doi: 10.1145/2043612.2043617

[42] N. Zhao. Full-featured pedometer design realized with 3-axis digital accelerometer. *Analog Dialogue*, 44(6), 2010.