HTC Vive 3D User Interaction Toolkit

● Design Document

Version 4.13.27

4/11/2018

Prepared by

2018-SP2-01

Ian Hanan and Kieran May

Table of Contents

1. Introduction / Purpose of SSD

This System Design Document (SDD) is an in-depth document that was constructed throughout the entire course of this project. The SDD is designed to cover all aspects of our HTC Vive 3D User Interaction Toolkit project. This involves the business requirements, the requirements gathering process, the system requirements of the toolkit, design considerations, and design overview of the interaction techniques. This Systems Design Document is written intended for a software developer with a basic-level understanding of VR to completely understand all elements covered in this project.

The purpose of the SDD is to provide all the information required to a degree that another developer could re-implement this toolkit based on it. The business requirements describe the purpose of the toolkit, the clients requirements/constraints on it, and the schedule and plans to execute this project on a 1 year timeframe. The design considerations, goals, guidelines, and development methods/contingencies describe exactly how this toolkit will be implemented, what it will be implemented with, and why those architectural decisions were made. The design overview discusses individual components of the project, what they are, and where their previous designs originated from with reference. The detailed designs in the SDD describe individual design requirements of each component of the toolkit and how they will be implemented in the Unity Game engine (discussed in requirements). They also provide an insight into the configurable options that a user of the technique will have available to them for each component of the toolkit.

1.1 About Us / Background

Kieran May

Kieran is a 4th year upcoming Bachelor of Software Engineering (Honours) graduate and Research Assistant at the University of South Australia. During this project, Kieran has developed a keen interest in Virtual Reality research and plans to pursue a PhD in 2019 specializing in Virtual/Augmented Reality.

Email Address: Maykw002@mymail.unisa.edu.au

Linkedin: https://www.linkedin.com/in/kieran-may-119340137/

Ian Hanan

Ian is a 4th Year Software Engineering Student Graduating from the University of South Australia late 2018. Currently living in Adelaide, South Australia, and working as a Software Engineer at GTSGroup, Ian's primary areas of interest and specialisation are in Virtual Reality, Real-time data, and AI/Model-Based Reasoning.

Email Address: hanis001@mymail.unisa.edu.au

Linkedin: https://www.linkedin.com/in/ian-hanan-722728143/

Website: ihanan.com

2. Business Requirements

This section describes the principles and strategies to be used as guidelines when designing and implementing the system.

2.1 Executive Summary

Currently a key problem in the Virtual Reality and consistent issue since the 1900s is how to handle human-computer interaction. More specifically; how a virtual user can select and manipulate objects within a virtual environment. Due to the lack of interaction techniques available for VR developers, this has resulted in many limitations, forcing them to either develop from scratch or use an interaction technique which doesn’t necessarily adhere to the requirements of their project. This results in a decreased level of immersion and presence for a users experience.

The aim of this project is to research existing virtual reality interaction techniques, and then implement them as a multiplatform open-source project. The software should be able to support and be ran on both the HTC Vive and Oculus Rift VR headsets through the SteamVR library. Implementing interaction techniques as prefabs allow for developers to simply drag one of our techniques into their scene and full functionality of that technique should just work when they run it. This prefab-format also allows us to provide an open-source alternative for developers to use not only on GitHub, but also directly by downloading it as an asset-package on the Unity asset store. Furthermore, having each technique broken down into a single prefabs provide an easy to use format, which doesn’t take up much space. This way it will not be a detriment to other projects or packages which can be quite convoluted. Each of our techniques also have no dependencies, meaning they are all separate and don’t rely on each other whilst being 100% free for any developer to use and and make changes as they wish. Developers can then decide if they want to utilize our techniques alongside other solutions as they will allow for maximum compatibility with existing VR tools. The implementation of our selection and manipulation techniques will be programmed in C# using the Unity 3D game engine and SteamVR library.

2.2 Project Purpose/Justification

The purpose of this project is to implement a wide-range of selection and manipulation techniques with an extremely user-friendly approach of allowing developers to simply drag techniques as prefabs into their Unity projects. Each interaction technique will have some fundamental customization inside the Unity inspector so developers can tailor the technique to meet their projects needs. Despite extensive research been done within Virtual Reality in regards to interaction techniques, there is still a significant lack of resources for developers who to wish to implement these techniques into their VR projects.

2.2.1 Business Need/Case

Currently in the realm of VR technology (especially in the open-source area) the available interaction techniques are quite limited. Almost all interaction techniques in other open-source solutions and even the SteamVR and Oculus SDK's consist of variations of a laser pointer, and select by touching technique to pick up an object. While these methods are fine for some scenarios they can be quite limiting to a user in a lot of others.
The interaction tools we will be creating will provide alternative techniques which will be better suited for different situations. For example it can be quite difficult to pick up small object such as an individual coin surrounded by other small objects with just a laser pointer. This is even harder utilizing the touching technique. Another scenario is if you wanted to select and pick up multiple objects at the same time. The techniques that will be implemented address issues such as these providing different methods for a developer to interact with the virtual environment.
These tools will provide an easy drag and drop use case where a developer will be able to drag the tool onto their worlds hierarchy and it will search through it and connect with the current VR system. With thorough documentation and the ease-of-use provided a developer will be able to save time, allow more diverse interaction with their environment, and customize them to their own needs for their project.

2.2.2 Business Objectives

The business objectives for the project are as followed:

- Implement various existing Virtual Reality interaction techniques as an open-source multiplatform project using the Unity 3D game engine. **(See 1.5 project deliverables for a full list of interaction techniques)**
- Implement the interaction techniques in a prefab (drag and drop) format to provide a user-friendly way for developers to utilize our interaction techniques in their project.
- Ensure that techniques have been programmed correctly with minimal bugs and glitches to avoid our interaction techniques potentially being dragged and dropped into an existing project and causing issues.
- Support multiple Virtual Reality platform which includes at least the HTC Vive and the Oculus Rift kits.

2.3 Project Description

The aim of this project is to research and implement a variety of existing Virtual Reality interaction techniques as prefabs within the Unity 3D game engine. Other developers should have the ability to import the techniques into their existing projects as a Unity asset package, and import a fill list of selection and manipulation techniques. Developers will then be able to simply drag the prefab into their existing projects scene, this process should be achieved with minimal to no bugs or glitches and be multi-platform compatible with both the HTC Vive and Oculus Rift. This will be accomplished by initially researching a variety of selection and manipulation techniques from the '3D User Interfaces theory and practice' textbook, and then reading the original research papers to have a complete understanding with how each technique works, and then going into the Unity 3D game engine and implementing them.

2.3.1 Project Objectives and Success Criteria

Separate techniques can be checked off as individual objectives as they are completed. These techniques can be found under 1.5 “Project deliverables“.

A detailed test environment will be available for every technique so that a user can run the environment on their own computer and try out and learn about the technique. It will contain a variety of interactable objects as well as a manipulation, selection, and docker game allowing the user to see the advantages of the technique they are utilizing. Following the agile development process the individual parts of the project must be tested repeatedly and strictly. This objective will be repeated at the end of each other objective.

2.3.2 Requirements

Note: A full breakdown of system non-functional and functional requirements are found in section 3.3.

- The techniques must be prefabs.
- The techniques must be individual and not rely on each other to work. (unless specified by the technique description)
- The techniques must be cross-platform to work with at oculus and HTC vive hardware.
- The techniques need to be able to be dragged onto the hierarchy and automatically attach themselves to existing vr controllers if this option is selected by the developer.
- Concise descriptions of the techniques and how to use them must be available.
- Parameters of the techniques must be customizable by the user.
- A user must be able to try out the techniques in a test environment.
- The project must be open-source and free to use for developers.
- The source control must be on github.

2.3.3 Constraints

- Time: There are 23 techniques to research and implement. This will take a lot of time and effort to research, build, debug, and user-test.
- The Unity 3D game engine. We cannot implement our interaction techniques in a different engine as using this specific game engine is a requirement of the client.
- Hardware Equipment
    - The project is designed to run the selection and manipulation techniques as a multi-platform tool however, due to time and hardware requirement limitation we will only support the two main VR platforms HTC Vive and Oculus Rift.
    - Computers with the minimum specs required to run the HTC Vive and Oculus Rift. (Nvidia GTX 960, Intel i3-6100, RAM 8GB or equivalent hardware)
    - Interaction techniques that use specific hardware requirements (For example the Voodoo-doll technique which utilizes a tracker-glove) will not be implemented due to not having access to the hardware.

- Budget: Due to having no project-budget (outside of using our own money) all assets and models used in our project will be free and taken from the Unity asset store.

2.3.4 Assumptions

- The amount of time that it will take us to implement each individual techniques; based on our research and programming abilities shouldn't individually take too long and we will be able to implement them within our clients required time-frame.
- The level of understanding required in Unity 3D. Although we both have prior understanding with Unity at a basic level, we don't have an expert-leveling understanding of Unity.
- Having direct access to all relevant Virtual Reality hardware required in the project, this includes the HTC Vive & Oculus Rift kits (headset, controllers, motion-sensors & relevant software to run the VR kits) and computer with the minimum specs of Nvidia GTX 960, Intel i3-6100, RAM 8GB or equivalent hardware to run the VR kits.

2.3.5 Preliminary Scope Statement

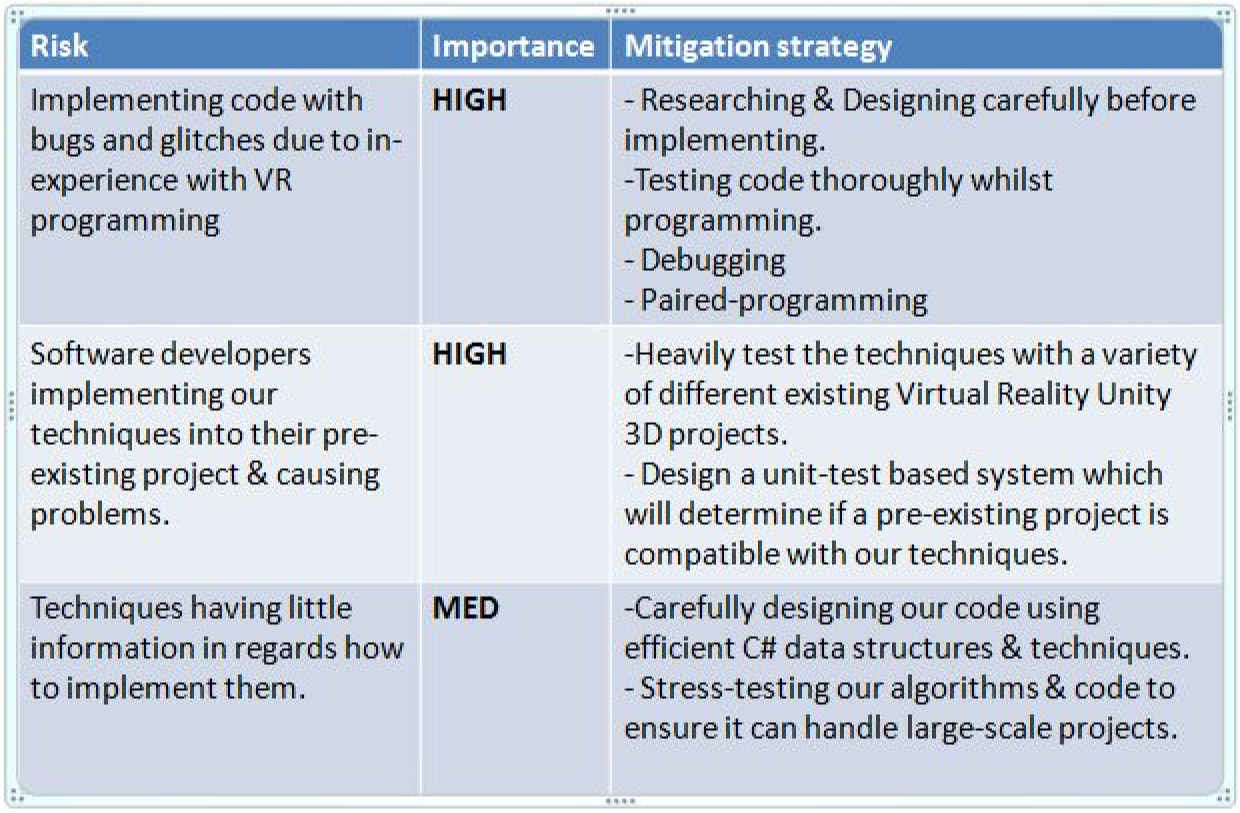
To implement (and re-implement) the specified Virtual Reality user interaction techniques by the client as an open-source alternative to make interaction techniques more accessible to the VR community. These techniques will be created as individual prefabs within the Unity 3D game engine to provide a user-friendly approach of being drag and dropped onto an existing projects. They must provide cross-platform support for both the HTC Vive and Oculus Rift hardware that utilizes stereoscopic lens, motion tracking controllers, and tracking sensors. These techniques must be highly customizables for developers to specifically tailor the technique to meet their project requirements whilst still being easy to use.

The end product will contain a project of all the interaction techniques (listed in 1.5 'project deliverables') with the ability to be imported individually to Unity 3D. Each technique will individually have a specific global testing environment to simulate a variety of realistic VR projects. These testing environments will have mini-games for the user to play with as well as a manipulation, selection, and docker game to allow the user to test the techniques. More information on this can be found in the design section. This product will be uploaded publicly as an open-source project on GitHub (with concise descriptions and tutorials on how each individual interaction technique works) and available for free on Unity asset store for developers to use.

2.4 Risks (Low, medium, or high)

- Many of the interaction and manipulation techniques have little to no information in regards to how to actually implement them, such as what algorithms to use. Implementing them incorrectly could result in major flaws in the code which may include bugs, performance issues, and glitches for other developers who wish to use our techniques in their projects. This presents as a medium level risk.
    - To mitigate this risk we had access to some prominent VR researchers at the university if we needed advice on how to proceed. Such as the client Bruce Thomas, Andrew Cunningham, Mark Billinghurst. Also we made sure to gather individual requirements of the techniques and their designs early so that if we ran into any issues we would have the option to decide to cancel the technique.

- Due to a lack of experience in Unity 3D & VR programming, the potential for errors, bugs, and glitches to occur is a major risk, especially with the techniques we implement early on in the project as we're still in the process of learning whilst we're implementing techniques. This may be an issue for developers who wish to implement our user interaction and manipulation techniques into their existing projects as the smallest bugs or glitches present in our code could potentially be detrimental to their existing projects. To the reduce risk of this occurring we will heavily user-test the interaction and manipulation techniques with a variety of different existing Virtual Reality Unity 3D projects. We also have a system set in place which can determine whether the developers pre-existing project is compatible with our interaction & manipulation techniques.



Risk	Importance	Mitigation strategy
Implementing code with bugs and glitches due to in-experience with VR programming	**HIGH**	- Researching & Designing carefully before implementing. -Testing code thoroughly whilst programming. - Debugging - Paired-programming
Software developers implementing our techniques into their pre-existing project & causing problems.	**HIGH**	-Heavily test the techniques with a variety of different existing Virtual Reality Unity 3D projects. - Design a unit-test based system which will determine if a pre-existing project is compatible with our techniques.
Techniques having little information in regards how to implement them.	**MED**	-Carefully designing our code using efficient C# data structures & techniques. - Stress-testing our algorithms & code to ensure it can handle large-scale projects.

2.5 Project Deliverables

NOTE: The definition for these techniques are taken as direct quotes from the 3D User Interfaces theory and practice textbook

Below are a list of interaction & manipulation techniques which have been taken from the textbook ‘3D User Interfaces theory and practice’ and are required to be implemented as prefabs within the Unity 3D game engine:

[1]Grasping metaphors

Hand-based grasping techniques

- Go-Go technique
    - ‘The ‘Go-Go’ technique attempts to improve on the simple virtual hand by providing an unobtrusive technique that allows the user to interactively change the length of the virtual arm.’

Enhanced grasping metaphors

- 3D bubble cursor
    - ‘A bubble cursor is an area cursor that dynamically changes it radius to always touch the closest target.’
- PRISM
    - ‘The basic concept of PRISM is to apply a scaled-down motion to the user’s virtual hand when the user’s physical hand is moving below a specified speed (SC).’
- Hook
    - ‘The hook enhancement is based on the observation that if a target is moving in an unpredictable but non-Brownian motion, the user must follow the target in order to attempt to overtake it.’

Pointing metaphors

- Fishing reel
    - Fishing reel technique is an addition to ray-casting, ‘this technique allows the user to select an object with the simple ray-casting technique, then reel it back and forth using the dedicated input mechanism, which could be, for example, a simple mechanical slider, a joystick etc..’
- Absolute and Relative Mapping
    - Absolute and relative mapping is in the form of a Raycast (However It could be adapted to a simple hand technique). When you press the set button (touchpad) the movement of the virtual controller relative to your real controller is scaled with a ration of 10:1. By doing this you can be precise when selecting small or distant objects with the ray. This is because the distance you have to move your hand

[1] LaViola, J., Kruijff, E., McMahan, R., Bowman, D. and Poupyrev, I. (2017). 3D user interfaces. 2nd ed. Boston: Addison-Wesley, pp.256-315.

across an object is amplified 10x due to the ratio.

Volume-Based pointing techniques

- Flashlight
    - ‘In the flashlight technique, the pointing direction is defined in the same way as in the simple ray-casting technique, but it replaces the virtual ray with a conic selection volume, with the apex of the cone at the input device.
- Aperture selection
    - ‘The aperture technique is a modification of the flashlight technique that allows the user to interactively control the spread of the selection volume.
- Sphere-casting
    - ‘Sphere-casting is essentially a modified version of ray-casting that casts a sphere onto the nearest intersected surface. Objects found within the sphere are considered selectable.’

Enhanced pointing metaphors

- Bendcast
    - ‘This enhancement bends the vector used for pointing toward the object closest to the vector’s path. To determine the closest object for Bendcast, the point-line distance from each selectable object to the pointing vector must be calculated.’
- Depth Ray
    - ‘The depth ray is a simple enhancement designed to disambiguate which objects the user intends to select when the pointing vector intersects multiple candidate targets.’
- Image-Plane Pointing
    - **Framing hands:** Uses two hands to select an object. The user places each hand around an 3d object in scene (as if it was a 2d object in front of them) and this allows them to select it from afar.
    - **Sticky Hands:** Allows the user to gesture at an 3D object and grab it out of the air as if it is a 2d object right in front of them.
- Serial selection mode
    - Allows the user to select multiple objects one after the other adding them to an selection array.

Proxy techniques

- World in miniature
    - ‘The world-in-miniature (WIM) technique provides the user with a miniature handheld model of the virtual environment, which is an exact copy of the VE at a smaller scale.’

Bimanual Techniques

Symmetric Bimanual techniques

- Spindle

    - ‘Spindle is a symmetric-synchronous bimanual technique originally developed by Mapes and Moshell. With the technique, two 6-DOF handheld controllers are used to define a virtual spindle that extends from one controller position to the other. The center of this spindle represents the primary point of interaction, which can be used to select and manipulate objects.
- iSith
    - ‘With iSith, the positions and orientations of two 6-DOF handheld controllers are used to define two separate rays, similar to ray-casting with both hands. The shortest line between these two rays is then calculated by crossing the two vectors to find a vector perpendicular to both.

Asymmetric Bimanual techniques

- Spindle + wheel
    - This technique is an extension from ‘the original Spindle technique to include an additional feature for rotation the pitch of the selected object. The feature was a virtual wheel collocated with the dominant hand’s cursor. The user can twist the dominant-hand controller to rotate this virtual wheel about its axis.’
- Flexible pointer
    - ‘The flexible pointer is a curved ray that can be used to point at partially obscured objects. The pointer is implemented as a quadratic Bezier spline. Three points - the nearest hand, the farthest hand, and a control point - control the position, length and curvature of the Bezier spline.’

Hybrid metaphors

- Homer
    - ‘HOMER stands for hand-centered object manipulation extending ray-casting. The user selects an object using a ray-casting technique, and instead of the object being attached to the ray, the user’s virtual hand instantly moves to the object and attaches to it.’
- Scaled-world Grab
    - ‘The scaled-world grab technique is based on principles similar to HOMER. The user starts by selecting an object using some selection technique. After successful selection, the interface switches into manipulation mode, and the user can position and rotate the virtual object in space. However, instead of scaling the user’s hand motion, as in HOMER, the scaled-world grab technique scales down the entire VE around the user’s virtual viewpoint.’

Progressive refinement

- SQUAD
    - SQUAD is a ‘progressive refinement approach for making object selections in dense virtual environments. The technique consists of two phases. During the first phase, the user specifies the initial subset of environment objects to consider selectable by sphere-casting. After the initial step of sphere-casting, the set of selectable objects is distributed and displayed in a quad menu that replaces the user’s view of the virtual environment.’

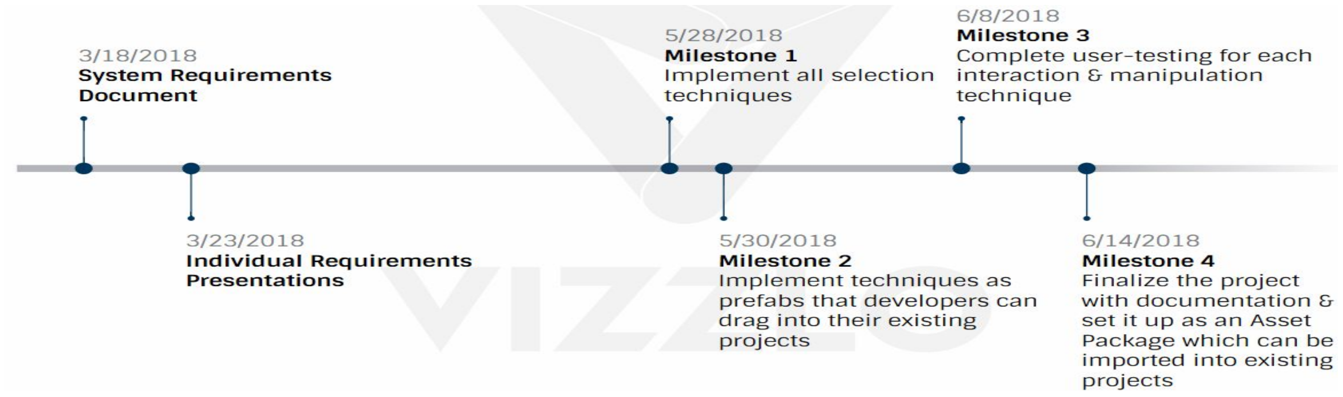
- EXPAND
  - EXPAND is also consists of two phases for selection. ‘During the first phase, the user uses a large cursor to define an area of selectable objects, similar to SQUAD’s sphere-casting step. However, during the second step, Expand zooms the user’s view with the quad menu. Along with this zoomed-in view, Expand creates clones of the selectetable objects and distributes the clones in a virtual grid in front of the view. To complete the second phase and finalize a selection, the user points to the clone in the grid that matches the original target, which can still be in the zoomed--in view.
- Double Bubble
  - ‘To reduce the number of items in the initial selection, a 3D Bubble Cursor is used. However, to make this technique usable in a dense environments, the bubble is not allowed to shrink beyond a certain size. If only a single object is in the bubble when the user confirms the selection, it is selected directly. Otherwise, the objects in the bubble are placed into a menu in the image plane, similar to the Expand technique.’

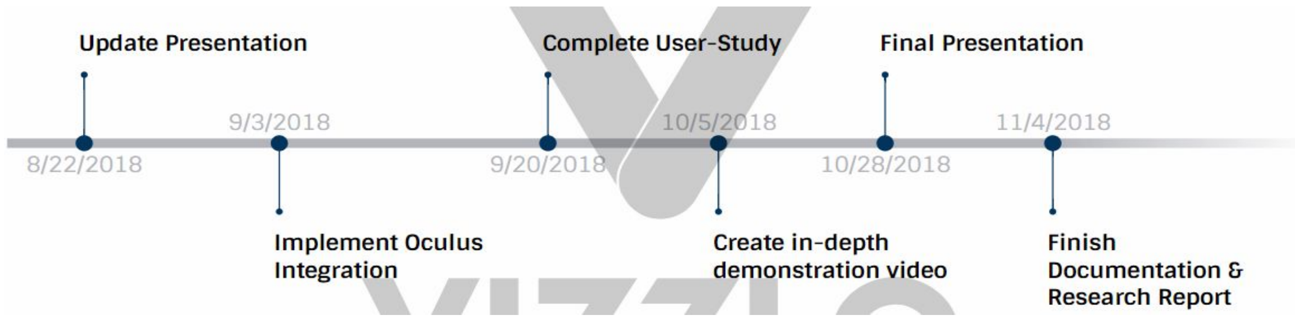
Test Environment(s) for the techniques

Functioning github project with explicit details of the techniques

2.6 Summary Milestone Schedule

Below is the timelines for each semester we created and followed to meet our clients criteria and requirements by the end of the project.


3/18/2018
System Requirements Document
3/23/2018
Individual Requirements Presentations
5/28/2018
Milestone 1
Implement all selection techniques
5/30/2018
Milestone 2
Implement techniques as prefabs that developers can drag into their existing projects
6/8/2018
Milestone 3
Complete user-testing for each interaction & manipulation technique
6/14/2018
Milestone 4
Finalize the project with documentation & set it up as an Asset Package which can be imported into existing projects



2.7 Summary Budget

The budget consists of various hardware & software associated with Virtual Reality technology which includes:

- HTC Vive Kit ($999) & Oculus Rift Kit ($600)
  - HTC Vive headset / Oculus Rift headset
  - Camera sensors
  - Controllers
  - Headphones
- Unity 3D (Free edition) - $0
  - We have confirmed that all licencing is correct for as well as asset licencing to build with this edition.
- Stream VR and Oculus software - $0
- Virtual Reality Toolkit library - $0
- Computers with minimum specs to run the software & hardware. (Price varies depending on specs - Approx $1000-$3000)

2.8 Project Approval Requirements

For the project to have reached an successful completion by the final deadline the techniques must be implemented as prefabs and be able run on a newly created Unity VR project. They must be at an acceptable level to be released on github and the Unity asset store for anyone to use. This means bugs must be at a minimal, documentation must be provided on each technique so people know how to use them, and test environments for every technique must be available.
Based on the prefabs we create an advanced developer should be able to figure out how to manually implement the prefabs into an existing project to utilize the techniques provided. Ideally though any developer will be able to drag the prefab anywhere into the projects hierarchy and then the prefab will find the VR controllers and attach itself onto them automatically. Then the controllers will immediately work when the project is run.

2.9 Project Management

Due to the nature of our project and there only being two of us the realm of project management falls onto both of us as a team. We are both highly contactable with each other and can get responses to questions from each other within minutes. All decisions, contact with clients, and planning can be executed by both of us concurrently.

2.9.1 Project Manager Responsibilities

Due to the nature of our project and there only being two of us the realm of project management falls onto both of us as a team. We are both highly contactable with each other and can get responses to questions from each other within minutes. Because of this all decisions, contact with clients, and planning can be executed by both of us concurrently.

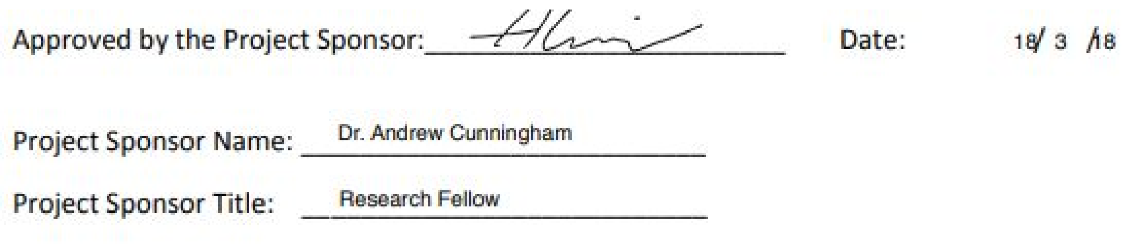
- Responsibilities as project manager(s) include:
- Client contact and cc'ing everyone into the discussion
- Task management knowing whos doing what technique and keeping track of the
- methods that we are executing these techniques
- Keeping on the same page and attacking problems with similar methods
- Making sure all code is in the same style and format
- Documentation of everything that is done along the way
- Scheduling meetings via outlook calendar etc

2.9.2 Project Management Plan

Our major method of undertaking these project management responsibilities to make sure we both remain on track and on task is via meetings. First we have a meeting with the client(s) weekly where we keep them up to date with everything to do with the project and its current status. Secondly we have weekly meetups where both of us go to the lab together to reaffirm what we are doing and going over the responsibilities with a iterative weekly approach. This includes:

- Pair programming to go through and clean up the code
- Reading through each others code
- Checking for inconsistencies in style
- Working on documentation and our approach to it
- Checking we document with the same style
- Discussing key concepts we to talk about in our client meeting so our time with them will be used optimally and getting all the information we require
- Updating plans to the current schedule and our progress
- Checking scheduled dates for meetings and deadlines
- Reaffirming what each of us will be doing when we are not together
- Discussing concepts and ideas to help each other accomplish our tasks

2.9.3 AUTHORISATION

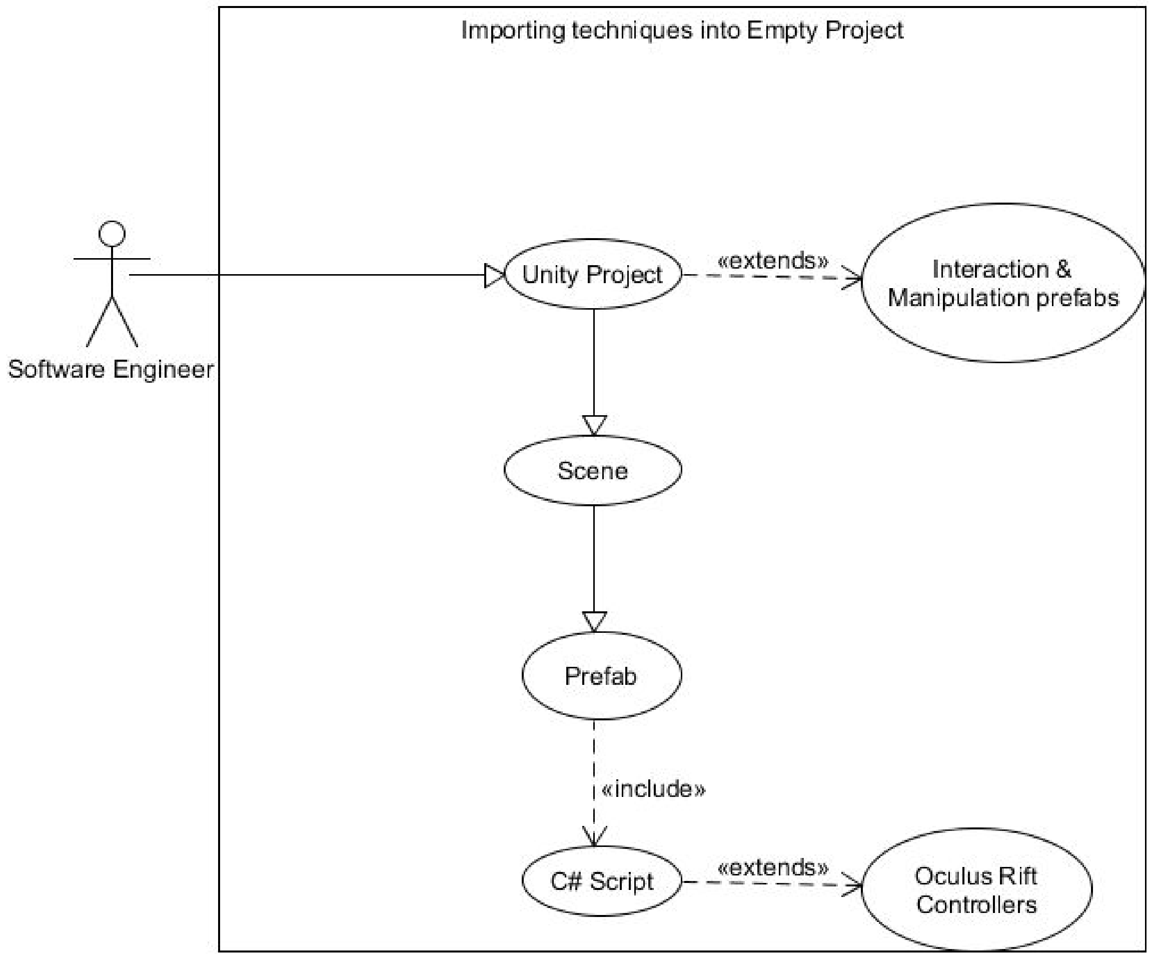


Approved by the Project Sponsor:	[illegible]	Date:	18/ 3 /18
Project Sponsor Name:	Dr. Andrew Cunningham		
Project Sponsor Title:	Research Fellow		

3. System Requirements

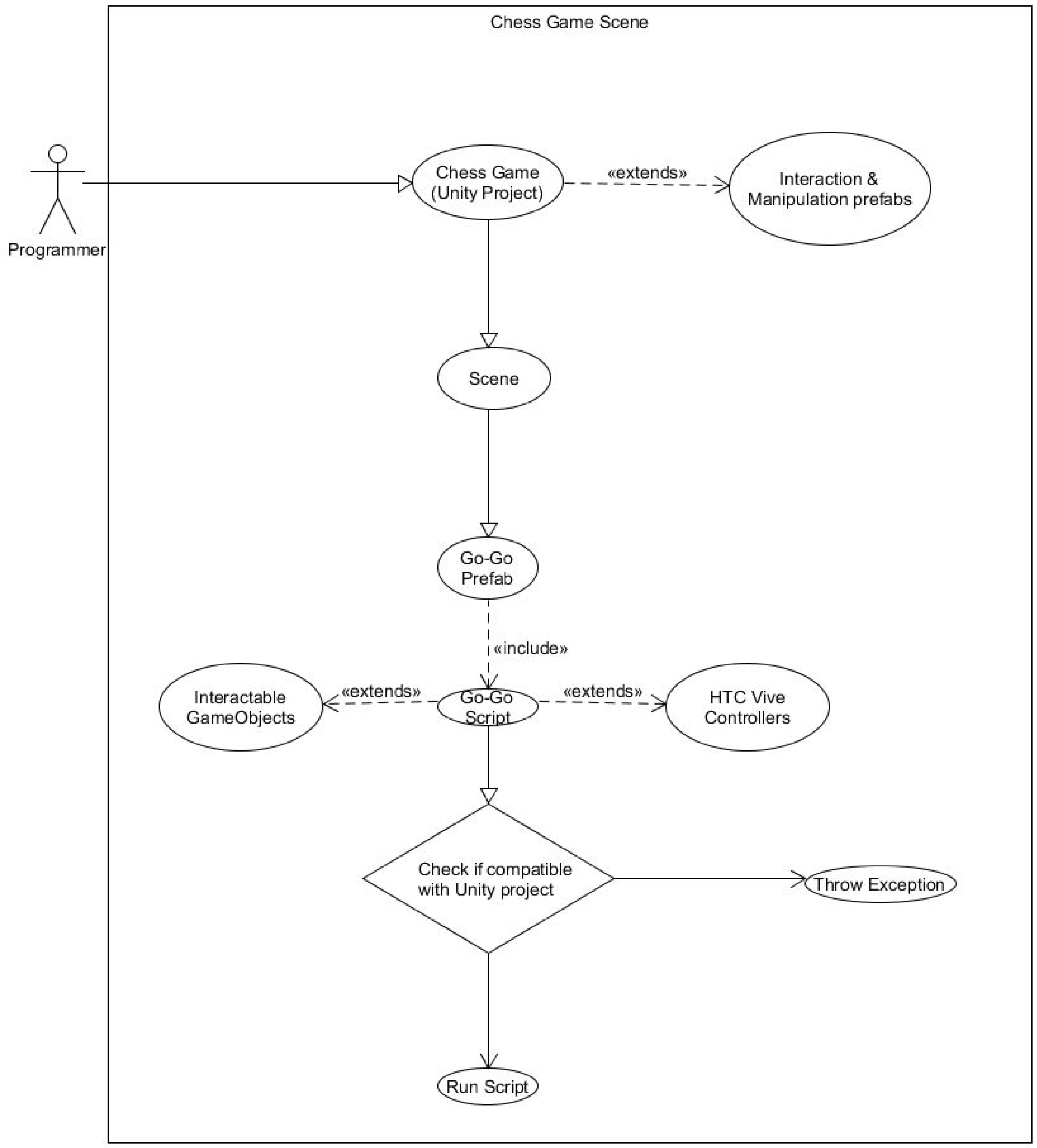
3.1 Use Case Analysis

3.1.1 Software Engineer importing Interaction & Manipulation techniques into a project



A Software Engineer wishes to create a new Unity 3D Virtual Reality project in order to test out different object interaction and manipulation techniques. The Software Engineer opens his/her Unity version and creates a new 3D Virtual Reality project, the Software Engineer is able to import the Interaction & Manipulation prefabs as an asset package. The Software Engineer then creates an empty new scene, and simply drags the interaction or manipulation technique in his/her project as a prefab. The prefab extends a C# script which has the ability to automatically detect the virtual hands/controllers and handles all the code behind the scenes which makes the interaction or manipulation technique to run.

3.1.2 Programmer importing Interaction & Manipulation techniques into a pre-existing project



A programmer has been working on creating a Virtual Reality multiplayer chess game where two separate players are matched up and have the ability to interact with the different chess pieces on the board using the HTC Vive. The programmer wishes to implement the Go-Go hand-grasping technique to interact with the chess pieces on the board. The programmer opens his/her chess game project through Unity 3D and imports the Interaction & Manipulation prefabs as an asset package. Within the selected scene, the programmer drags the Go-Go prefab into the scene which extends the Go-Go C# Script. The C# script automatically detects the HTC

Vive Controllers and the Interactable game objects present in the scene. When running the scene, a compatibility test occurs to check if the Go-Go script is able to be successfully implemented into the pre-existing project without any errors or glitches occuring.

3.2 Functional Requirements

The system shall:

ID	Title	Description
FR1.1	Allow techniques to be used as prefabs	The Software Engineer has the ability to simply import the different VR manipulation and interaction techniques into their new project as prefabs.
FR1.2	Automatically detecting VR controllers	The prefab will automatically detect the VR controllers through the script. This process will take place at run-time.
FR1.3	Drag prefabs into scenes	The manipulation or interaction technique will be implemented by simply dragging it as a prefab into the existing scene.
FR1.4	Detect interactable game objects	The prefab will have the ability to automatically detect interactable game objects present in the scene.
FR1.5	Determine based on the users choice which objects can be interacted with via layers	The user will be able to select from drop-downs on our techniques what layers the techniques will be able to interact with. This way you wont “pick up” the floor by accident.
FR1.6	Dependencies	All techniques must not be reliant or have dependencies on each other. If a user desired they should be able to access the github, open the asset folder, and individually download one single technique from the repository and it should work out of the box.
FR1.7	Github wiki	Documentation for the techniques will all follow the same below structure so there will be no ambiguity: Overview

		Small Gif of the technique in action in Unity Source paper reference Overview of how we implemented the technique described into unity Inspector parameters list Class methods list Test scenes/examples scenes list

3.3 Non-Functional Requirements

(Note: Use Cases 1 & Use Cases 2 non-functional requirements have been joined together as both use cases contain the same non-functional requirements)

ID	Title	Description
NFR1.1	Usability	● The prefabs are accessible through Unity 3D using a VR Kit.
NFR1.2	Performance	● Dragging prefabs into a scene will not significantly decrease performance of the scene. ○ No more than 10 FPS lost when using minimum hardware required for HTC Vive. (Nvidia GTX 970, Intel i5-4590, RAM 4GB or equivalent hardware) ○ No more than 15 FPS lost when using minim hardware for Oculus rift. (Nvidia GTX 960, intel i3-6100, RAM 8GB or equivalent hardware)
NFR1.3	Reliability	● Prefabs implemented will have minimal bugs and glitches. ● Programmers have the ability to drag Prefabs into their pre-existing projects without causing issues.

NFR1.4	Multi-environment support	● Programmers will be able to implement prefabs into their existing projects on multiple-platforms including the HTC Vive * Oculus Rift.
NFR1.5	Scalability	● Prefabs will be able to run effectively on both small-scale and large-scale Unity projects.
NFR1.6	Availability	● Prefabs should be re-usable and obtainable within the developers project files.

4. Design Considerations

The majority of issues that need to be addressed and resolved before attempting to devise a complete design solution stem from key requirements of the system and how we will implement them. Once we decide on how to implement a specific design pattern, this will be utilized for all 23 techniques. Because of the large amount of objects using them, finding out later on that our implementation is not optimal, or doesnt work as expected and needing to fix it would waste a lot of project time and resources.

Considerations include:

- How we are going to design a mechanism to allow unity to detect the HTC Vive/Oculus headset and controller objects automatically and attach our scripts to them. Do we search the entire game scene for them? This could hurt performance. Or do we create a set of constants of the names of the controllers so that they can find them?

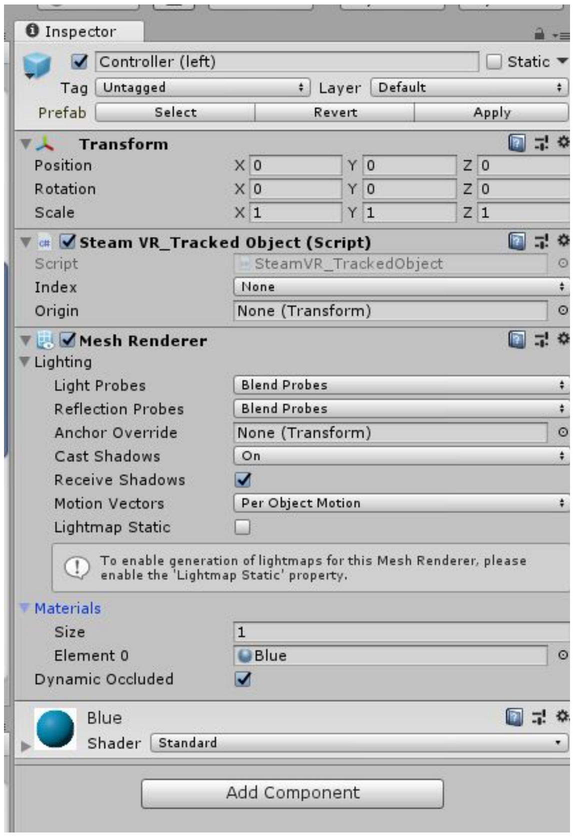
4.1 Goals and Guidelines

- The 23 techniques must not be reliant on eachother (or each others scripts to work). A user should be able to access the toolkits github, open the asset folder, and then individually download a single technique from the repository and it should work out of the box.
- While the techniques are independent, they must still follow some broad design patterns as to not confuse a new user trying out the toolkit. Patterns include:
    - Each technique has a user-configurable interactable objects layer
    - Interactable object’s layer in Unity must be the same for each technique
    - Each technique must have a drag-drop and play prefab where dragging the prefab onto the scene will automatically set up the technique
    - If the automatic prefab fails, there should be public variables that the user can manually assign to set up the technique
    - Each technique must have a option to be used as purley selection or manipulation
    - Each technique will utilize the unity events system e.g. Event invoked on:
        - Object hover
        - Object un-hover
        - Object selected
- Github wiki documentation for the techniques will all follow the same below structure so there will be no ambiguity
    - Overview
    - Small Gif of the technique in action in Unity
    - Source paper reference
    - Overview of how we implemented the technique described into unity
    - Inspector parameters list
    - Class methods list
    - Test scenes/examples scenes list
- A primary examples scene will have a variety of testing games (Manipulation, selection, docking) and will exist for every technique with the only difference being the technique itself being displayed.

4.2 Development Methods & Contingencies

Our choices for the method or approach for the software design of our toolkit are limited by the workflow presented by the Unity3D game engine. Management patterns such as MVC, MVVM, work against the workflow and architecture of the Unity system. Also, due to the fact that this project will be released as an open-source toolkit for any developer to use it would be counter-productive to go against the grain and not use the most common design approach amongst unity developers. Due to this we will follow this popular Unity Component gameobject architectural design. The approach of the design is Object-oriented but each in-game object/entity needs to be treated as its own separate self-contained system. We can follow general object-oriented methodology except with minimal inheritance. To do this unity utilizes a component architecture, which is set up in the user interface. For example, rather than programming a poodle and beagle class, which inherit from a dog class. You would just create a dog class and then specify differences between a poodle and beagle via different components attached to the same gameobject as the dog script. Such as their animation, and mesh.

e.g. Component view in unity:



Controller (left) Static
Tag Untagged Layer Default
Prefab

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Steam VR_Tracked Object (Script)
Script SteamVR_TrackedObject
Index None
Origin None (Transform)

Mesh Renderer
Lighting
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Cast Shadows On
Receive Shadows ✓
Motion Vectors Per Object Motion
Lightmap Static ☐

To enable generation of lightmaps for this Mesh Renderer, please enable the 'Lightmap Static' property.

Materials
Size 1
Element 0 Blue
Dynamic Occluded ✓

Blue
Shader Standard

In the image to the left, we have a 'Controller object'. The Controller has a script attached to it called "SteamVR_trackedObject" which contains C# code specifying some greater detail about the objects interaction. However the gameobject gets lots of its features from other 'components' attached to it. The transform component describes its location in 3D space, rotation, and scale. The mesh renderer describes the object as blue. etc.

5. Operational Scenarios

Below is some descriptions of the core operational scenarios of how a user will make use of our techniques for their projects. Specific operational scenario and game play of individual techniques and how they work are explained under their headings in 7. detail design.

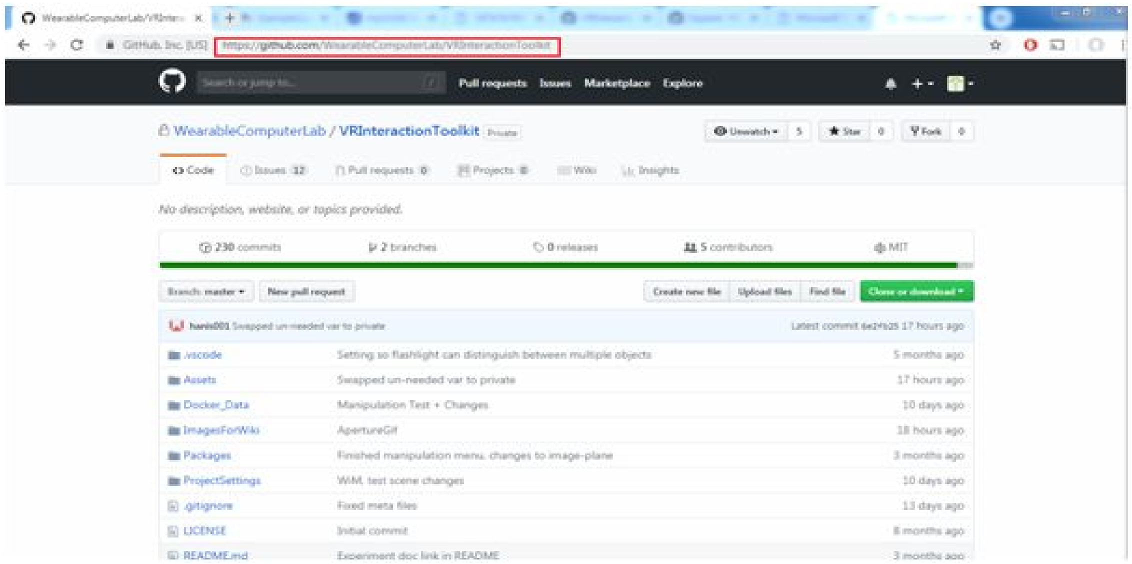
5.1 Importing our techniques

There are two ways which a user can prepare our techniques in a Unity project.

- 1: Downloading the source/asset package directly through the GitHub
- 2: Installing our techniques through the Unity Asset Store

5.1.1 Downloading directly through GitHub

Step 1: Open up a web-browser of your choice and enter the url: https://github.com/WearableComputerLab/VRInteractionToolkit



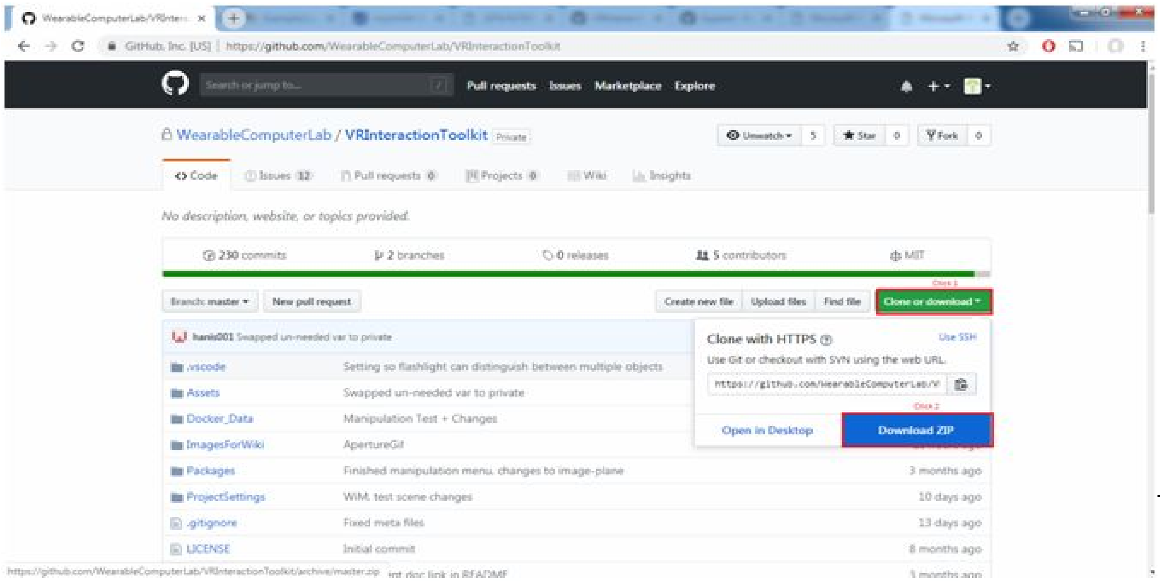
No description, website, or topics provided.

230 commits | 2 branches | 0 releases | 5 contributors | MIT

hanis001 Swapped un-needed var to private — Latest commit 6e2fb25 17 hours ago

.vscode	Setting so flashlight can distinguish between multiple objects	5 months ago
Assets	Swapped un-needed var to private	17 hours ago
Docker_Data	Manipulation Test + Changes	10 days ago
ImagesForWiki	ApertureGif	18 hours ago
Packages	Finished manipulation menu, changes to image-plane	3 months ago
ProjectSettings	WiM, test scene changes	10 days ago
.gitignore	Fixed meta files	13 days ago
LICENSE	Initial commit	8 months ago
README.md	Experiment doc link in README	3 months ago

Step 2: Once the GitHub page loads left click the 'Clone or download' button, and you can specify whether you either want to open it inside your desktop, or download it as a ZIP. In this scenario, I've decided to download it as a .ZIP file



No description, website, or topics provided.

230 commits | 2 branches | 0 releases | 5 contributors | MIT

hanis001 Swapped un-needed var to private

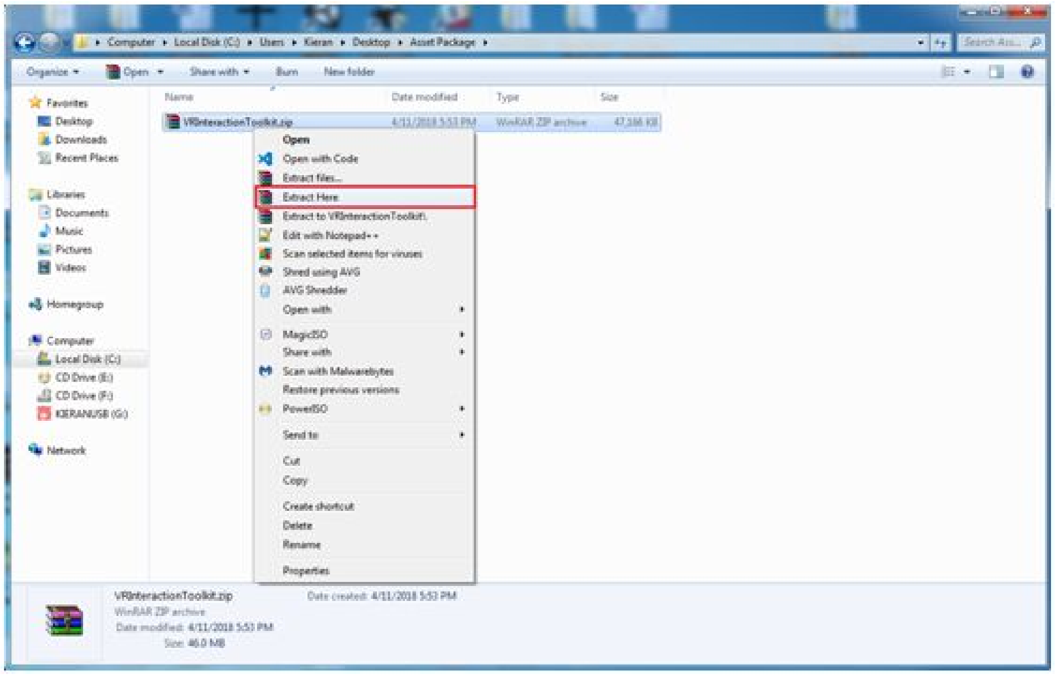
Use Git or checkout with SVN using the web URL.

https://github.com/WearableComputerLab/V

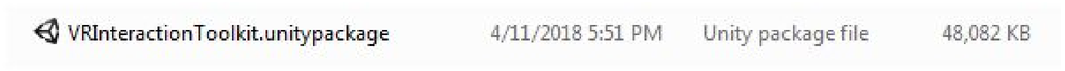
.vscode	Setting so flashlight can distinguish between multiple objects	
Assets	Swapped un-needed var to private	
Docker_Data	Manipulation Test + Changes	
ImagesForWiki	ApertureGif	
Packages	Finished manipulation menu, changes to image-plane	3 months ago
ProjectSettings	WiM, test scene changes	10 days ago
.gitignore	Fixed meta files	13 days ago
LICENSE	Initial commit	8 months ago

https://github.com/WearableComputerLab/VRInteractionToolkit/archive/master.zip

Step 3: After downloaded the .zip file, make sure to extract the .zip file using a compression extraction software of your choice.

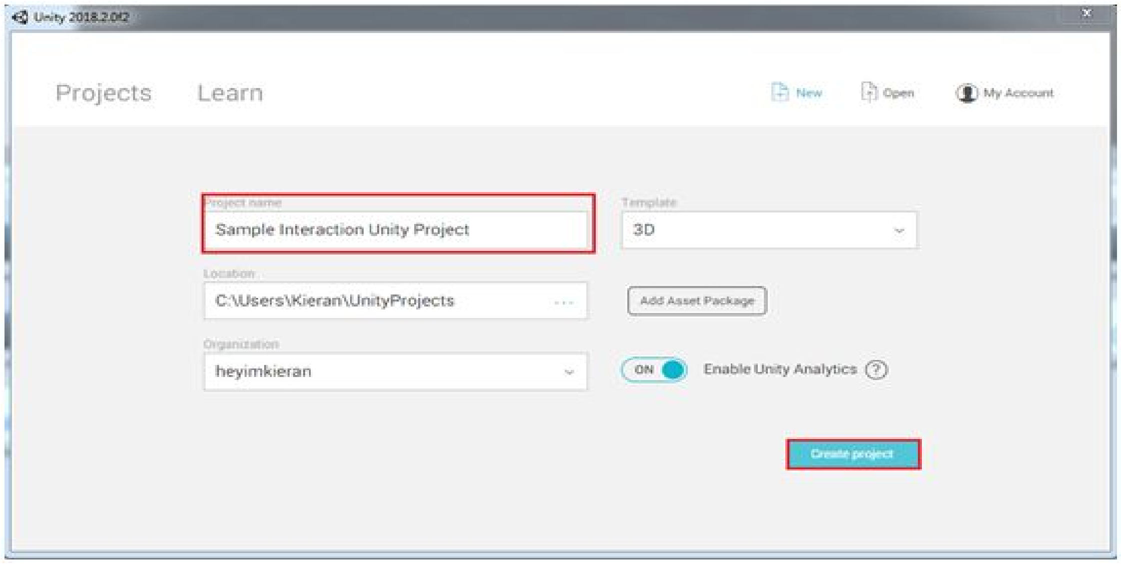


A .unitypackage will then appear in the directory you extracted to.



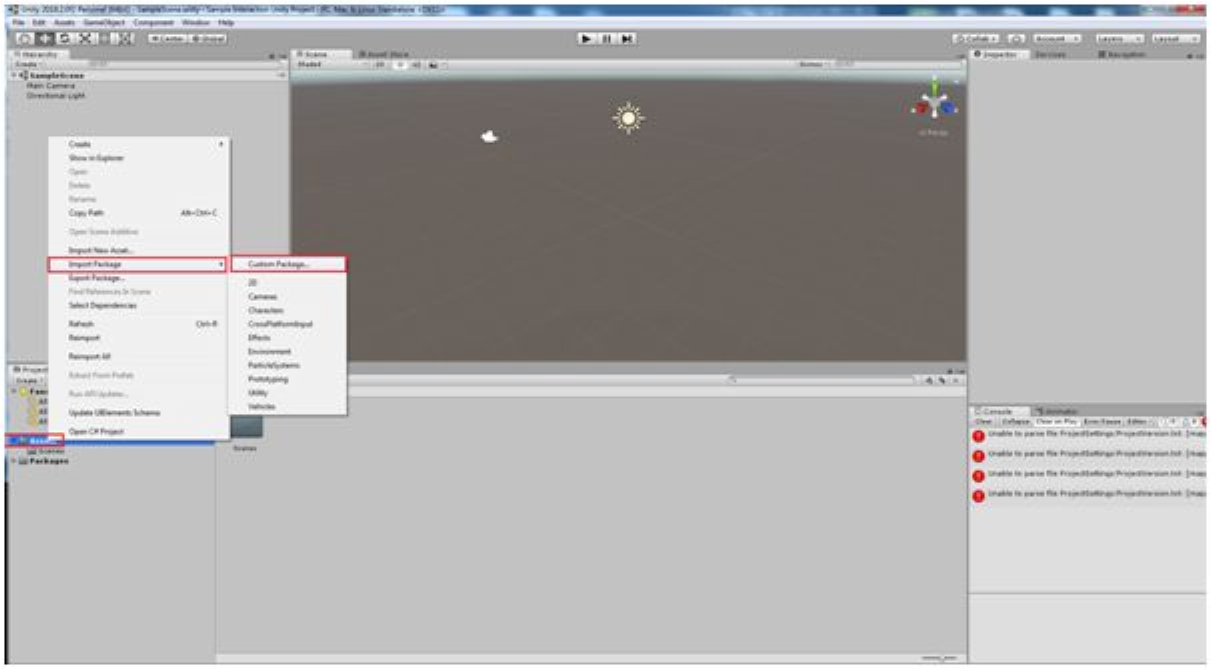
VRInteractionToolkit.unitypackage	4/11/2018 5:51 PM	Unity package file	48,082 KB

Step 4: Next either open your existing Unity Project, or create a blank project. In this example, I've created a empty Unity 3D project.

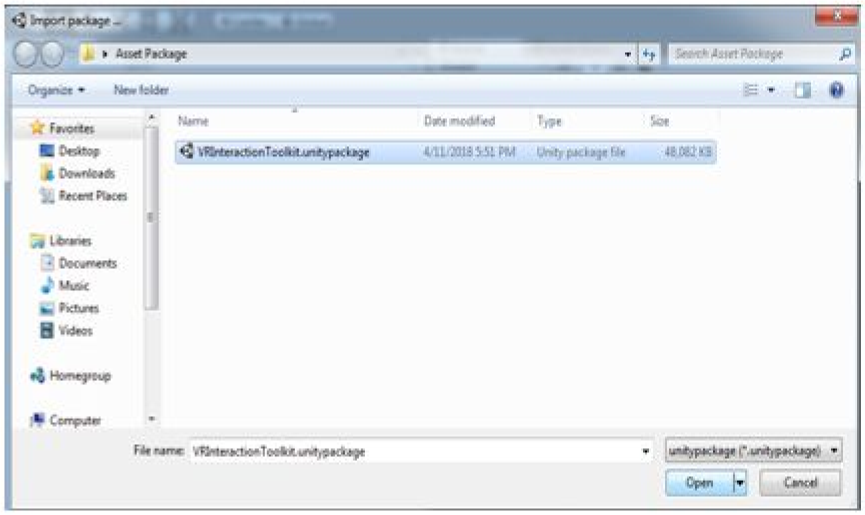


Project name: Sample Interaction Unity Project
Template: 3D
Location: C:\Users\Kieran\UnityProjects
Organization: heyimkieran
ON Enable Unity Analytics

Step 5: After creating/opening your project, right click the Asset folder, click Import Package > and select Custom Package..

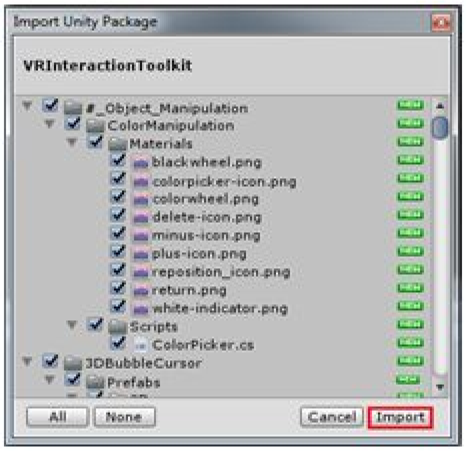


Step 6: Locate to the directory where you saved your VRInteractionToolkit.unitypackage and open it.



Name	Date modified	Type	Size
VRInteractionToolkit.unitypackage	4/11/2018 5:51 PM	Unity package file	48,082 KB

File name: VRInteractionToolkit.unitypackage



```
#_Object_Manipulation
  ColorManipulation
    Materials
      blackwheel.png
      colorpicker-icon.png
      colorwheel.png
      delete-icon.png
      minus-icon.png
      plus-icon.png
      reposition_icon.png
      return.png
      white-indicator.png
    Scripts
      ColorPicker.cs
3DBubbleCursor
  Prefabs
```

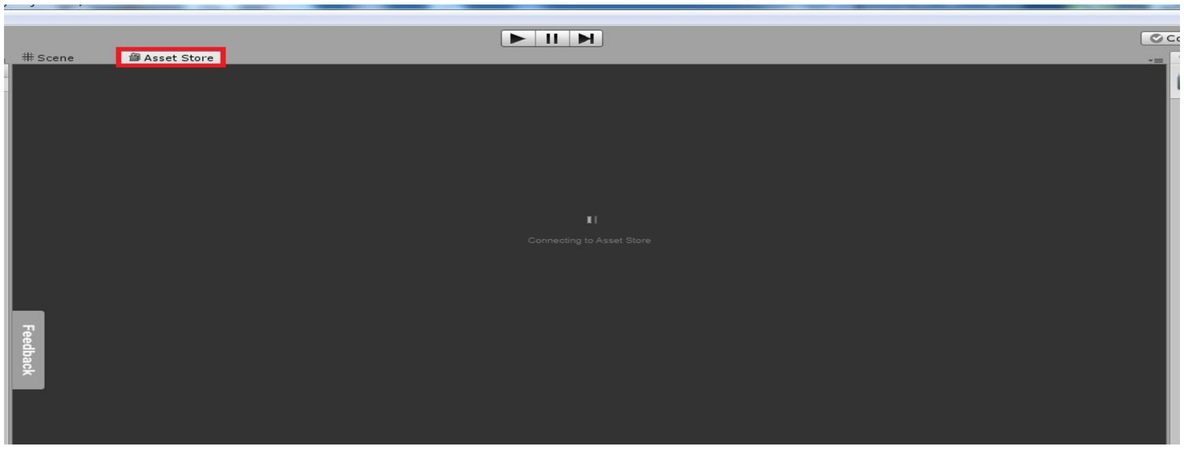
Note: When importing the unity package, you can specify which techniques you want to import into your project.

CONGRATULATIONS, OUR TECHNIQUES ARE NOW PREPARED IN YOUR PROJECT

5.1.2 Installing through the Unity Asset Store

[Note: Currently as of this document, our techniques are not set-up onto the Unity Asset store however, we plan on putting them up soon]

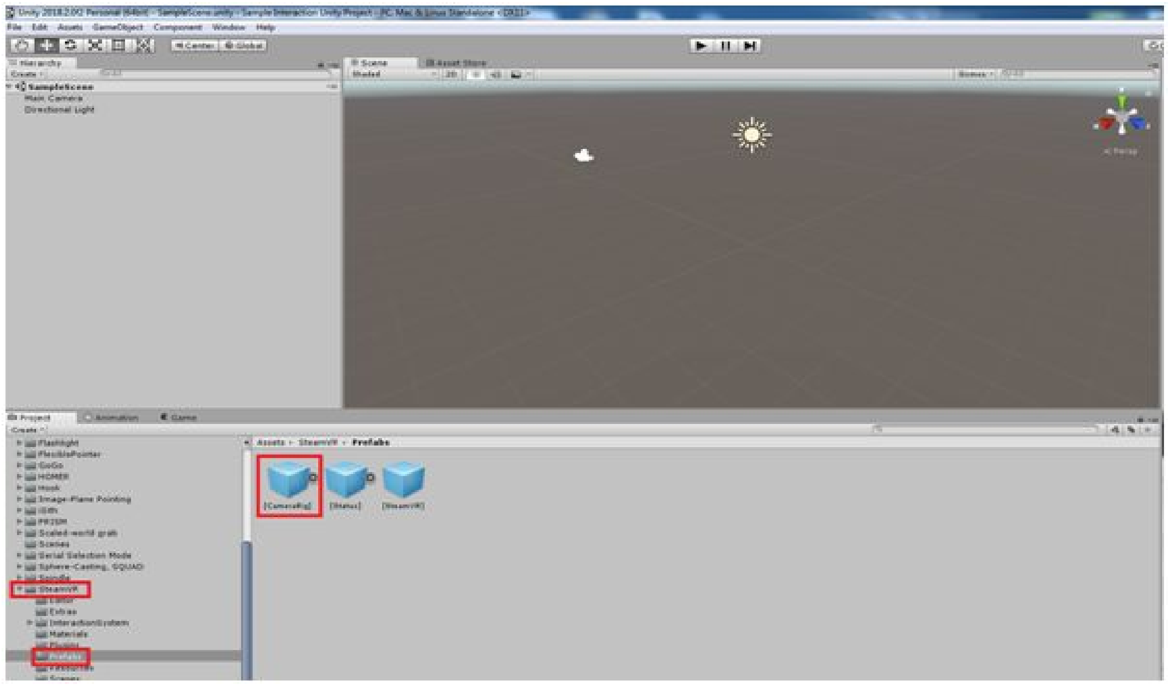
Directions: Installing the project through the Unity Asset Store is quite simple. In the Unity Editor select the 'Asset Store', search for VRInteractionToolkit. Select install, and then import to the project.



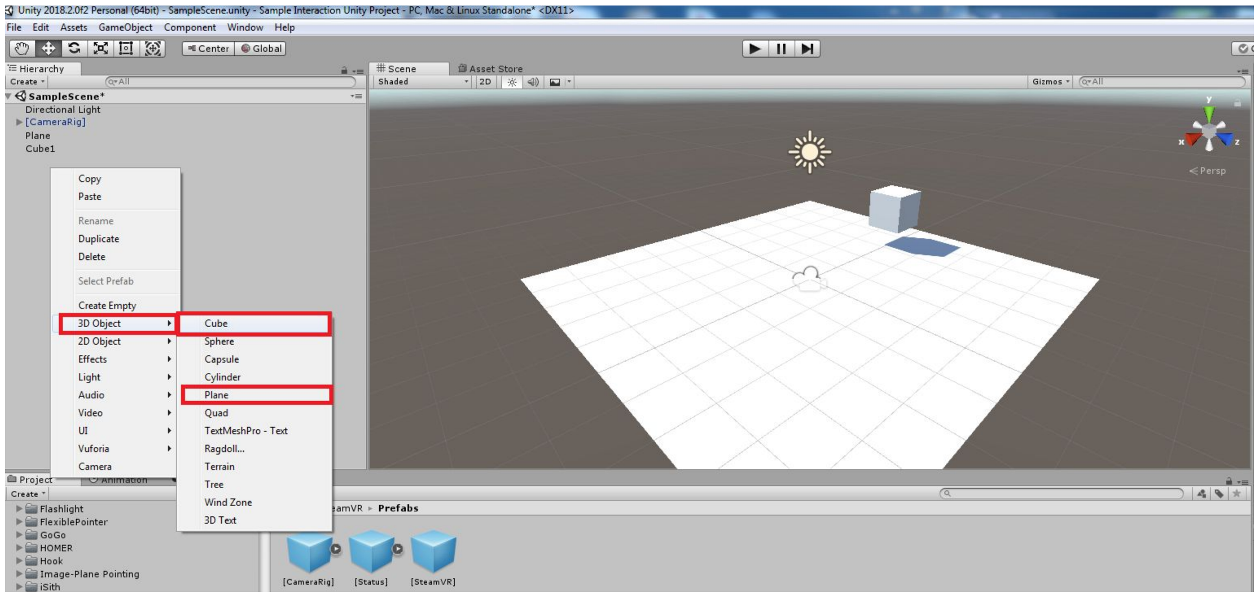
5.2 Setting up with a fresh project

See above on how to firstly import our techniques into a Unity Project

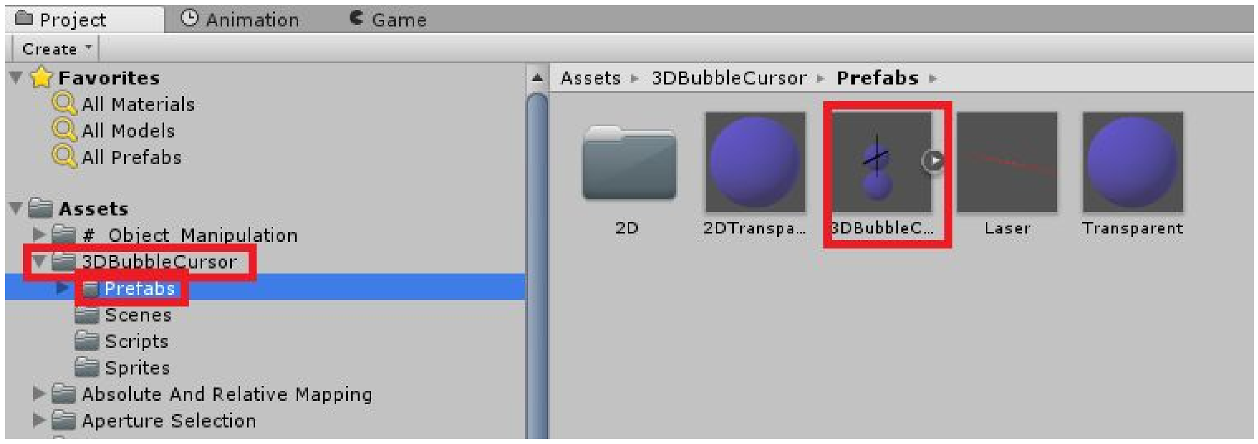
Step 1: Locate the SteamVR Folder, open Prefabs, and drag the [CameraRig] prefab into your scene.



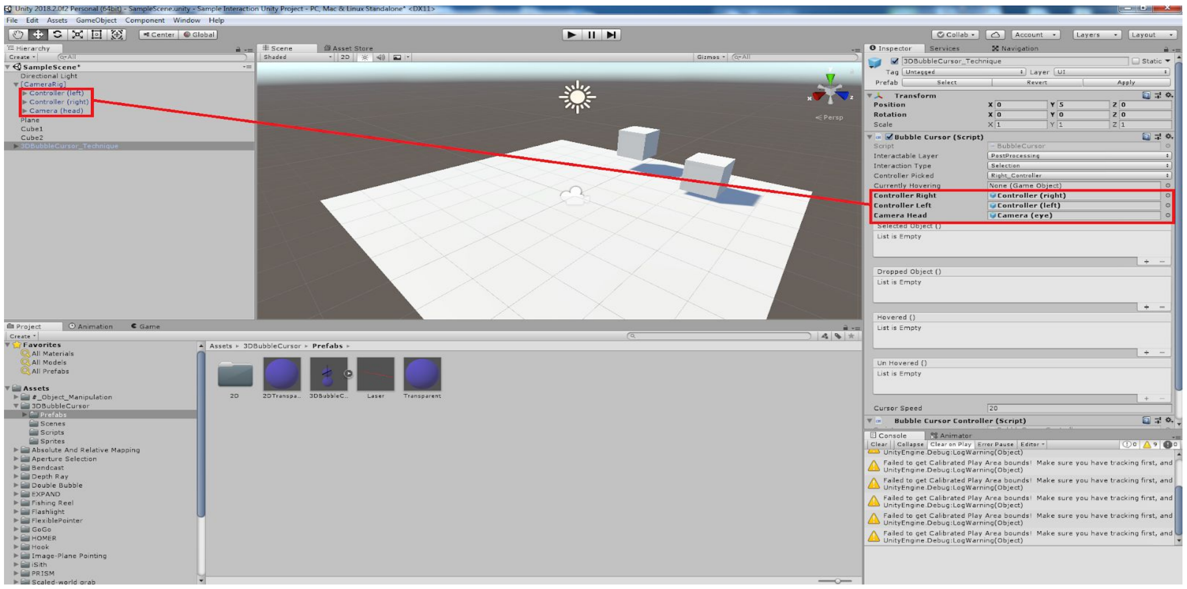
Step 2: Now that SteamVR is setup. Remove the Main Camera from the scene and create some sample GameObjects. Right click a black area in the hierarchy, and create a 3D Cube, and Plane.



Step 3: Now simply to implement the technique into the scene. Find a technique you want to implement in the Assets directory, in this scenario I've picked 3D Bubble Cursor. Open the prefabs folder, and drag the technique into the inspector.



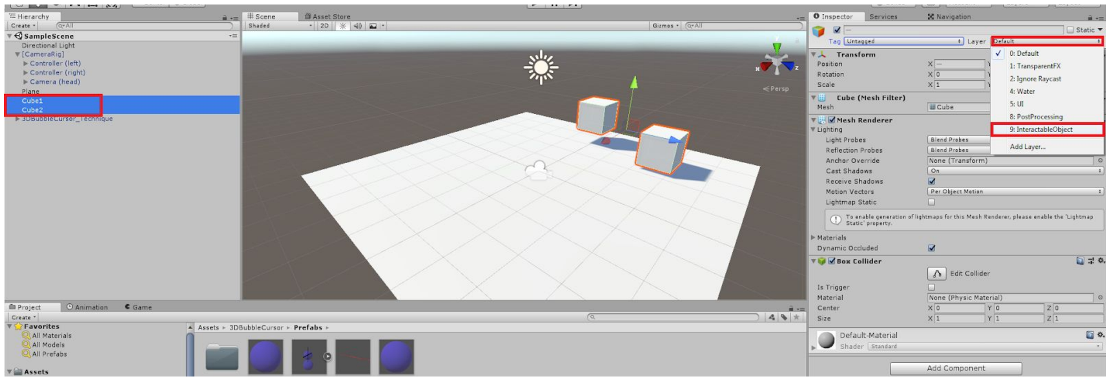
Step 4: Once implemented, you will notice the controllers were automatically picked up and attached to the prefab. However, if for some reason the controllers don't automatically attach, drag the Controllers into the technique Controller variables.



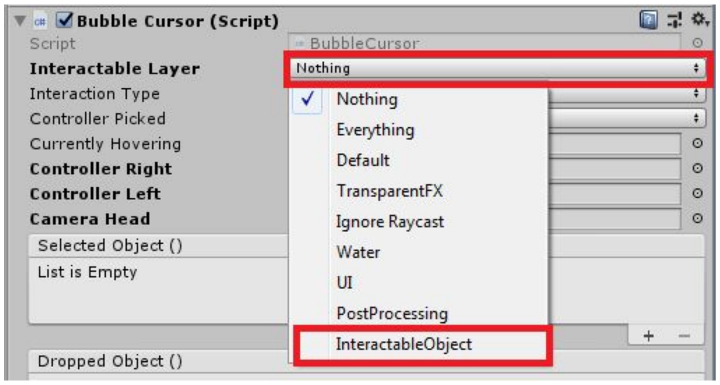
Controller Right: Controller (right)
Controller Left: Controller (left)
Camera Head: Camera (eye)

Failed to get Calibrated Play Area bounds! Make sure you have tracking first, and
UnityEngine.Debug:LogWarning(Object)

Step 5: Ensure that you have specified the layers of a given object in your scene.



Step 6: Set the interactable layer to the layer which your objects are applied on.



CONGRATULATIONS, OUR TECHNIQUES IS NOW SET UP ON A UNITY PROJECT

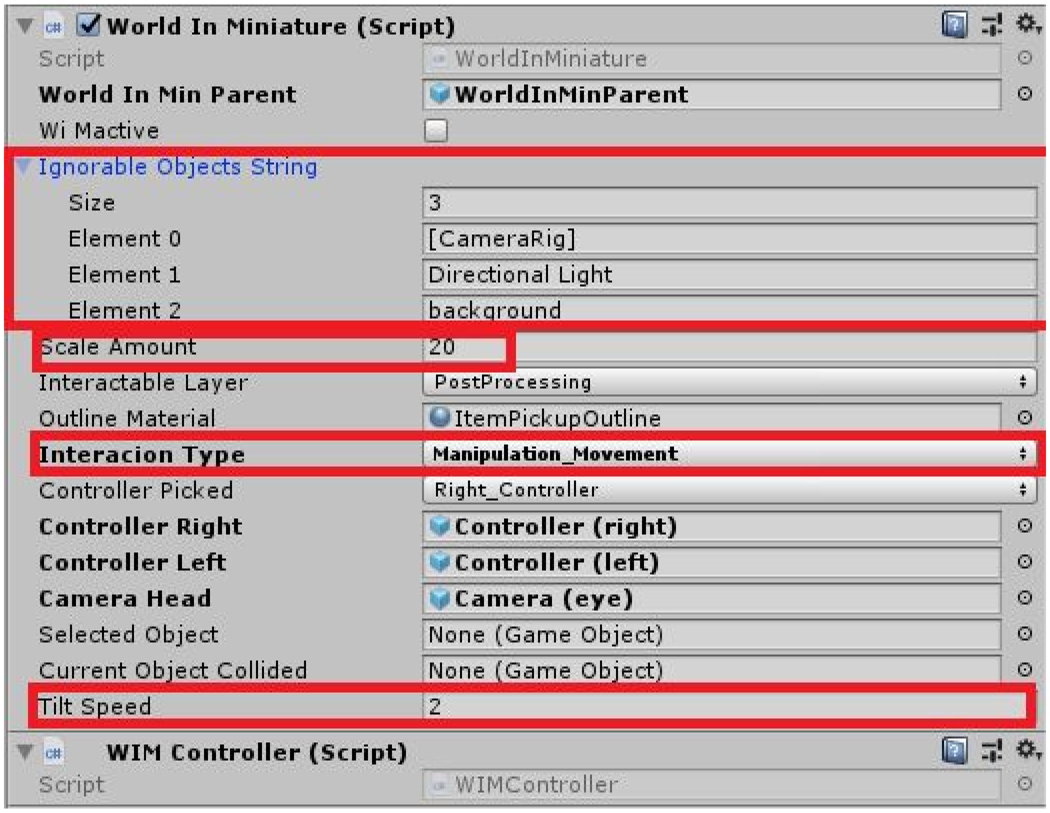
5.3 Setting up with an existing project

Note: Setting up on an existing project is the exact same (5.2 Setting up with a new project)

5.4 Customizing techniques

5.4.1 Customizing through prefab settings

The way we've implemented our techniques in a prefab format allow for people without programming experience to still customize the techniques to tailor it to their Unity scenes through the prefab settings.



World In Miniature (Script)	
Script	WorldInMiniature
World In Min Parent	WorldInMinParent
Wi Mactive	☐
Ignorable Objects String	
Size	3
Element 0	[CameraRig]
Element 1	Directional Light
Element 2	background
Scale Amount	20
Interactable Layer	PostProcessing
Outline Material	ItemPickupOutline
Interacion Type	Manipulation_Movement
Controller Picked	Right_Controller
Controller Right	Controller (right)
Controller Left	Controller (left)
Camera Head	Camera (eye)
Selected Object	None (Game Object)
Current Object Collided	None (Game Object)
Tilt Speed	2
WIM Controller (Script)	
Script	WIMController

Example: World In Miniature

Ignorable Objects String: Developers can specify objects they don't want cloned in WiM

Scale Amount: Developers can set the scale of the miniature world (default = 20)

Tilt Speed: Developers can set the rotational tilt speed (increase faster tilt, decrease slower)

Interaction Type: Developers can choose between Pure Selection and Manipulation

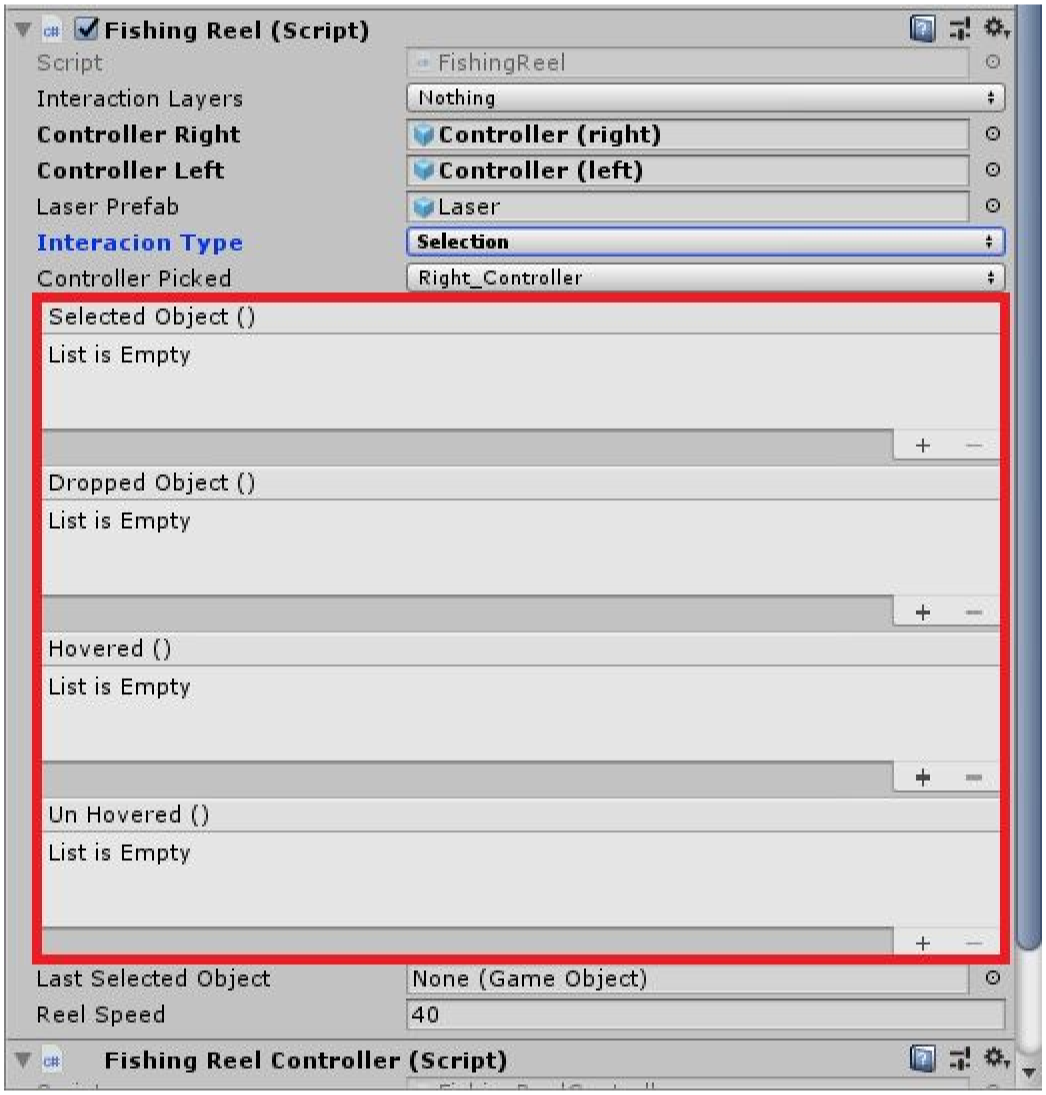
Parameter Information

Details for what all these individual inspector parameters do for each technique are located in 7. Detailed design. A user can also located these details on the main wiki for the project at: https://github.com/WearableComputerLab/VRInteractionToolkit/wiki

5.4.2 Customizing through scripts/events

Software developers can also easily customize or modify our techniques directly through the C# scripts, or add functionality using our event-system setup.

Developers can simply add their scripts into the prefab settings. For example, if a developer wants to do something once fishing reel selects an object, they can drag their method into Selected Object() rather than directly changing our scripts.



Fishing Reel (Script)

Script	FishingReel
Interaction Layers	Nothing
Controller Right	Controller (right)
Controller Left	Controller (left)
Laser Prefab	Laser
Interacion Type	Selection
Controller Picked	Right_Controller

Selected Object ()
List is Empty

Dropped Object ()
List is Empty

Hovered ()
List is Empty

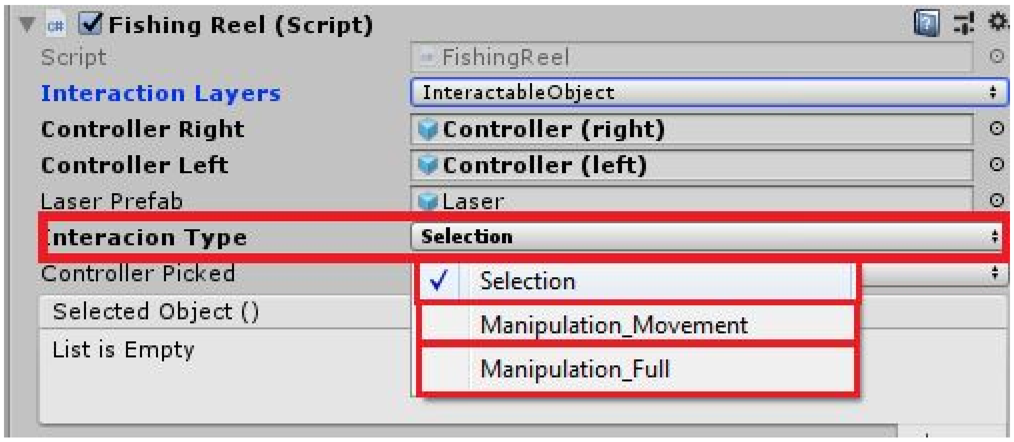
Un Hovered ()
List is Empty

Last Selected Object	None (Game Object)
Reel Speed	40

Fishing Reel Controller (Script)

However, although it's not recommended developers can also directly modify our scripts to change functionality of techniques.

5.5 Technique Interaction Settings

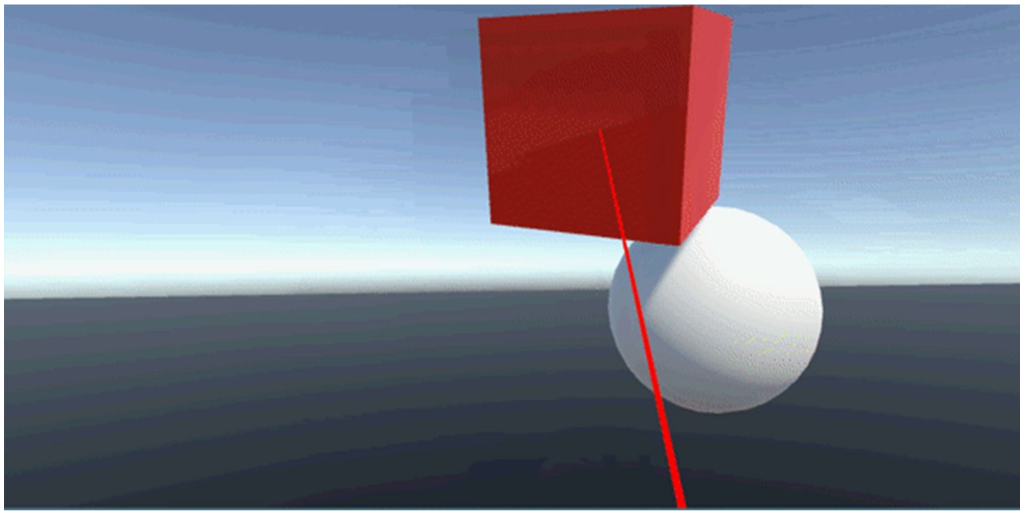

Fishing Reel (Script)
Script FishingReel
Interaction Layers InteractableObject
Controller Right Controller (right)
Controller Left Controller (left)
Laser Prefab Laser
Interacion Type Selection
Controller Picked
Selected Object ()
List is Empty

5.5.1 Pure Selection Mode

In pure selection-mode users select an object, and its stored into a public selected object variable. No manipulation is performed on the desired object.

5.5.2 Manipulation (Movement only)

Manipulation (Movement) is the fundamental principle behind manipulation in VR. All of our selection techniques have a default setting of Manipulation (movement), and all of our manipulation techniques are designed to perform a object movement based manipulation.



5.5.3 Manipulation (Menu)

The Manipulation menu is comprised of 5 categories:

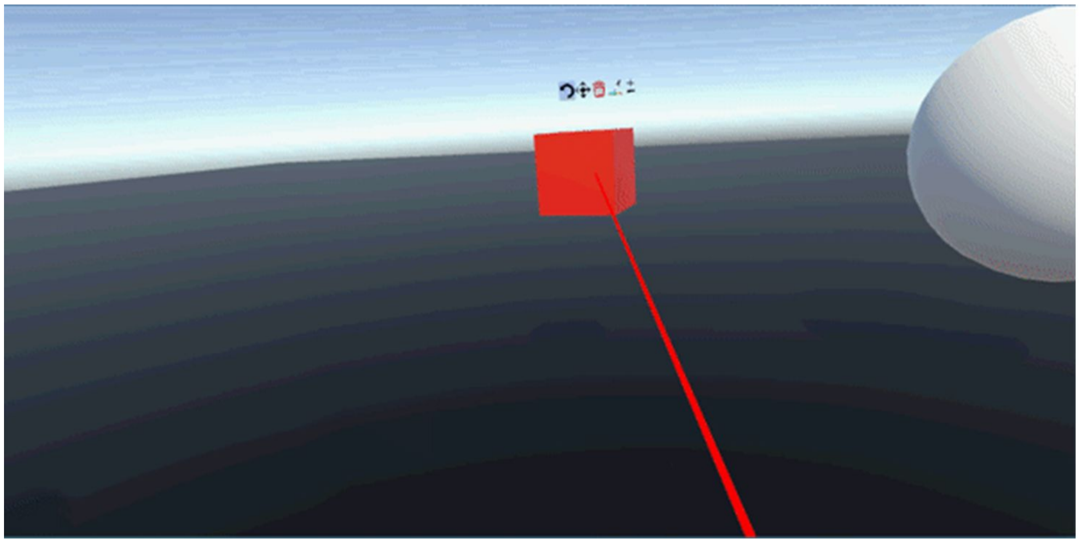
Return: Where a user simply exits out of the manipulation menu and performs no manipulation.

Movement: Where the selected object will attach to the ray, and the user can move its position.

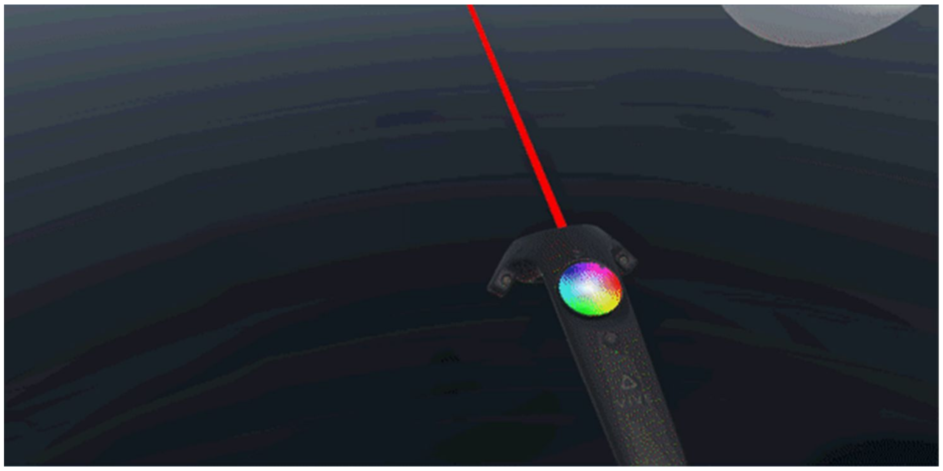
Delete: A selected object will be deleted from the scene.

Colour Picker: A 32bit colour picker will appear on the controller, and the user will be able to change the colour of an object.

Scale Option: Using the HTC Vive touchpad or Oculus Rift joystick, a user will be able to increase or decrease the size of the selected object.



Color picker example



6. Design Overview

The design overview for the techniques below describes the example scenes, an overview of how each technique is meant to work, prototype screenshot of each technique, as well as a reference to the paper that we obtained information about this technique on. Information on how to implement these techniques inside of the Unity game engine is in *7. Detailed Design*

6.1 Example scenes design overview

Each techniques folder within the projects asset files contains the prefabs, materials, scripts etc. as well as a very basic scene with a VR camera-rig, the technique, and some blocks set up to play with. As well as these the project contains a more in-depth example scene for users to test their techniques in. This major example scene is the same for each technique with the only difference being the technique itself. This allows users to compare techniques to see their strengths and weaknesses and have a bit of fun at the same time.

The major example scenes will be located on the same level as the techniques under a “Technique example scenes” folder. This folder will contain the scenes, as we

Requirements of the scene:

Three core testing game components:

- Selection game
    - The selection game allows users to heavily compare all of the selection techniques. There should be multiple of these scattered across the testing scene of different shapes, heights, and sizes.
- Manipulation game
    - The manipulation game allows users to compare the distance/movement aspect of techniques. They should be able to move objects from one space to another to test the difficulty of doing so in different scenarios.
- Docker game
    - The docker game allows users to compare the rotational and fine movement aspects of techniques. This game should have the user rotate/move an object to fit into a silhouette of itself.

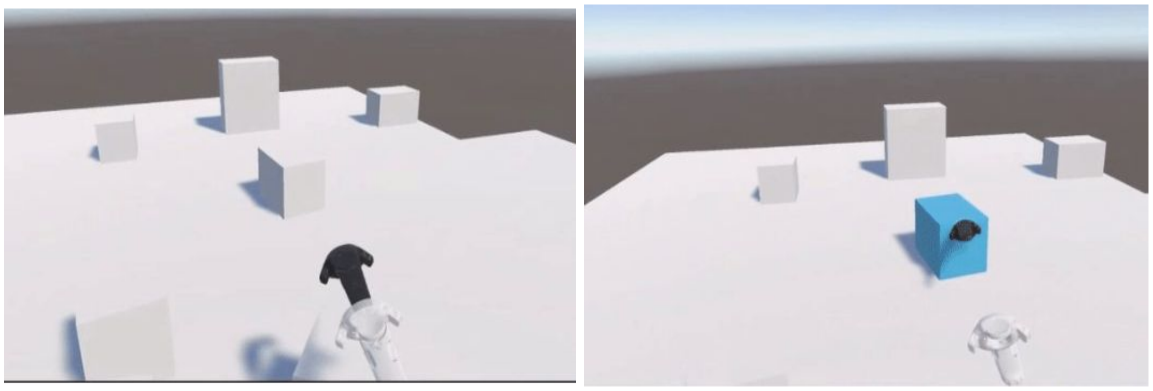
Full implementation located in section 7.1 of detailed design.

6.2 GoGo

This VR interaction technique allows the user to extend their reach beyond their arm length. It does this by utilizing a non-linear mapping between the real location of the remote and the virtual remote. As the user extends out their arm the virtual remote will extend further following this mapping.

Our prototype in unity:

The transparent controller represents the real users controller and the black controller represents the duplicate controller which is manipulated by reaching out causing it to extend past their usual arm length.



Source paper(s) of the technique:

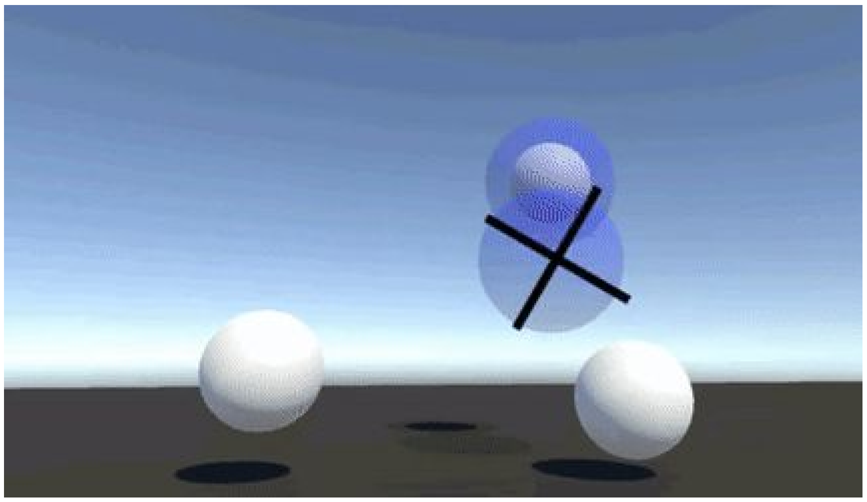
Poupyrev, Ivan & Billinghurst, Mark & Weghorst, Suzanne & Ichikawa, Tadao. (1998). The Go-Go Interaction Technique: Non-linear Mapping for Direct Manipulation in VR. Proc. of UIST'96.

6.3 3D Bubble Cursor

The 3D Bubble Cursor is a redesigned version of the bubble cursor by Vanacken et al (2007) it was extended from the 2D Bubble Cursor which was originally developed by Grossman and Balakrishnan (2005). The 3D Bubble Cursor is a single-target selection technique which consists of a spherical semi transparent sphere which dynamically changes its radius always to encapsulate the nearest virtual object. When the virtual object is not fully encapsulated in the radius, an additional semi transparent sphere appears around the nearest virtual object.

Our prototype in unity:

The crosshair is controlled via a raycast from the controller. As seen in the screenshot, even though the center of the crosshair is not touching the object, a bubble forms around it allowing the user to still be able to select it.



Source paper(s) of the technique:

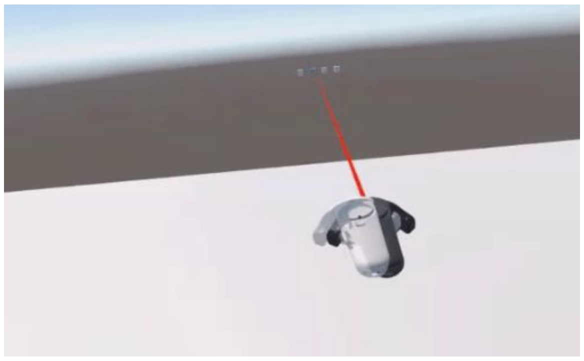
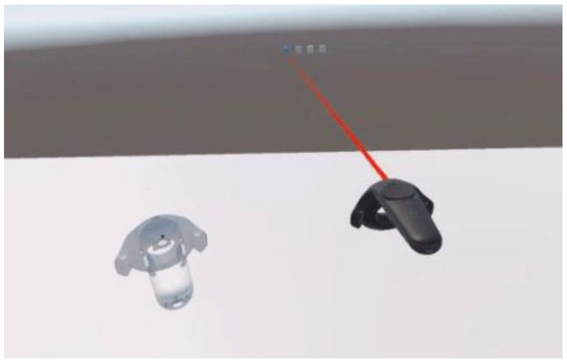
T. Grossman, R. Balakrishnan. [2005] The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area

6.4 Absolute and Relative Mapping

Absolute and relative mapping is in the form of a Raycast (However It could be adapted to a simple hand technique). When you press a set button the movement of the virtual controller relative to your real controller is scaled with a ration of 10:1. By doing this you can be precise when selecting small or distant objects with the ray. This is because the distance you have to move your hand across an object is amplified 10x due to the ratio.

Our prototype in unity:

The transparent controller represents the real users controller, while the black controller represents a duplicate controller. As seen in the first picture the objects are hard to accurately select with ray cast from far away. So the technique is enabled. The user can then move the controller a far distance (10x) for just a small movement of the actual remote. By doing this they can easily select the object.

Source paper(s) of the technique:

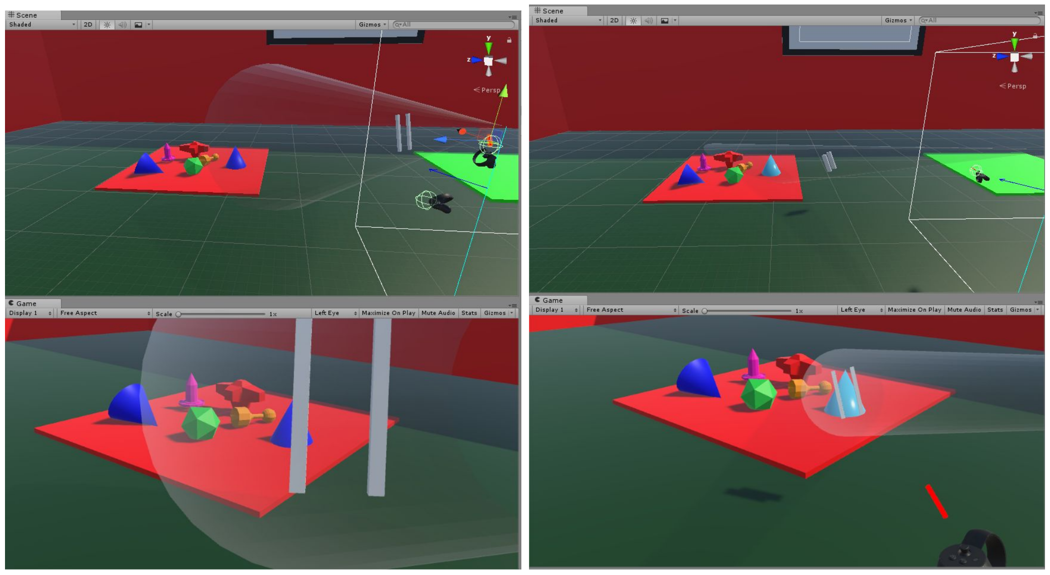
Kopper, R., D. Bowman, M. Silva, and R. McMahan (2010). "A Human Motor Behavior Model for Distal Pointing Tasks." International Journal of Human-Computer Studies 68(10): 603–615.

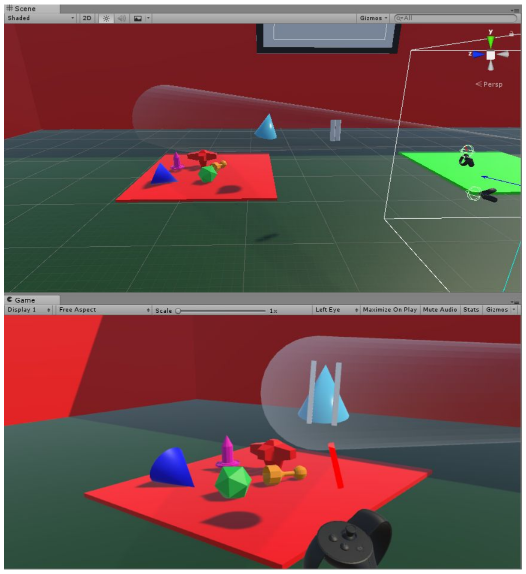
6.5 Aperture Selection

Aperture selection is a modified version of the flashlight. The core modification is instead of a cone originating from the controller it originates from the users camera/view-point. The user can then use one of their hands to control the spread/distance of the flashlight. To select from far away the user would point further away and the cone would grow in size, conversely it will get smaller if you point closer to yourself. for further disambiguation the user can control the rotation of two plates at the end of the cone to allow precise selection between objects inside of the cone.

Our prototype in unity:

As displayed in the three screenshots below of the user's perspective, and the game perspective. The user can extend their controller out to adjust depth. Two plates are also on the cone to allow disambiguation by orientation.





Source paper(s) of the technique:

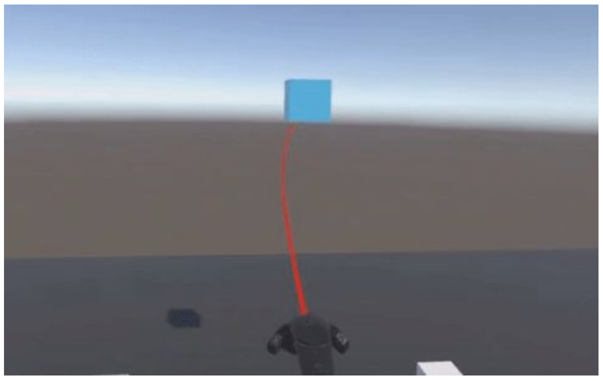
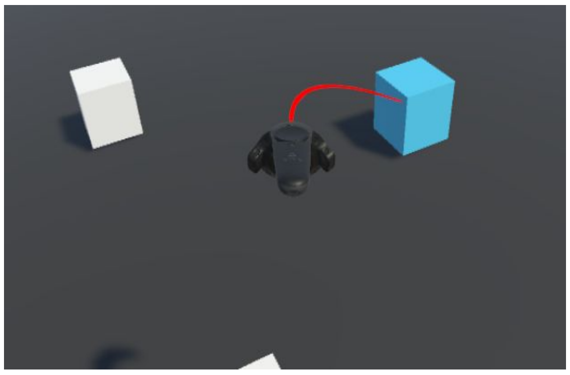
Forsberg, A., K. Herndon, and R. Zeleznik (1996). Aperture Based Selection for Immersive Virtual Environments. Proceedings of the 1996 ACM Symposium on User Interface Software and Technology (UIST '96), ACM Press, 95–96.

6.6 Bendcast

Bendcast is the pointing version of the bubble cursor. By this it uses essentially the same algorithmic approach to distinguish between objects and the difference is how this is visualized for the user to control. The bendcast bends a vector originating from the control towards the closest object to that vector. This is essentially an enhancement of ray-casting as it is implemented as a bendy ray-cast.

Our prototype in unity:

As displayed in the two screenshots below, even though the user is aiming in a direction slightly off from the cube, the ray curves towards it (the closest one).

Source paper(s) of the technique:

Riege, K., T. Holtkamper, G. Wesche, and B. Frohlich (2006). "The Bent Pick Ray: An Extended Pointing Technique for Multi-User Interaction." Proceedings of the 2006 IEEE Symposium on 3D User Interfaces (3DUI '06), 62–65.

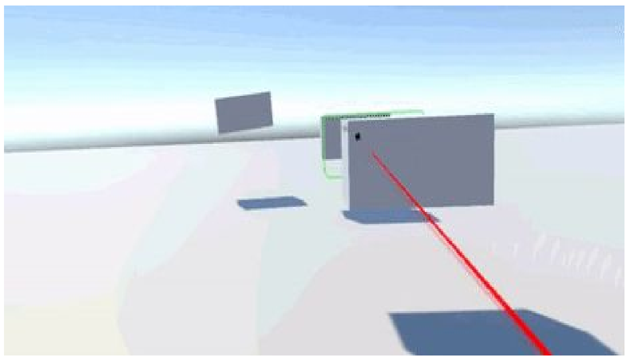
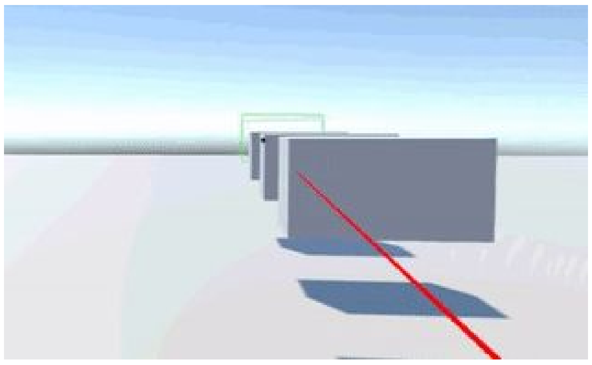
Cashion, J., C. Wingrave, and J. LaViola (2013). "Optimal 3D Selection Technique Assignment using Real-Time Contextual Analysis." Proceedings of the 2013 IEEE Symposium on 3D User Interfaces (3DUI '13), 107–110.

6.7 Depth ray

The Depth Ray designed by Grossman and Balakrishnan 2006; Vanacken et al. 2007 is a 3D pointer enhancement which consists of a depth marker (In our implementation we've chosen this to be a 3D black cube) that is able to determine the closest object when multiple objects are intersected by the pointing vector. The user is able to determine the position of the depth marker based on their finger position on the HTC Vive touchpad or (Oculus Rift joystick - not currently implemented Oculus support yet though).

Our prototype in unity:

As seen in the screenshots below, even though the ray is passing directly though the object in front, the selection is based on the small black cube closest to it (which can be controlled to go back and forth a distance along the laser).

Source paper(s) of the technique:

Grossman, T., and R. Balakrishnan (2006). "The Design and Evaluation of Selection Techniques for
3D Volumetric Displays." Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST '06), 3–12.

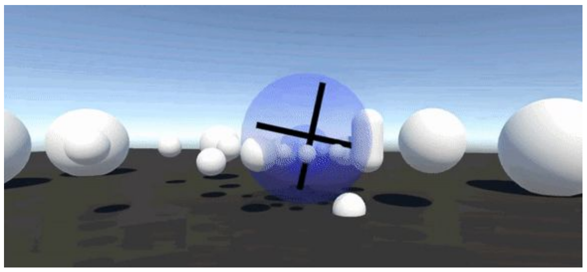
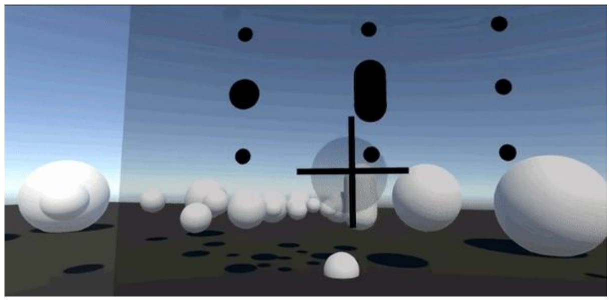
Vanacken, L., T. Grossman, and K. Coninx (2007). Exploring the effects of environment density and
target visibility on object selection in 3D virtual environments. Proceedings of the 2007 IEEE Symposium on 3D User Interfaces (3DUI '07), IEEE, 117–124.

6.8 Double Bubble

The Double Bubble is an extension of the 3D Bubble Cursor but can instead select multiple objects at once with the bubble. After selection, a 2D bubble cursor menu pops up which allows you to select the individual object that you want.

Our prototype in unity:

On the left screenshot the 3D bubble cursor selects a group of objects, and on the right the menu pops up with another bubble cursor to be able to quickly select the object they want from the group.

Source paper(s) of the technique:

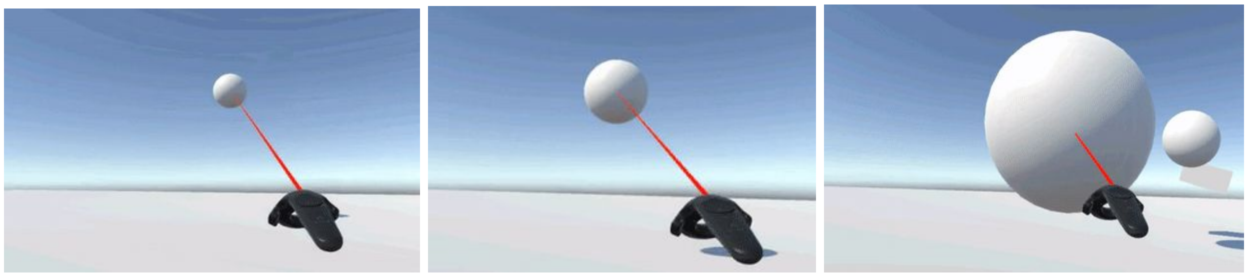
Bacim, F. (2015). "Increasing Selection Accuracy and Speed through Progressive Refinement." Doctoral dissertation, Virginia Polytechnic Institute and State University.

6.9 Fishing Reel

The Fishing Reel technique is a pointing technique extension of Ray-Casting developed by Bowman and Hodges (1997). Much like Ray-Casting, the Fishing Reel technique uses a virtual ray which the user points at virtual objects to select them. After selection, the user is able to reel an object back and forth using an input mechanism.

Our prototype in unity:

User selects the object with a raycast as seen below and with the touchpad pulls the object towards them.



Source paper(s) of the technique:

Bowman, D., and L. Hodges (1997). “An Evaluation of Techniques for Grabbing and Manipulating
Remote Objects in Immersive Virtual Environments.” Proceedings of the 1997 ACM Symposium on Interactive 3D Graphics (I3D ‘97), 35–38.

6.10 Flashlight

The flashlight technique allows a user to select objects with less precision required by a ray-cast. Like a real flashlight a cone volume is projected out of the controller. A user can then select an object by hovering over it with the flashlight.

There are two ways to select between objects if multiple are encompassed by the flashlight:

- If multiple objects are in the volume the one that is closer to the centre line of the cone is selected.
- If the distance between the objects to the centre line are the same then the object closest to the controller is selected.

Our prototype in unity:

The user can easily and quickly select anything within the selection cone. In the right screenshot, the user can distinguish which box they want to select even though 4 are within the cone.

Source paper(s) of the technique:

Liang, J., and M. Green (1994). "JDCAD: A Highly Interactive 3D Modeling System." Computers and Graphics 18(4): 499–506.

6.11 Flexible Pointer

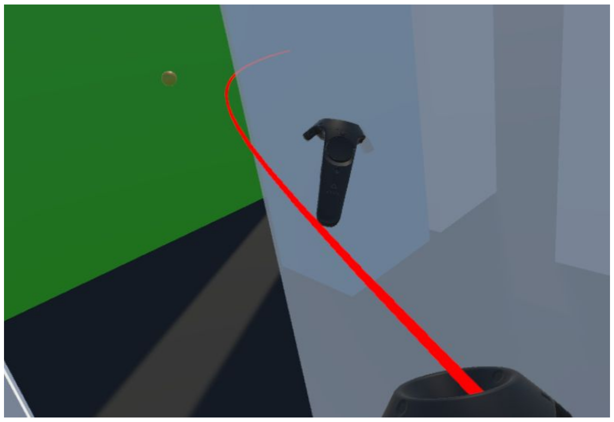
Flexible pointer is an extension of the ray cast. It allows the user to manipulate and bend the projected ray to select obscured objects. It utilises two controllers to control it and the curvature is based on a quadratic bezier curve.

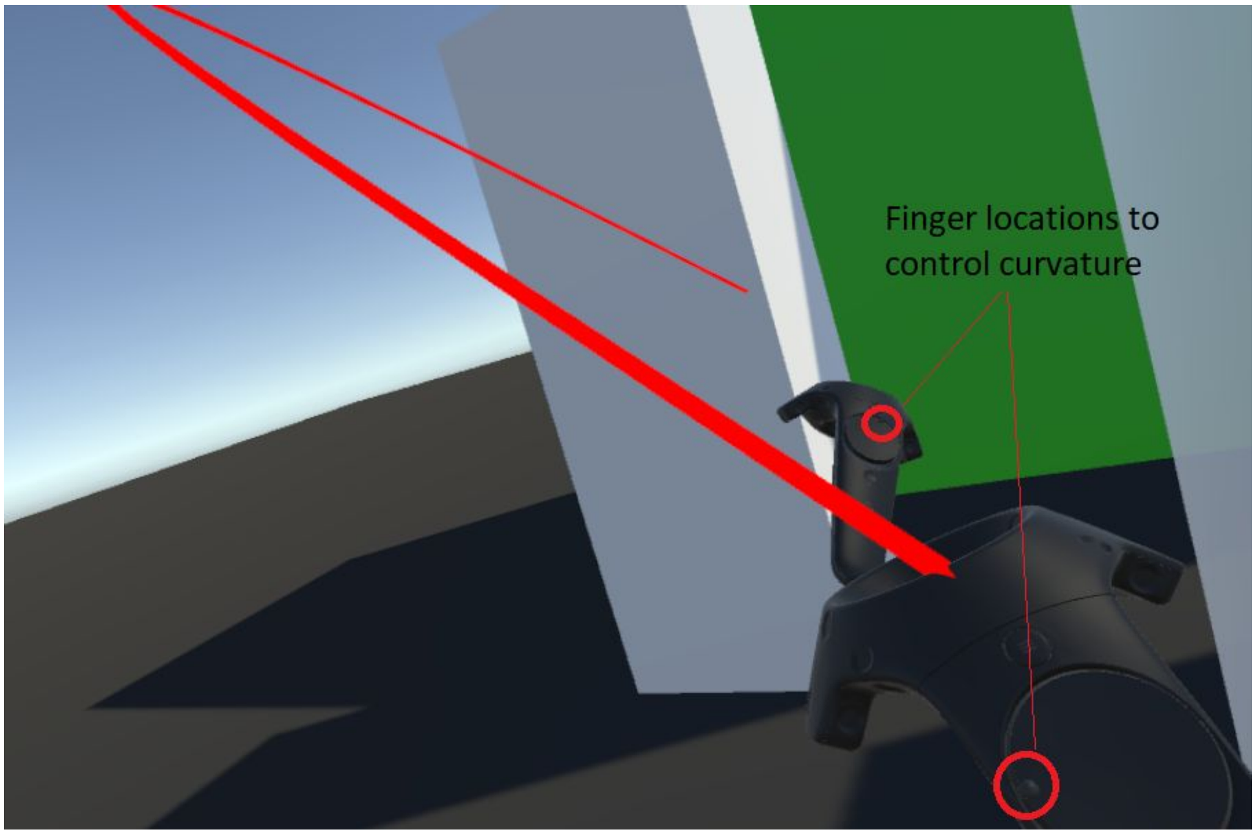
Source paper(s) of the technique:

Olwal, Alex & Feiner, Steven. (2012). The Flexible Pointer: An Interaction Technique for Augmented and Virtual Reality.

Our prototype in unity:

The top screenshot shows the controllers curving the ray around a transparent wall and managing to select the object behind it without touching it. The bottom shows a example of how much the curve can bend.



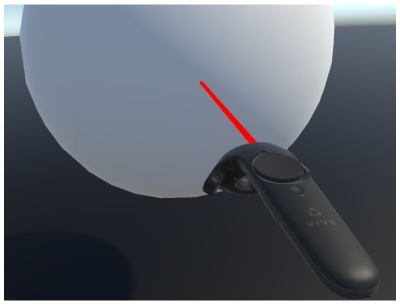
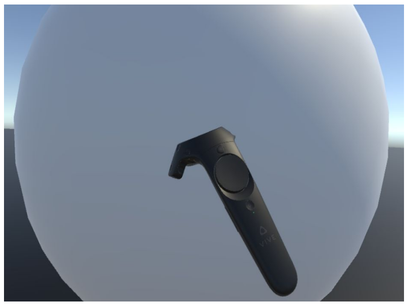
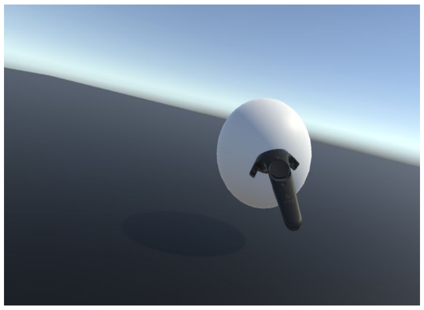


6.12 HOMER

HOMER introduced by Bowman and Hodges 1997 is a Hybrid technique which utilizes ray-casting to make the initial object selection, the user's virtual hand then attaches to the selected object and the technique switches the manipulation mode, which allows the users to re-position the selected object. HOMER uses linear-scaling, tracking the distance between the users physical and virtual hand. This allows for an isomorphic mapping between the users physical and virtual hand, extending the users hand in real-time will manipulate the distance of the object.

Our prototype in unity:

First the object is selected with a ray cast, then the user can move their arm in and out to control the object and its depth from the position its currently in without directly touching it with the controller.

Source paper(s) of the technique:

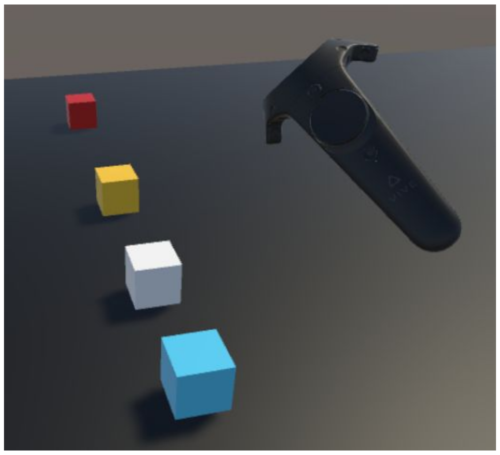
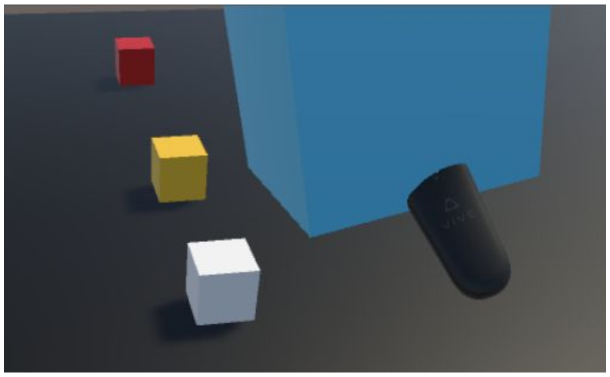
Bowman, D., and L. Hodges (1997). “An Evaluation of Techniques for Grabbing and Manipulating
Remote Objects in Immersive Virtual Environments.” Proceedings of the 1997 ACM Symposium on Interactive 3D Graphics (I3D ‘97), 35–38.

6.13 Hook

On each update hook detects the distance between all selectable objects and the controller. It keeps an ordered list of the selectable objects ranked by a scoring system. Based on this the closest objects will gain score while everything else will lose points. By doing this a user can follow a moving object with the controller and differentiate the object that they want to select by its velocity/movement.

Our prototype in unity:

Closest object is highlighted. On selection it jumps into the user hand.

Source paper(s) of the technique:

Ortega, M. (2013). “Hook: Heuristics for Selecting 3D Moving Objects in Dense Target Environments.” Proceedings of the 2013 IEEE Symposium on 3D User Interfaces (3DUI), 119–122.

6.14 Image-Plane Pointing

Image-plane allows a user to simply grab an object ‘out of the air’ as if it's right in front of you on a 2D plane even if it is in 3D. There are quite a few variations of this technique. We are implementing “sticky finger” and “Framing hands”.

Sticky finger:

Sticky finger allows selection by using an invisible ray cast through the users eye/camera and into the controller (the users finger). By allowing selection like this it gives the illusion that the user is grabbing the object out of the air in 2D.

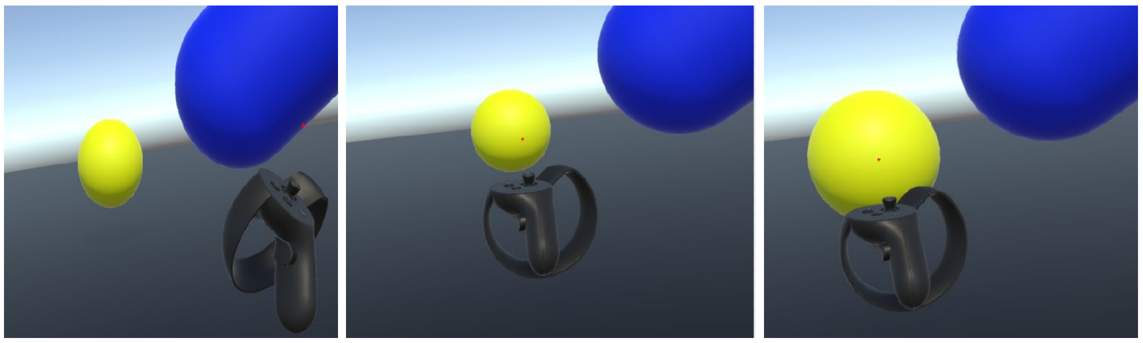
Framing hands:

Framing hands utilizes a similar method and illusion to sticky finger except that the user has two controllers. The user can place one controller above the object, and one below as if they are creating a picture frame of the object. Then when they select the object found in that frame is selected.

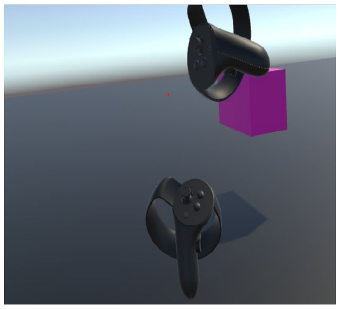
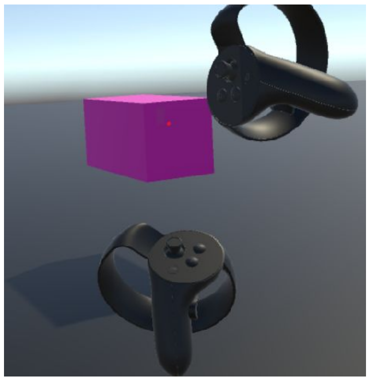
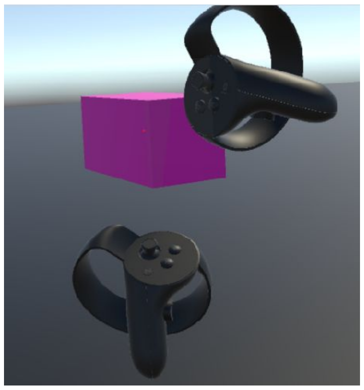
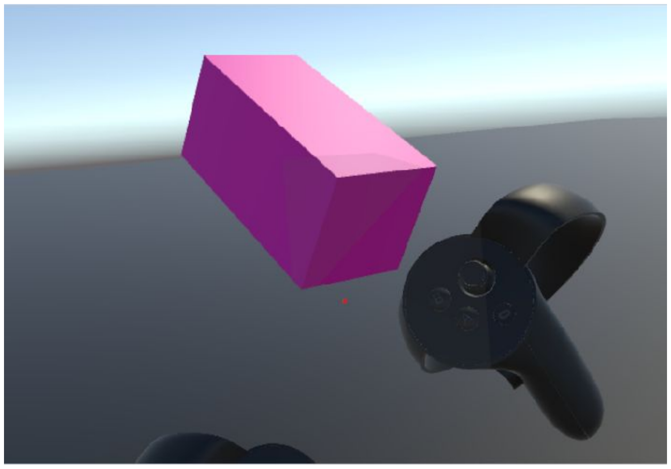
Our prototype in unity:

In the below screenshot examples of sticky finger and framing hands the user selects the object from a distance as if it was on a 2D plane in front of them. A key characteristic is even once the object has changed position to your hand the perspective has not changed.

Sticky finger:



Framing hands:

Notice how in the third screenshot the shadow disappeared? That's when the user selected the object moving its position into their hands.

Source paper(s) of the technique:

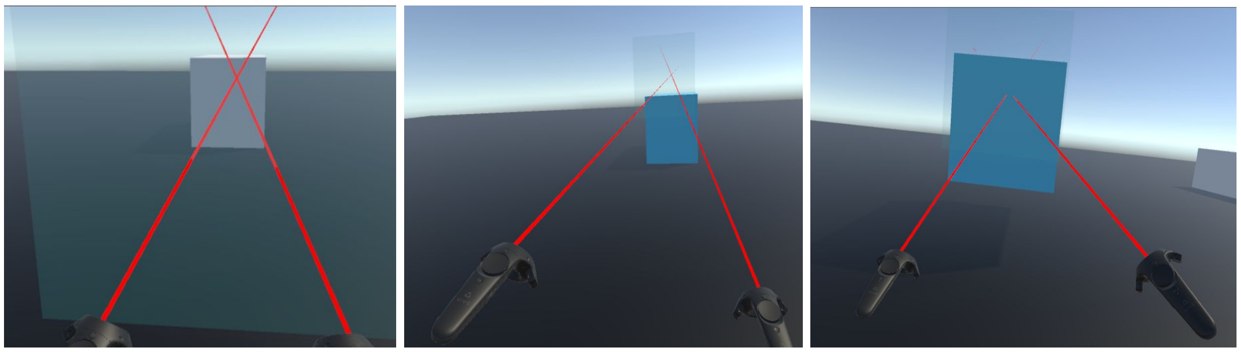
Pierce, J., A. Forsberg, M. Conway, S. Hong, R. Zeleznik, and M. Mine (1997). "Image Plane Interaction Techniques in 3D Immersive Environments." Proceedings of the 1997 ACM Symposium on Interactive 3D Graphics (I3D '97), 39–44

6.15 iSith

The iSith technique utilises both VR controllers. A laser is projected from each controller and then the interaction point is located at point with the shortest distance between each of those lasers. This is calculated by finding the nearest points between two skew vectors and calculating the midpoint between them.

Our prototype in unity:

Using both remotes, the user can control the position of the selection point using the the calculation mentioned above. Doing this they can easily and quickly pick up and move the object in in all dimensions.



Source paper(s) of the technique:

Wyss, H., R. Blach, and M. Bues (2006). "iSith-Intersection-Based Spatial Interaction for Two Hands." Proceedings of the 2006 IEEE Symposium on 3D User Interfaces (3DUI '06), 59–61.

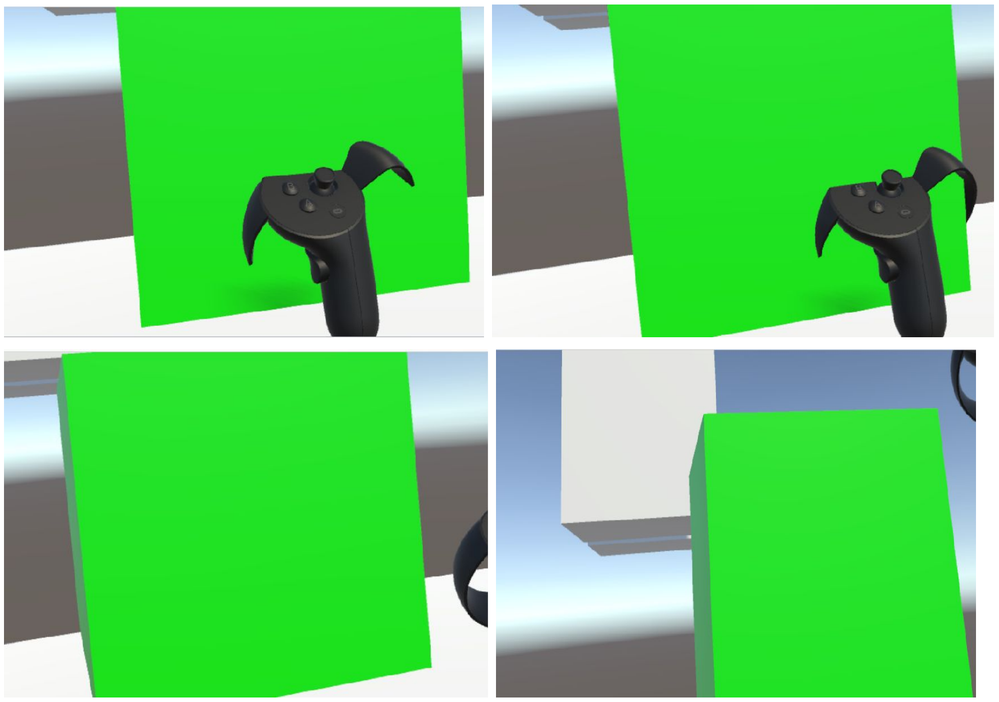
6.16 PRISM

PRISM allows a user to accurately manipulate objects with high precision. After selecting an object a user can precisely move with slow hand movements which are then scaled down relatively. This scale is independent on each of the axis (x,y,z). If a user speeds up their movement the object will "catch" up to the hand with a 1 to 1 ration to it. PRISM also allows this scaling to be applied to the rotation axis to allow high precision in 6DOF.

Source paper(s) of the technique:

S. Frees, G. Kessler and E. Kay, "PRISM interaction for enhancing control in immersive virtual environments", ACM Transactions on Computer-Human Interaction, vol. 14, no. 1, p. 2-es, 2007.

Our prototype in unity:
In the screenshots below the user is always holding onto the object. The scale factor causes it to lag behind to allow precision.



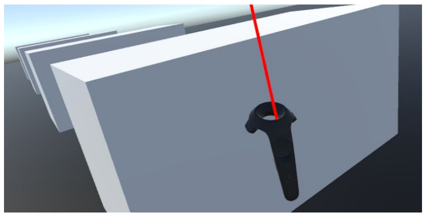
Due to the precisional nature, it is hard to visualize in pictures. A gif of this technique in action is available on the Github wiki page for this project.

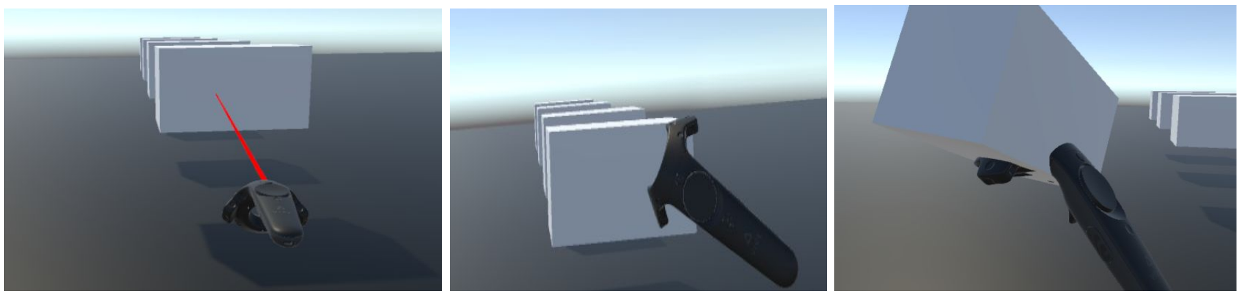
6.17 Scaled-World Grab

For scaled-world grab, the user selects an object far away from them using a long-distance interaction technique (In our case, ray-casting). On selection the entire world will be scaled around the user (without affecting the users viewpoint). This gives the illusion of the user being a giant and using simple touch the user can grab the now-close object and manipulate it around the world with ease.

Our prototype in unity:

User points at cube from far away, on selection the user is scaled up in unity without changing their view. In this form they can just reach out and easily grab the cube.


Normal controller size vs cube for reference.



Source paper(s) of the technique:

Mine, M. (1997). “ISAAC: A Meta-CAD System for Virtual Environments.” Computer-Aided Design
29(8): 547–553.

6.18 Serial Selection Mode

Serial selection mode is a simple technique based on storing multiple objects in a row. When activated, if a user selects an object, it will be added to a list of stored objects (a simple array suffices). This allows them to collect as many objects from the scene as they want. To clear the list the user needs to make an invalid selection such as pulling the trigger when hovering over nothing.

Our prototype in unity:

No screenshot required. User can select any object with any user interaction method. If the controller has this technique attached it will be added to the array etc.

Source paper(s) of the technique:

Lucas, J. (2005). “Design and Evaluation of 3D Multiple Object Selection Techniques.” M.S. Thesis,
Virginia. Polytechnic Institute and State University, Blacksburg, VA, USA.

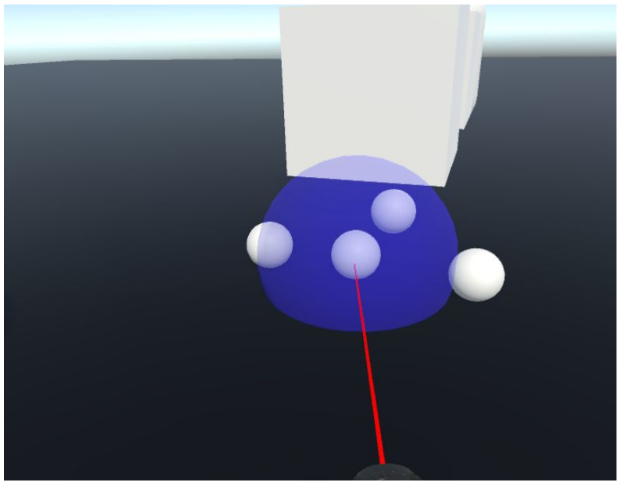
6.19 Sphere-Casting

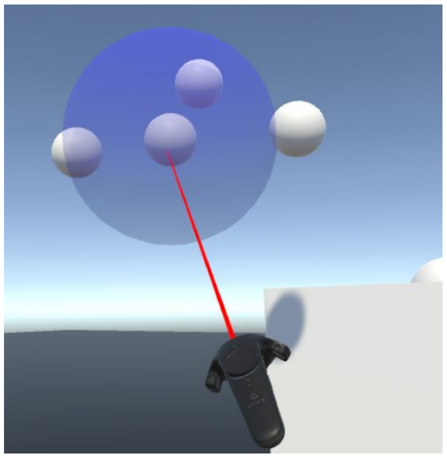
Sphere-casting is a volume-based multiple selection techniques that defines a ray with a sphere-attached to the end of the way which is used to measure the selection radius. The objects that exist within this sphere are considered selectable objects by the user. Kopper et al

(2011) used the Sphere-casting as the pointing technique for the SQUAD technique (Sphere-casting refined by QUAD menu) to select individual objects within the sphere.

Our prototype in unity:

A ray is casted out of the players controller. At the hit point a sphere is displayed (the player can adjust the size of the sphere with the touchpad). everything within the sphere is selected by the player.





Source paper(s) of the technique:

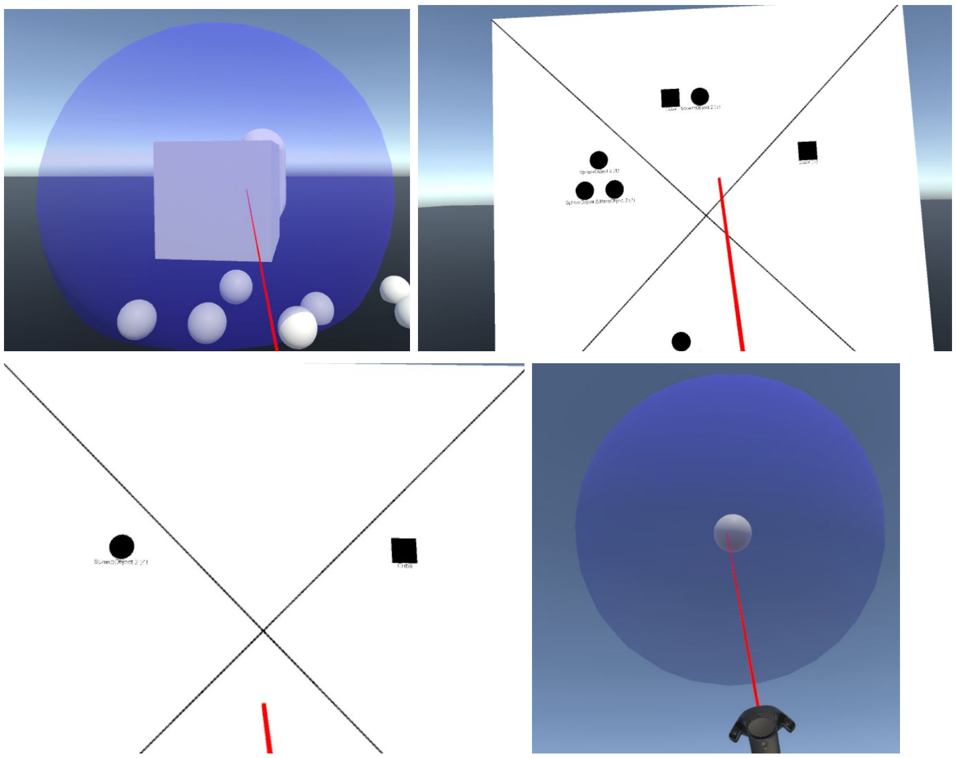
Kopper, R., F. Bacim, and D. Bowman (2011). “Rapid and Accurate 3D Selection by Progressive
Refinement.” Proceedings of the 2011 IEEE Symposium on 3D User Interfaces (3DUI ‘11), 67–74.

6.20 SQUAD

SQUAD is a technique which is to be used with another user interaction technique which selects multiple objects at once. The referenced papers example is with sphere-casting which is another technique we are implementing. On selection of multiple objects SQUAD will display a grid in front of the user with the selected techniques distributed among four quadrants of the grid. The user can then narrow down their selection by choosing one of the four quadrants. Objects in the other quadrants of the menu are discarded and the objects in the selected quadrant are distributed among the four. This is repeated until a single object is selected.

Our prototype in unity:

Player selects a group of objects with sphere casting. The menu pops up and the user can narrow down that selection by clicking on the appropriate quadrant. Eventually they are left with just one object selected.



Source paper(s) of the technique:

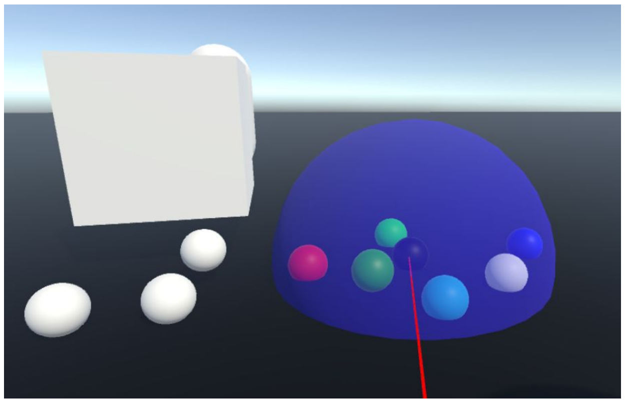
Cashion, J., C. Wingrave, and J. LaViola (2012). "Dense and Dynamic 3D Selection for Game-Based Virtual environments." IEEE Transactions on Visualization and Computer Graphics 18(4): 634– 642

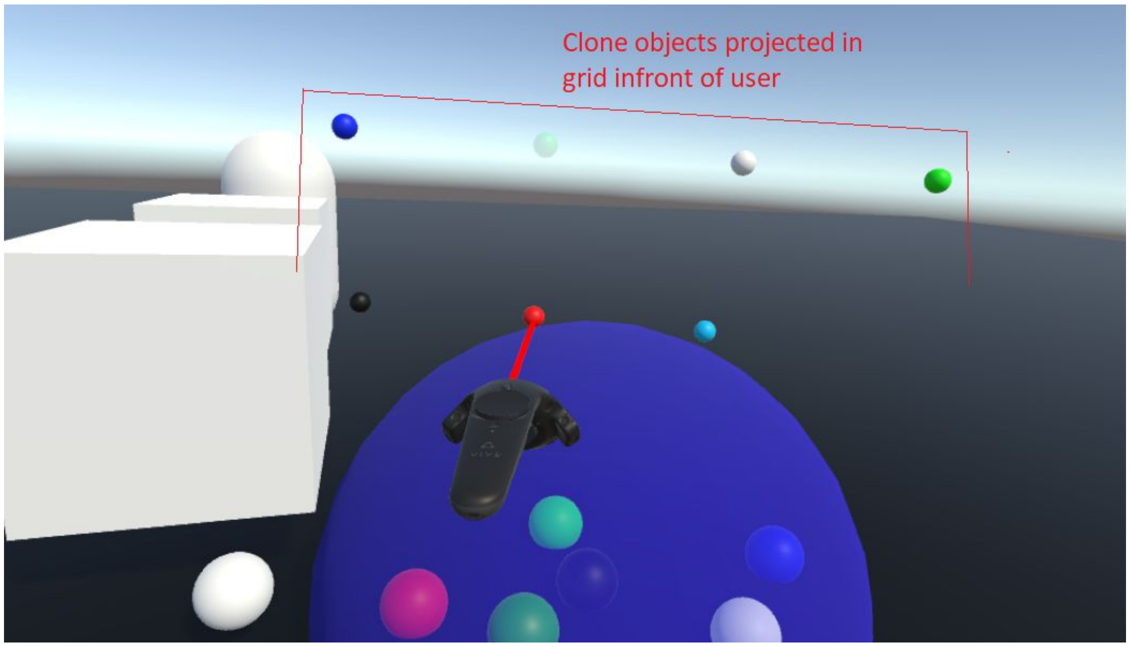
6.21 EXPAND

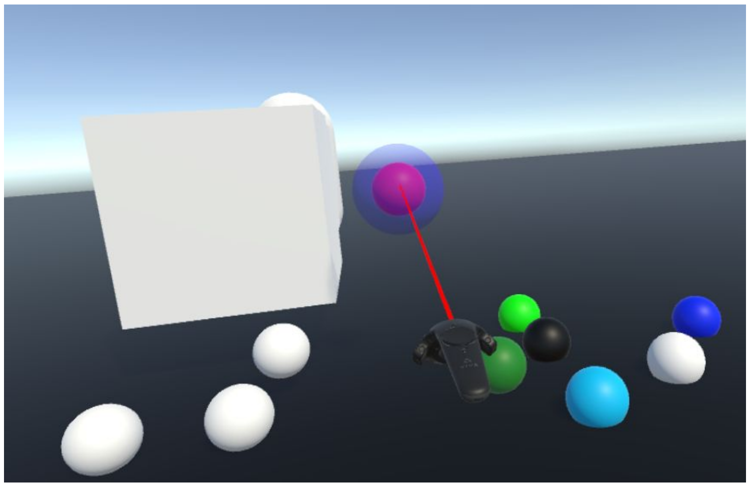
Like the SQUAD technique, expand is utilized when a user selects multiple objects at once. On selection clones of these objects are created and distributed in a grid access in front of the user. The user then selects the object they want from the grid.

Our prototype in unity:

User selects a group of objects with sphere casting, clones of the objects are projected in front of the user, user then selects the cloned red object to use it.




Clone objects projected in grid infront of user



Source paper(s) of the technique:

Kopper, R., F. Bacim, and D. Bowman (2011). "Rapid and Accurate 3D Selection by Progressive Refinement." Proceedings of the 2011 IEEE Symposium on 3D User Interfaces (3DUI '11), 67– 74.

6.22 Spindle

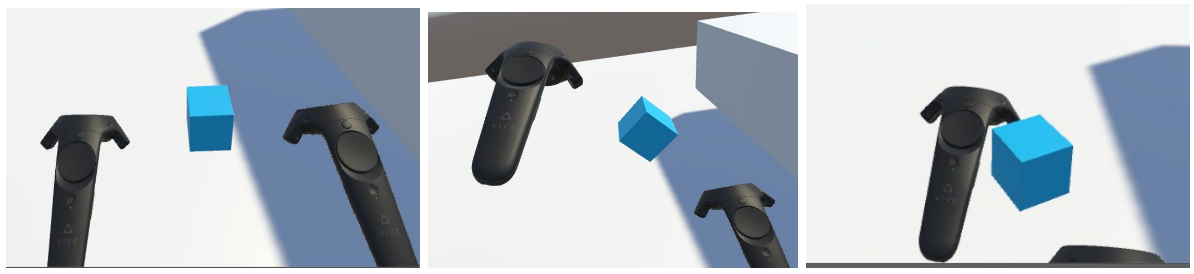
This technique simulates a virtual spindle controlled using two of the VR controllers. The center point of the spindle midpoint of the line between the controllers is the interaction point. Once you select and object both controllers can be moved to manipulate it. As stated by the paper this allows 3-DOF translations.

Manipulations allowed

1. Moving the object while holding
2. Rotating the yaw and roll of the object by rotating the hands relative to each other
3. By changing the distance between the controllers while holding the object you can scale and change the size of the object.

Our prototype in unity:

Screenshots show how you can rotate object on all axis except for pitch.



Source paper(s) of the technique:

Mapes, D., and J. Moshell (1995). "A Two-Handed Interface for Object Manipulation in Virtual Environments." Presence: Teleoperators and Virtual Environments 4(4): 403–416.

Schultheis, U., J. Jerald, F. Toledo, A. Yoganandan, and P. Mlyniec (2012). "Comparison of a Two handed Interface to a Wand Interface and a Mouse Interface for Fundamental 3D Tasks." IEEE Symposium on 3D User Interfaces (3DUI 2012), 117–124.

6.23 Spindle+Wheel

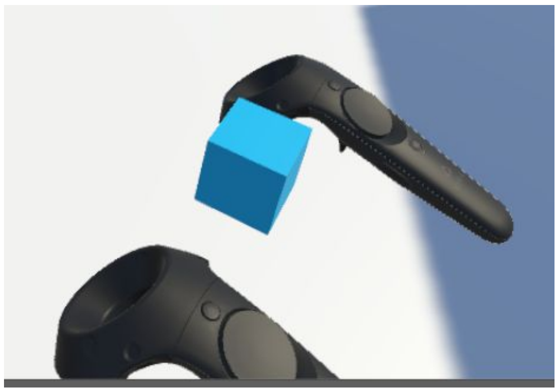
The Spindle + Wheel has the same functionality as the Spindle except it allows you to also rotate the pitch of the selected object. This is achieved by twisting the chosen dominate hand/controller which will in turn rotate the pitch of the object.

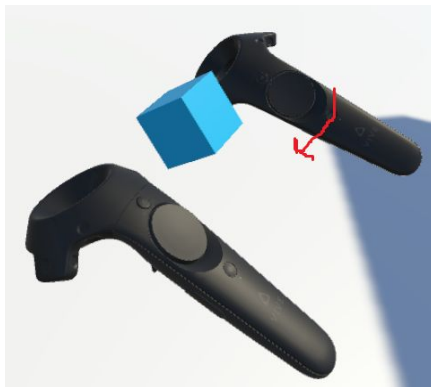
Manipulations allowed

1. Moving the object while holding
2. Rotating the yaw and roll of the object by rotating the hands relative to each other
3. By changing the distance between the controllers while holding the object you can scale and change the size of the object.
4. Rotate an objects pitch by twisting your dominant hand.

Our prototype in unity:

Allows rotation with the exact same methods as pure spindle. Except turning the controller also allows pitch rotation.





Source paper(s) of the technique:

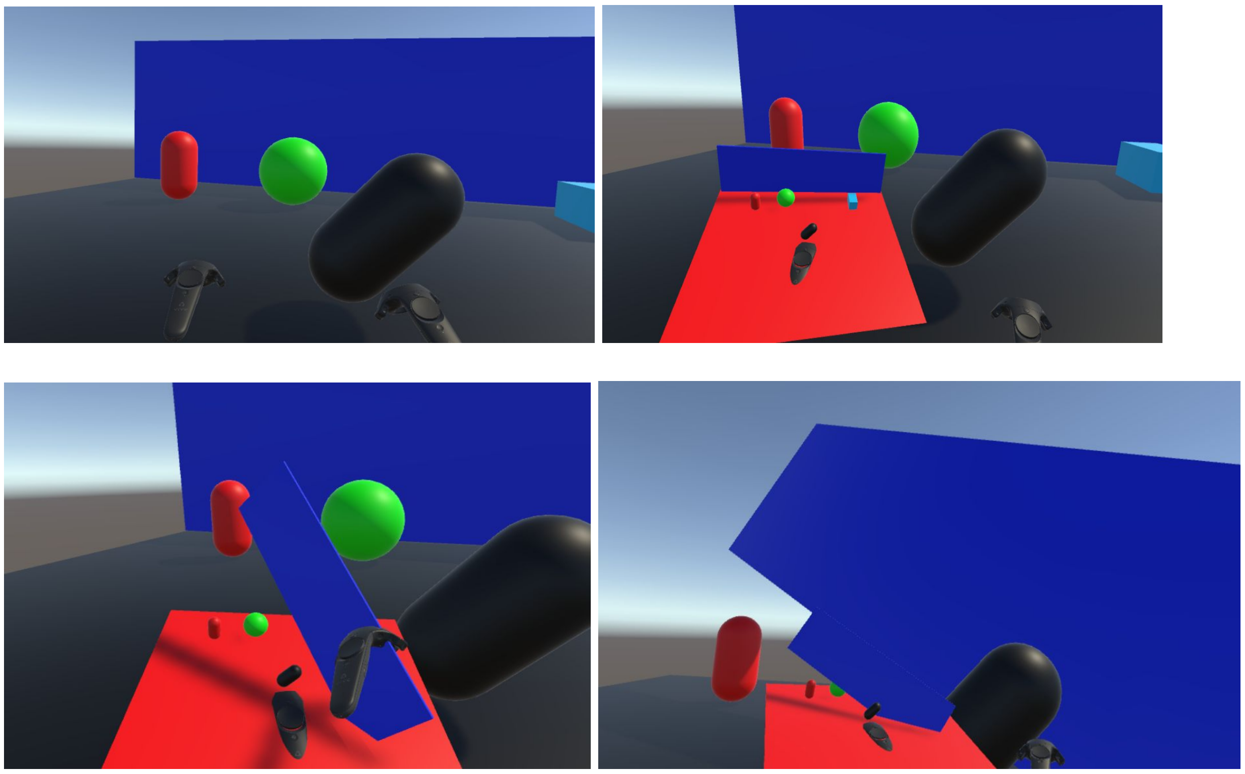
Cho, I., and Z. Wartell (2015). “Evaluation of a Bimanual Simultaneous 7DOF Interaction Technique in Virtual Environments.” Proceedings of the 2015 IEEE Symposium on 3D User Interfaces (3DUI ‘15), 133–136.

6.24 World In Miniature

World-in-Miniature is an indirect selection & manipulation technique developed by Stoakley et al. 1995. The World-in-Miniature technique involves the user having the ability to scale the entire world (or scene) down to have the objects accessible within the user's reach. The user is able to select or manipulate the original object by physically moving the miniature replica of the original object.

Our prototype in unity:

Copies of all the objects in scene are made which can then manipulate the objects in the real world. In the the screenshots below the player first opens the world in the miniature menu, next they grab the blue backboard, move it, and then finally they drop it into a new location. This is reflected in the main scene.



Source paper(s) of the technique:

Stoakley, R., M. Conway, and R. Pausch (1995). "Virtual Reality on a WIM: Interactive Worlds in Miniature." Proceedings of the 1995 ACM Conference on Human Factors in Computing Systems (CHI '95), 265–272.

7. Detail Design

This detailed design provides an overview of the design requirements of the major example scene as well as each individual technique and how they are to be implemented in the Unity 3D game engine.

An overview of the code design of each technique is also provided with relevant Unity Inspector variables. Inspector variables are the public variables (or events) for the scripts that are available for a user to modify in the unity Inspector/GUI. These allow quick customizations and configuration of the techniques for a developer without accessing the code. This gives a further insight into the way these techniques work. If a developer was to going to use these techniques they could look at this design document but the first place they would go to is the Github wiki created for this project which contains similar information about techniques and their code/design.

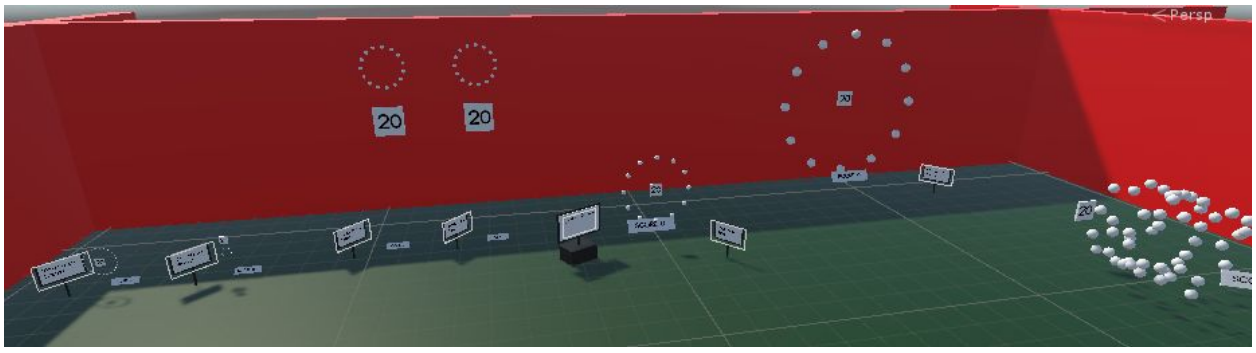
Relevant references to research papers for diagrams and algorithms can be found under the specified technique in 6. Design.

7.1 Major Example scene

Fits Test Selection Game

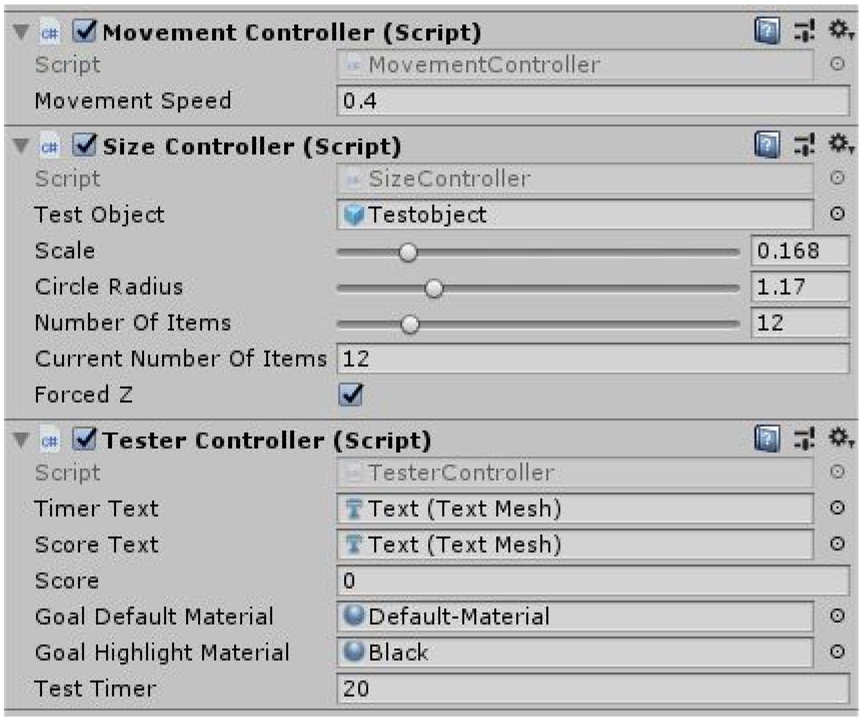
The fits test game to test selection is a variation of the full fits test. To play a user would.

1. Touches their controller against the “Touch to Start!” Sign.
2. A 20 second timer will begin counting down. One of the white balls will highlight black. The user must select the black ball. On selection the score will increase by one and a new ball will highlight black. They will have until the timer runs out to get a score as high as they can.



The screenshot above displays the range of different tests the user can run within the major testing scene. Some are large, tiny and of different shapes. This encourages the user to see for themselves which techniques are best for these different scenarios along with an incentive to get as high as score as possible.

Customizations available within the editor to individual games:



Movement Controller (Script)

Script	MovementController
Movement Speed	0.4

Size Controller (Script)

Script	SizeController
Test Object	Testobject
Scale	0.168
Circle Radius	1.17
Number Of Items	12
Current Number Of Items	12
Forced Z	✓

Tester Controller (Script)

Script	TesterController
Timer Text	Text (Text Mesh)
Score Text	Text (Text Mesh)
Score	0
Goal Default Material	Default-Material
Goal Highlight Material	Black
Test Timer	20

In the above screenshot individual games can be customized further to either create more advanced testing scenes and scenarios or allow the user to build their own.

Key:

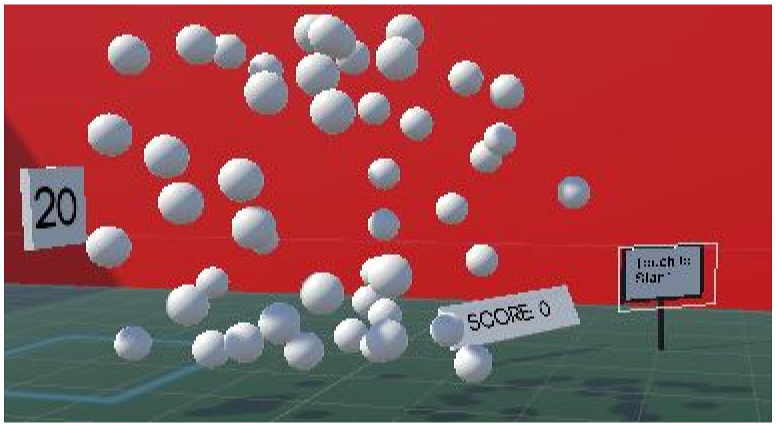
Movement speed: All of these mini-fits games also have the ability to add a movement and speed rotating along the circle. Each ball will move to the original position of its adjacent ball.

Scale: The set scale of individual spherical balls in the test.

Radius: Distance balls are from the center

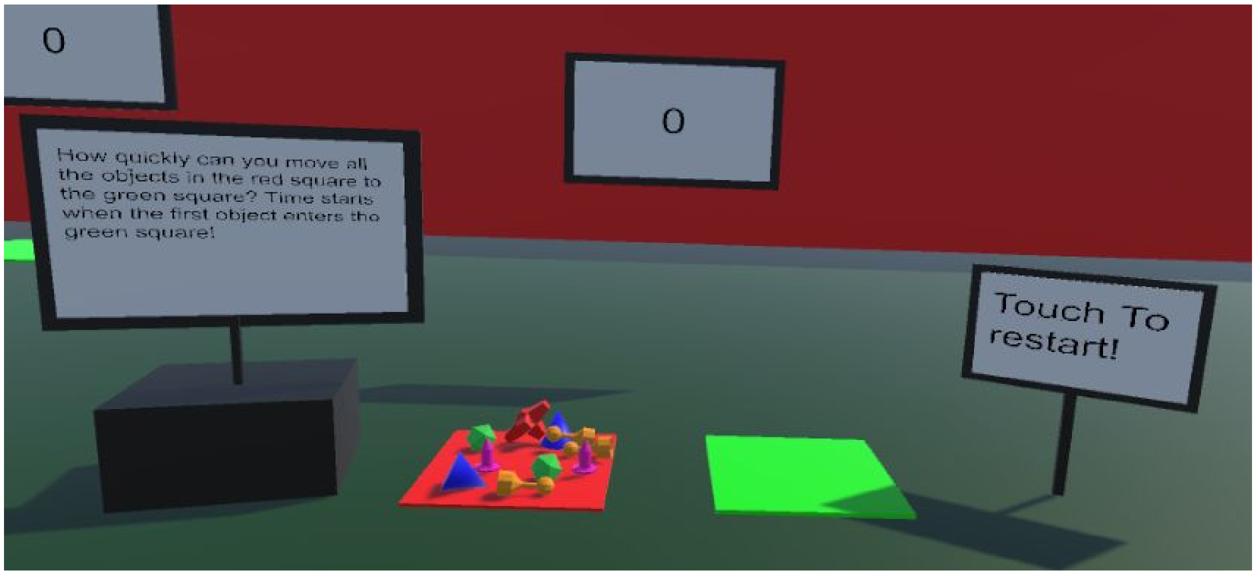
Number of items: Number of balls around the circle

ForcedZ: Forces a circle with no depth. When off a user can drag balls along the z axis to allow different shaped tests. For example like this:



Test timer: The amount of time to accumulate as much score as possible.

Manipulation Red to Green Square Game


0
0
How quickly can you move all the objects in the red square to the green square? Time starts when the first object enters the green square!
Touch To restart!

The manipulation game is simple: Move all objects from the red square to the green square as quickly as possible. As soon as the first object touches the green square the timer starts counting up and will not stop until the green square contains all of the items. Touching the restart sign will reset the timer and place all objects back into their original position in the red square.

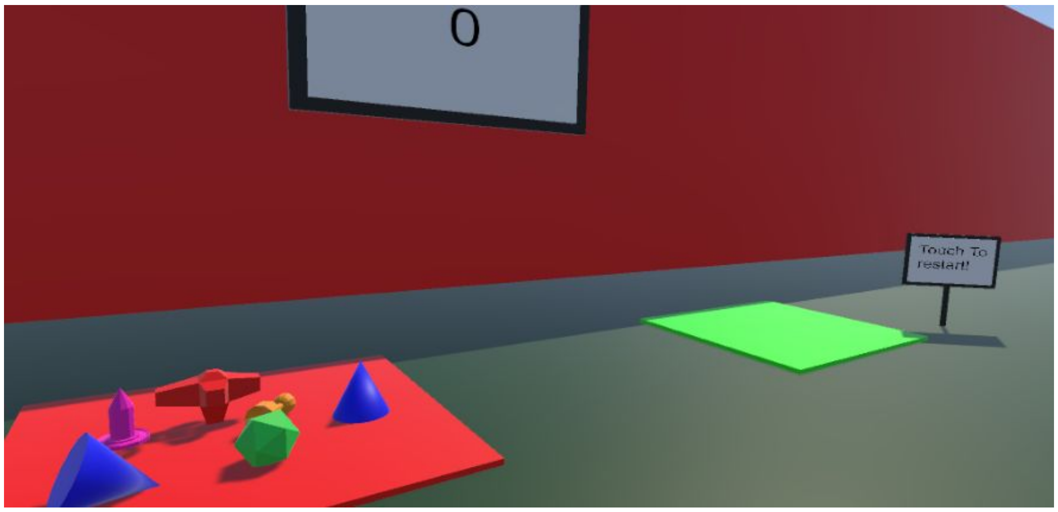
Variations of this game for different levels of testing are easy to create. The user simply just has to move the red square and green square to different locations within the unity editor to change it up.

The major example scene contains three different scenarios of this game to try.

1. Close up (Above screenshot)

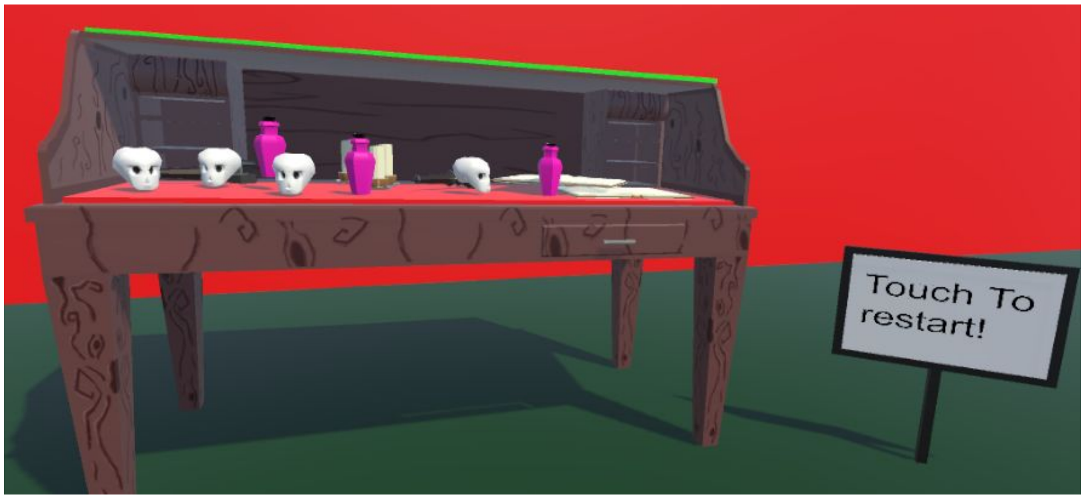
2. Distance:

   Squares are bigger but there is a large distance between them forcing the user to transverse a large distance with the object. (Unless the technique they are using has a way around that which would show its usefulness)

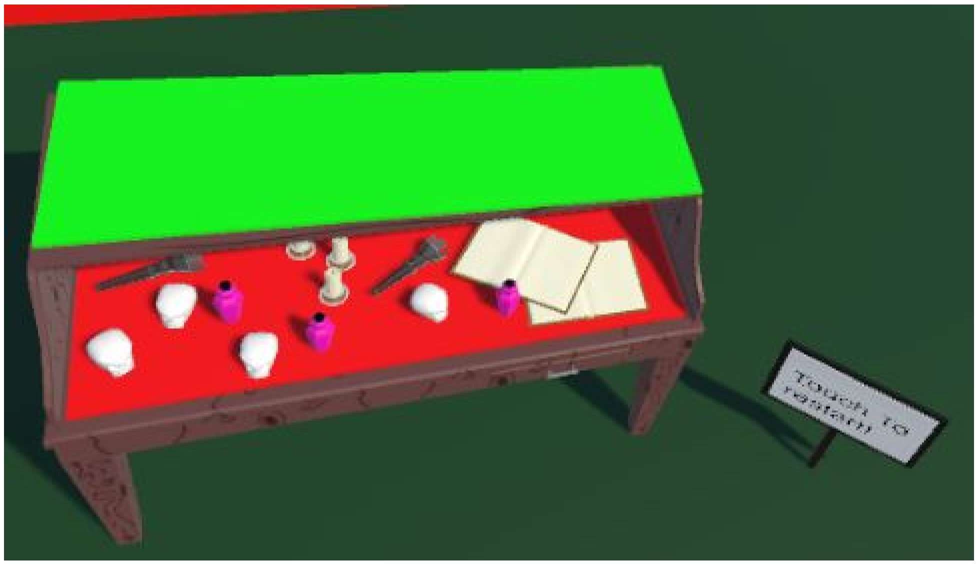


3. Cabinet reach based:

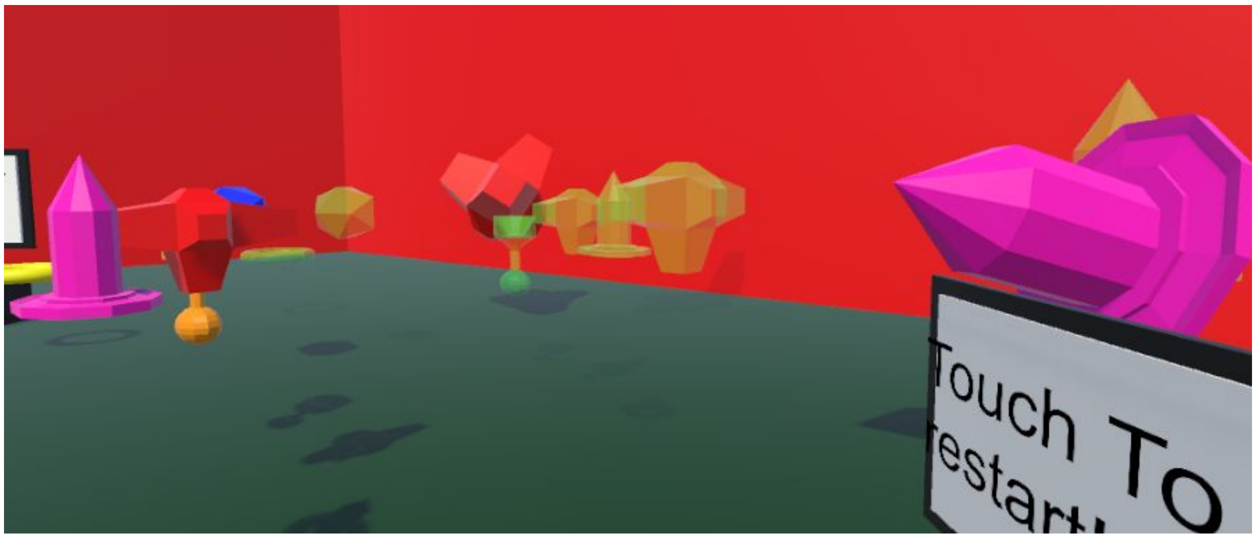
   The red square is on a lower level of a cabinet and the green above. The user will have to grab the objects, pull them out and then reach and put them on top. (As an example some techniques that don't have depth capabilities such as ray-casting would have issues with this showing their weaknesses)



   From above:



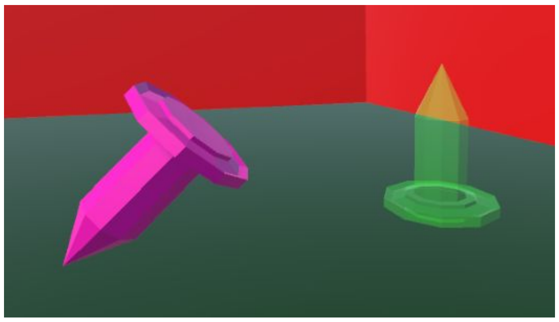
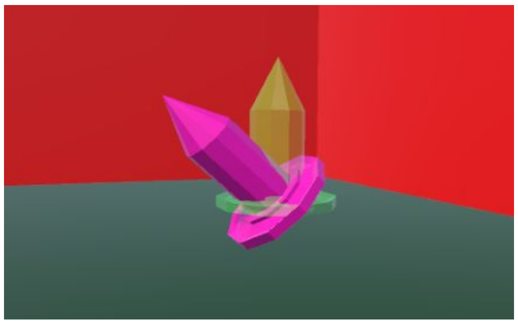
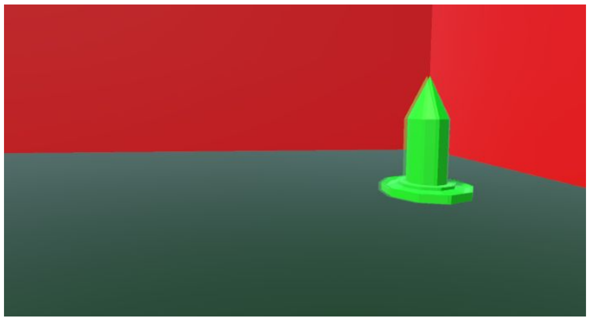
Docker Game



For the docker portion of the major example scene a variety of shapes float in the corner. A user can grab any of the shapes and attempt to reposition/rotate them to fit into their silhouette provided.

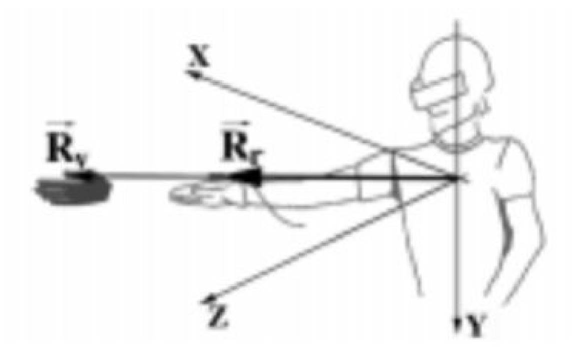
These can also be manually set with a certain error allowance. e.g. rotation can be off by 0.1% and position can be 0.2 units off of the exact location.

Example of completing a docker:

7.2 GoGo

7.2.1 Design Requirements



- Selection and manipulation must work via a simple controller grasp
    - When controller is colliding with an object the user holds down the selection button and the object will be attached at the collision point.
- The gogo must extend beyond the users reach via the following non linear mapping function from the original paper:

$$R_v = F(R_r) = \begin{cases} R_r & \text{if } R_r < D \\ R_r + k(R_r - D)^2 & \text{otherwise} \end{cases},$$

Rv is the length of the arm, Rr is the position from the user to their hand with their arm fully extended. Rv is the length from the position of the user to the extended hand. k is a coefficient between 0 < k < 1. D is to be set at 2/3rds of the users arm length.

- Within unity there will be two options. One where the user can stretch out there are and calibrate their arm length. Or another where that is disabled and the user inputs an arm length in the inspector.

7.2.2 Unity Inspector parameters

For left and right hand gameobject “GoGo Shadow” scripts:

Parameter	Default	Description
Arm Length calculator	Off	When on the user can calibrate their arm length in-game by pressing down on the touch pad while
Arm Length	0.66	When the arm length calculator is off, the arm length is manually set in the inspector.
Distance from head to chest	0.3	GoGo extension is from the chest. This is the manual distance down from the camera head to calculate chest location.
The Model	Model	The controller model which is a child of the controller to be cloned when GoGo is in use
Extension variable	10	The higher the extension variable the further the GoGo will extend at a full arm extension.

For left and right hand gameobject “GrabObject” scripts:

Parameter	Default	Description
Interaction Layers	PickableObject	The layers that the technique can interact with

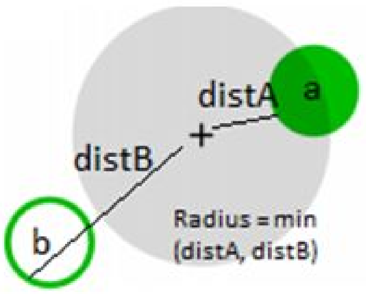
Interaction Type	Manipulation	When set to manipulation GoGo can move objects via fixed joints. When pure selection it will just send an event that it has selected and place the object into a public variable.
Selection	None	The publicly accessible variable showing which object is selected.
Selected Object()	Empty	Event Invoked when an object is selected
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.3 3D Bubble Cursor

7.3.1 Design Requirements

- The 3D Bubble Cursor consists of a cursor attached to the controller which can be reeled back forth by the user with a HTC Vive touchpad or Oculus Rift joystick.
- A 3D semi-transparent bubble encompasses the cursor, and dynamically changes its radius to always encapsulate the nearest virtual object.
- An additional semi-transparent bubble encompasses the nearest virtual object when the radius of the 3D Bubble Cursor cannot increase without intersecting other objects.

The way this has been calculated in our implementation is with the following algorithm:



- The two closest objects to the cursor (a, and b) are determined.
- Intersecting Distance **a** = The distance between the cursor and the farthest edge of the closest object.
- Intersecting Distance **b** = The distance between the cursor and the closest edge of the 2nd closest object.
- Min(M) = The minimum value out of (a, b)
- The **radius** of the bubble is then set to the value M, and the closest object is determined.

7.3.2 Unity Inspector parameters

For Bubble Cursor Script:

Parameter	Default	Description
Intractable Layer	PickableObject	The layers that the technique can interact with
Interaction Type	Selection	When set to manipulation 3D Bubble Cursor can move objects via parenting. When pure selection it will just send an event that it has selected and place the object into a public variable.
Currently Hovering	None	Object that the bubble is currently encapsulating
Selected Object()	Empty	Invoked when object is selected
Dropped Object()	Empty	Invoked when object is Dropped
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over
Cursor speed	20	Speed that cursors depth moves

7.4 Absolute and Relative Mapping

7.4.1 Design Requirements

- The ratio to scale down the mapping between the real controller and the virtual controller must be 10:1.
- This mapping must effect both the rotational and positional values of the controller.
- The mapping can be enabled and disabled at will.
    - On enable the mapping will immediately being with the origin location as the point the controller was in on enable.
    - On disable the controller will snap back to the original location of the device.
- For this implementation the controller projects a visual raycast for selection.

7.4.2 Unity Inspector parameters

For ARMLaser script:

Parameter	Default	Description
Interaction Layers	PickableObject	The layers that the technique can interact with
Interaction Type	Selection	When selected an object will go into a public variable for the player to use. Currently nothing happens on manipulation.
Laser Prefab	Laser	The laser projected from the controller
The Model	Model	The controller model which is a child of the controller to be cloned when ARM is in use
Last Selected Object	None	The last selected object by ARM
Currently Pointing At	None	Object hovered over by ARM
Selected Object()	Empty	Event Invoked when an object is selected

Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.5 Aperture Selection

Design Requirements

- A small raycast will be projected out of the dominate controller so that it is identifiable by the user. (Configurable on and off)
- The volume will originate from the center of the users viewpoint on the camera
- Each frame, the dominant controller calculates the distance between the headset and the dominate controller. This distance along with a customizable amplification value is used to calculate the depth of the volume. This will be achieved by setting the new volumes z scale for the size and then resetting the volumes position so that the point is still on the users eye.
    - This allows the user to dynamically control the volume by pointing the ray out in front of them virtually to match the hardware version described in the original paper.
- The cone volume will have to plates on its end to allow the user to narrow down their selection via orientation.
- All objects within the cone are viable selections until disambiguated.

7.5.1 Unity Inspector parameters

For Aperture Selection Script (Controls the cone volume):

Parameter	Default	Description
Laser Prefab	Laser	Small laser projected from dominate hand to show its the control hand
Laser Container	Lasercontainer	Place to keep the laser in-game

Amplification of length	4	Amplification of how far the cone volume will reach relative to the distance from domiante controller to head

For Aperture Selection Selector Script (Allows the cone to make slections):

Parameter	Default	Description
Interaction Layers	PickableObject	The layers that the technique can interact with
Interaction Type	Manipulation	When set to manipulation Aperture selection can move objects via fixed joints. When pure selection it will just send an event that it has selected and place the object into a public variable.
Selection	None	The currently selected object
Object Hovered Over	None	The object currently hovered over
Selected Object()	Empty	Event Invoked when an object is selected
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.6 Bendcast

7.6.1 Design Requirements

- Visually the bendcast is to be represented by a laser pointer following a bezier curve.

    - This will be achieved via a customizable number of small straight lasers positioned and oriented between each other to form the full curve between the controller and the object.
- Viable objects for selection are calculated as all objects in front of the direction that the controller is facing. The algorithm won't even look at objects behind the controller to avoid the ray bending behind the direction the person is looking.
- Object to bend to is the closest object to a raycast pointing out of the controller. This will be calculated using a formula to find the closest point on a vector to a point.
    - The vector being the vector projected out of the controller and the point being the object.
    - This calculation is made for all potential objects in front of the controller. The distance between these points is calculated for each object and then the object with the smallest distance to the vector is to be bent towards.

7.6.2 Unity Inspector parameters

For Bendcast script:

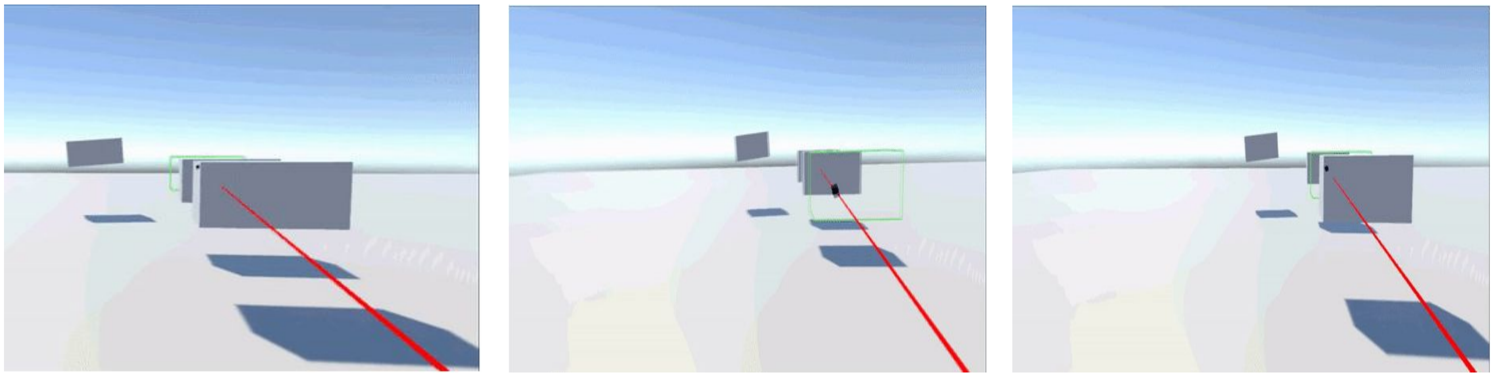
Parameter	Default	Description
Interaction Layers	PickableObject	The layers that the technique can interact with
Set Controller	Right	The controller this technique is for
Interaction Type	Selection	When selected an object will go into a public variable for the player to use. Currently nothing happens on manipulation.
Last Selected Object	None	The last object selected
Currently Pointing At	None	The currently hovered over object
Laser Prefab	Laser	Representing the laser projected from remote
Selected Object()	Empty	Event Invoked when an object is selected

Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.7 Depth ray

7.7.1 Design Requirements

- Depth Ray is an enhanced version of ray-casting used to select occluded objects.
- Much like ray-casting, the user controls a virtual ray casted from the users controllers to use as a point mechanism.
- A depth marker is then attached onto the virtual ray which the user can reel back and forth using the HTC Vive touchpad or Oculus Rift joystick. The closest object to the depth-marker out of the intersected objects is then selected.
    - In our implementation the depth marker is a small 3D black cube easily distinguishable from the virtual ray. A special material is used to for the depth marker to always have it visible in front of objects.



7.7.2 Unity Inspector parameters

For Depth Ray Script:

Parameter	Default	Description

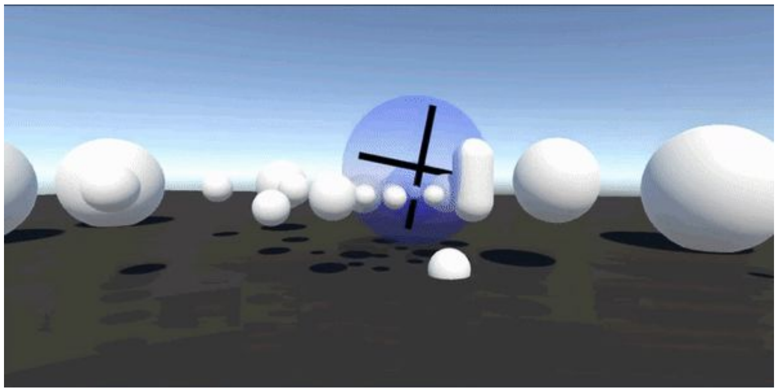
Intractable Layer	PickableObject	The layers that the technique can interact with
Laser Prefab	Laser	The laser projected out of the cotnroller
Interaction Type	Manipulation_Movement	When set to manipulation Depth Ray can move objects via parenting. When pure selection it will just send an event that it has selected and place the object into a public variable.
Controller Picked	Right_Controller	The controller used by the technique
Selected Object()	Empty	Invoked when object is selected
Dropped Object()	Empty	Invoked when object is Dropped
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over
Thickness	0.002	Thickness of the depth ray

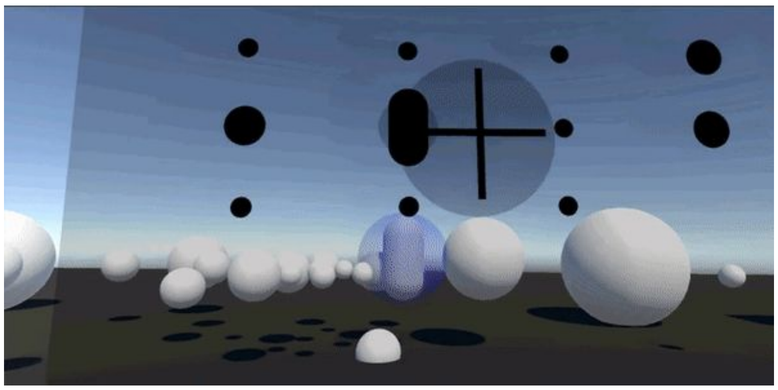
7.8 Double Bubble

7.8.1 Design Requirements

- Double Bubble is an extension of the 3D Bubble Cursor (see 7.3 for design requirements of 3D Bubble Cursor) to select objects in a dense environment.

- The 3D Bubble Cursor is used to select an object however, the radius of the semi-transparent cursor is set to a cap making it unable to shrink below a radius of 1. Therefore, forcing the user to select multiple objects in the Bubble when objects are intersecting or too close together.
- If selecting multiple objects, a 2D menu then appears in front of the user with scaled-down clones of the original objects. A 2D Bubble Cursor is then used to determine which object the user wants to select.
- When hovering over the cloned 2D object, a semi-transparent sphere encompasses the real 3D object as visual assistance for the user.





7.8.2 Unity Inspector parameters

For Bubble Cursor 3D Script:

Parameter	Default	Description
Intractable Layer	PickableObject	The layers that the technique can interact with
Interaction Type	Selection	When set to manipulation after selection of the object via the 2D menu Bubble Cursor 3D can move objects via parenting. When pure selection it will just send an event that it has selected and place the object into a public variable.
Controller Picked	Right_Controller	The controller used by this technique

Cursor speed	20	Speed that cursors depth moves

For Selectable Objects Script that controls 2D menu:

Parameter	Default	Description
Radius bubble	RadiusBubble2D	The 2D Radius bubble for the 2D menu

For Bubble Selection Script to allow the cursor to pick up objects:

Parameter	Default	Description
Intractable Layer	PickableObject	The layers that the technique can interact with
Currently Hovering	None	The object the bubble is currently encapsulating
Selected Object()	Empty	Invoked when object is selected
Dropped Object()	Empty	Invoked when object is Dropped
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.9 Fishing Reel

7.9.1 Design Requirements

- A ray will be projected out of the controller.
- Objects are selected as with the ray-cast technique and are based on the first object colliding with the laser.
- On selection (and in manipulation mode) the user should be able to manipulate the location of the object up and down the ray via the controllers touchpad (or analog sticks for the oculus)

7.9.2 Unity Inspector parameters

For Fishing Reel Script:

Parameter	Default	Description
Intractable Layer	PickableObject	The layers that the technique can interact with
Laser Prefab	Laser	Laser representing the fishing reel which is projected out of the controller
Interaction Type	Selection	When set to manipulation Fishing reel can move objects up and down the reel via parenting. When pure selection it will just send an event that it has selected and place the object into a public variable.
Controller Picked	Right_Controller	The controller this technique works with
Selected Object()	Empty	Invoked when object is selected
Dropped Object()	Empty	Invoked when object is Dropped
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over
Last Selected Object	None	The last object selected by the fishing reel

Reel speed	40	Speed objects will scroll up and down the cast fishing reel

7.10 Flashlight

7.10.1 Design Requirements

- Visually represented by a transparent cone projecting out of the controller like a real flashlight.
- Cone colour can be changed and modified by developer.
- Objects encompassed by cone are candidates for selection.
- If multiple objects are within the cone the one closest to the center is selected.
  - Objects closest to the center of the cone are calculated by getting the distance between each object and the vector projecting out of the controller. The object with the shortest distance is selected as the closest.
  - If two objects are approximately the same distance within a set parameter the object cleset the the controller is selected.

7.10.2 Unity Inspector parameters

Flashlight Script (Controls the Cone volume):

Parameter	Default	Description
Transparency	0.5	The level of transparency the cone volume is
The Colour	Black	The colour of the cone volume

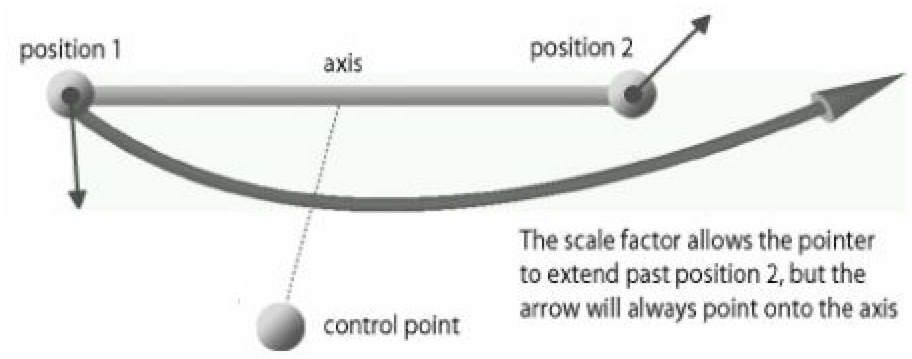
Flashlight Selection script:

Parameter	Default	Description

Interaction Layers	PickableObject	The layers that the technique can interact with
Interaction Type	Manipulation	When set to manipulation Flashlight can move objects via fixed joints. When pure selection it will just send an event that it has selected and place the object into a public variable.
Selection	None	The currently selected object
Object Hovered Over	None	The object currently hovered over
Selected Object()	Empty	Event Invoked when an object is selected
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.11 Flexible Pointer

7.11.1 Design Requirements


position 1
axis
position 2
control point
The scale factor allows the pointer to extend past position 2, but the arrow will always point onto the axis

- Requires both controllers to be used.
- Flexible pointer is calculated via a quadratic bezier curve.
    - Position 1 of the curve is represented by the right or dominate controller of the user.
    - Position 2 of the curve is represented by the other controller.
    - User will hold the non dominant controller out in front of them to use the flexible pointer.
- The control point of the cursor is manipulated via the touchpad (or the analog sticks for the Oculus Rift) of each remote.
    - The touchpad of the controller in position two controls the y axis of the control point in 3D space.
    - The touchpad of the controller in position one controls the x, and z axis of the control point in 3D space.
    - The above parameters control the local location of the cube relative to the vector projected between the two controllers so that it follows the user.
- A scaling factor should be customizable in the unity inspector to extend the reach of the pointer a certain percentage past the second controller position. By default this is set to 2.

7.11.2 Variables and Methods

For Flexible Pointer Script:

Parameter	Default	Description
Interaction Layers	PickableObject	The layers that the technique can interact with
Interaction Type	Selection	When selected an object will go into a public variable for the player to use. Currently nothing happens on manipulation.
Control Point Visible	True	If true a red sphere will indicate the location of the control point that is used to bend the bezier
Laser Container	Laser	Container for all the lasers generated for the Bezier Curve
Scale Factor	2	The scale of which the Flexible pointer extends past the pointing controller

Currently Pointing at	None	The currently hovered over object
Selection	None	The currently selected object
Selected Object()	Empty	Event Invoked when an object is selected
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.12 HOMER

7.12.1 Design Requirements

- HOMER is a Hybrid manipulation technique which utilizes ray-casting to initially determine which object a user wants to select.
- HOMER then uses a linear-scaling equation to track the distance between the users physical and virtual hand. This provides for an isomorphic mapping between the users physical and virtual hand.
- Once initially selecting the object, the user can extend their physical hand to manipulate the position and distance of a selected object.

The equation used to calculate the algorithm used to calculate this is the following:

On initial selection

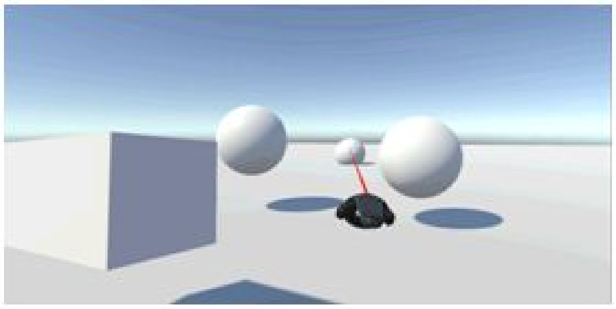
- Disth = Distance between the users physical hand and head at initial selection
- Disto = Distance between the selected object and head at initial selection

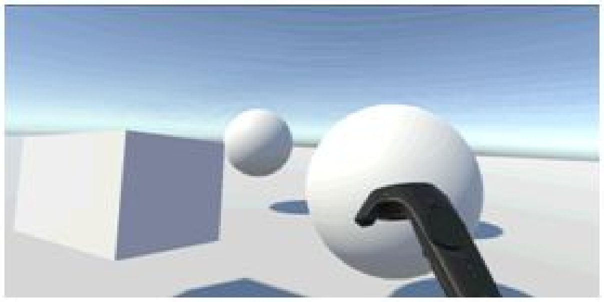
Calculated per frame

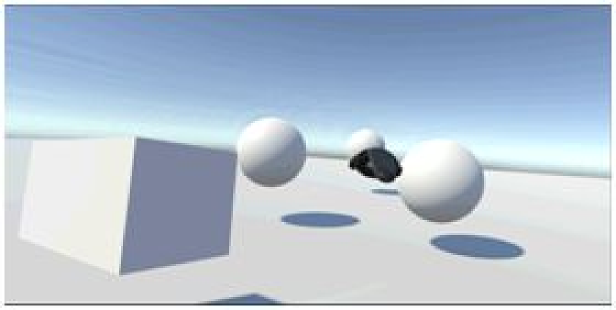
- Disthcur = distance between the users physical hand, and head.
- Distvh = Disthcur * (Disto / Disth)
- TorsoPos = physical hand position - head position

- Virtual hand position = head position + distvh * TorsoPos

The object is then set as a child to the virtual hand.







7.12.2 Unity Inspector parameters

For HOMER Script:

Parameter	Default	Description
Laser Prefab	Laser	The laser projected out of the controller
Intractable Layer	PickableObject	The layers that the technique can interact with
Controller Picked	Right_Controller	The controller that the technique uses
Hand Prefab	Hand Prefab	Hand that attaches to a selected object
Selected Object	None	The currently selected object

Selected Object()	Empty	Invoked when object is selected
Dropped Object()	Empty	Invoked when object is Dropped
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.13 Hook

7.13.1 Design Requirements

- The hook checks all objects in the area and each frame will detect the distance between the controller and all selectable objects.
- Selectable objects will be scored based on time spent closest to the controller as well as distance away and the highest score is selected. This will be collected in an array.
- The hook should be able to account for newly spawned objects in the scene.
  - The new objects will be added to the scoring array and hook will account for them.
- On destruction of objects hook must be able to clean up and remove objects from the array of scored objects or it could break the technique.
- If in manipulation mode when the user attempts to select an object via a trigger or button press the object with highest score will be moved into their hand and attached by setting its position to hand position.

7.13.2 Unity Inspector parameters

For Hook Script:

Parameter	**Default**	**Description**
Interaction Layers	PickableObject	The layers that the technique can interact with

Interaction Type	Manipulation	When set to manipulation object will jump into hook hand attached via a fixed joint, selection will add the object to an public Variable
Selection	None	The publicly accessible variable showing which object is selected.
Check For Newly Spawned	True	When enabled any new objects spawned into the scene will be added to the hooks tracking list
Currently Hovered	None	Object that will be selected by hook if trigger pressed
Selected Object()	Empty	Event Invoked when an object is selected
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.14 Image-Plane Pointing

7.14.1 Design Requirements

- Two different variations: Sticky Hand, and Framing hands.

Sticky Hand

- Selection is based off of a vector projected from the users camera viewpoint through the controller.
- To allow some leeway for error if an object is not hit by the a raycast through the vector, an algorithm is used to determine the distance between all nearby objects to the vector and itself. The object with the closest distance will then be selected.
- This give the feeling of the user looking at an object an “plucking it out of the air”

Framing Hand

- Uses the same algorithm and method as the sticky hand however instead of the vector passing from the head to the controller two controllers are used.
    - The midway point between the controllers is to be calculated and represented via a small red dot.
    - The Vector will be calculated starting from the headset and through the red dot.

7.14.2 Unity Inspector parameters

For Stickyhand ImagePlane_StickyHand Script:

Parameter	Default	Description
Interaction Layers	PickableObject	The layers that the technique can interact with
Selected Object	None	The last selected object by stickyhand
Currently Pointing At	None	The object hovered by the stickyhand
Laser Prefab	InvisLaser	Laser showing where stickyhand is aiming
SelectedObject	None	Currently selected object
Selected Object()	Empty	Invoked when object is selected
Dropped Object()	Empty	Invoked when object is Dropped
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

For Stickyhand ImagePlane_FramingHands Script:

Parameter	Default	Description

Interaction Layers	PickableObject	The layers that the technique can interact with
Currently Pointing At	None	The object hovered by the framing hands
Selected Object()	Empty	Invoked when object is selected
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over
Point of Interaction	InteractionPoint	Shows a dot in between the hands that i the interaction point
Laser Prefab	InvisLaser	Laser showing where framinghand is aiming
Selected Object	None	The last selected object by framinghand

7.15 iSith

7.15.1 Design Requirements

- Requires both controllers to be used.
- Visually a ray-cast laser will be projected from each users controller.
- A set interaction point is used for selection and manipulation. Objects can be attached to this point by the user pressing on the set controller button.
- The location of the interaction point is calculated each frame by the skew vector algorithm. The two vector inputs for the algorithm are projected out of the left and right controllers of the user. Where they cross over is the output of this algorithm.

7.15.2 Unity Inspector parameters

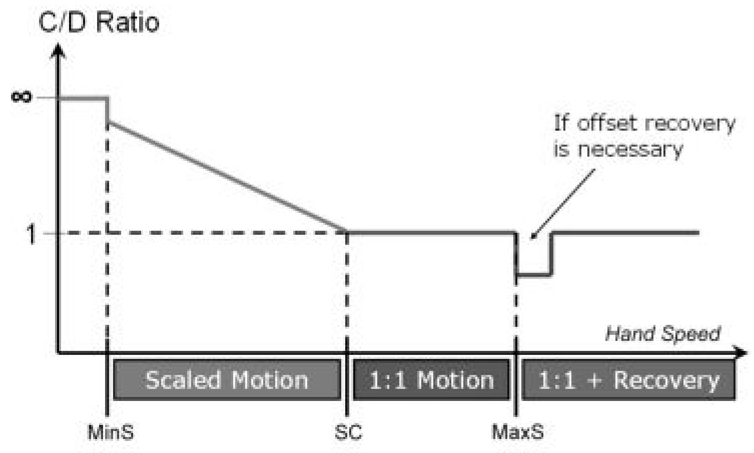
For ISithGrabObject Script:

Parameter	Default	Description

Interaction Layers	PickableObject	The layers that the technique can interact with
Colliding Object	None	Currently Highlighting object
Object In Hand	None	Currently Selected Object
Selected Object()	Empty	Event Invoked when an object is selected
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.16 PRISM (Movement)

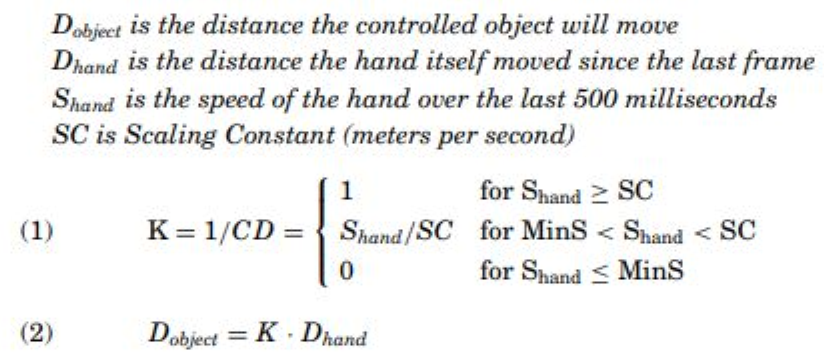
7.16.1 Design Requirements



- This technique is always active on the controller it is attached to.
- The technique will run its algorithms once an object is selected.

- On selection of an object its position is determined by the speed of the controller.
    - If a user is moving the controller faster than a set speed the object will 1-1 keep up with the hand as if it is moving normally.
    - If the user slows down the speed of the object will slow down further on a set mapping.
    - These exact scales are determined by variables MinS, SC, and MaxS shown in the diagram above.
    - These variables will be checked and compared to each other every 500 milliseconds.
        - Achieved in unity by tracking time passed in the scripts every frame and once it reaches 500 milliseconds the algorithm will run.

Below is the exact algorithm from the research paper to base this off of:



D_{object} is the distance the controlled object will move
D_{hand} is the distance the hand itself moved since the last frame
S_{hand} is the speed of the hand over the last 500 milliseconds
SC is Scaling Constant (meters per second)

$$(1) \qquad K = 1/CD = \begin{cases} 1 & \text{for } S_{\text{hand}} \geq \text{SC} \\ S_{hand}/SC & \text{for MinS} < S_{\text{hand}} < \text{SC} \\ 0 & \text{for } S_{\text{hand}} \leq \text{MinS} \end{cases}$$

$$(2) \qquad D_{object} = K \cdot D_{hand}$$

- An offset recovery algorithm is utilized to smooth out the transition from scaled speed back to 1 to 1 speed of the object when the user speeds up their arm. Below is this algorithm from the original paper:

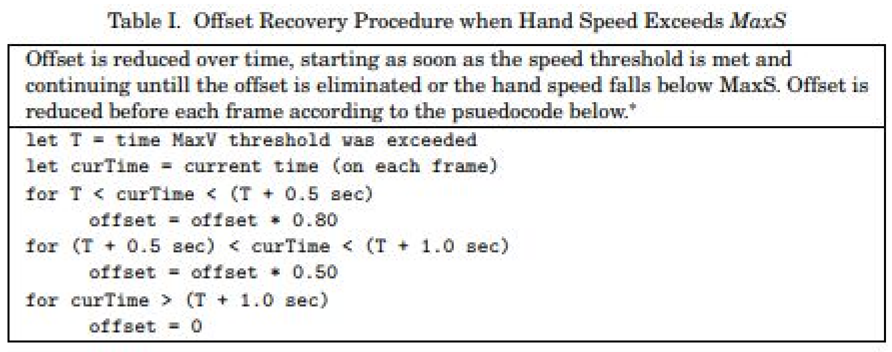


Table I. Offset Recovery Procedure when Hand Speed Exceeds *MaxS*

Offset is reduced over time, starting as soon as the speed threshold is met and continuing untill the offset is eliminated or the hand speed falls below MaxS. Offset is reduced before each frame according to the psuedocode below.*

```
let T = time MaxV threshold was exceeded
let curTime = current time (on each frame)
for T < curTime < (T + 0.5 sec)
      offset = offset * 0.80
for (T + 0.5 sec) < curTime < (T + 1.0 sec)
      offset = offset * 0.50
for curTime > (T + 1.0 sec)
      offset = 0
```

7.16.2 Unity Inspector parameters

For PRISM Script:

Parameter	Default	Description

Interaction Layers	PickableObject	The layers that the technique can interact with
Colliding Object	None	Object that PRISM hand is hovered over
Object In Hand	None	Object that PRISM hand is currently holding
Min S	0.001	Minimum velocity before object wont move
Scaled Constant	0.5	Minimum velocity before object enters scaled mode
MaxS	2	Velocity at which PRISM recovers distance from scaling
Selected Object()	Empty	Event Invoked when an object is selected
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.17 Scaled-World Grab

7.17.1 Design Requirements

- Scaled-world grab is a manipulation technique that initially uses some selection technique. (In our implementation we've use ray-casting.)
- Once the user selects an object, the entire user scales up without affecting the user's original perspective. The user will now be able to physically reach out and grab the object with their controller to manipulate it.

In order to achieve this the following equation is used:

Calculating the world-scale amount

- disteh = The distance between the user's head and hand.
- disteo = The distance between the users head and the selected object.
- The scale is then set to (=) disteo / disteh

Scaling the camera rig whilst maintaining the same perspective

- pivotPar = the pivot of the virtual head parent

- pivotPosition = The position of the user’s head
- pivotParent = The parent of the virtual head is then set to the camera rig
- The camera rig is then set to the scale which we calculated above.
- The position of the camera rig is = to itself + (pivotPosition - head position)
- pivotParent (parent of the virtual head) is then returned back to its original parent.

We then divide the eye proportion perspective of the current scale of the head / the scale calculated above.

7.17.2 Unity Inspector parameters

For Scaled World Grab Script:

Parameter	Default	Description
Intractable Layer	PickableObject	The layers that the technique can interact with
Interaction Type	Selection	When set to manipulation Scaled world grab can move objects via parenting. When pure selection it will just send an event that it has selected and place the object into a public variable.
Controller Picked	Right_Controller	The controller the technique uses
Selected Object	None	Object selected by technique

For Controller Collider (For selecting objects) Script:

Parameter	Default	Description
Scaled World Grab	None	The Scaled world grab script, this will be set up on run time.
Selected Object()	Empty	Invoked when object is selected
Dropped Object()	Empty	Invoked when object is Dropped

Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over
Scale Selected	None	Object selected in scale mode

7.18 Serial Selection Mode

7.18.1 Design Requirements

- Example method of selection showing this technique in use is a simple ray cast projected from the controller.
- The core code of this serial selection will contain a dynamic array.
    - Will use a C# List.
- On object selection an object will be added to the list.
- The list must be public for a developer to easily access and do with the list as the please.
- On an invalid selection, the list will be emptied. This will be a way for the user to reset the list.
    - Invalid selection is when the user presses on the selection button (Such as the trigger) and nothing is selected.

7.18.2 Unity Inspector parameters

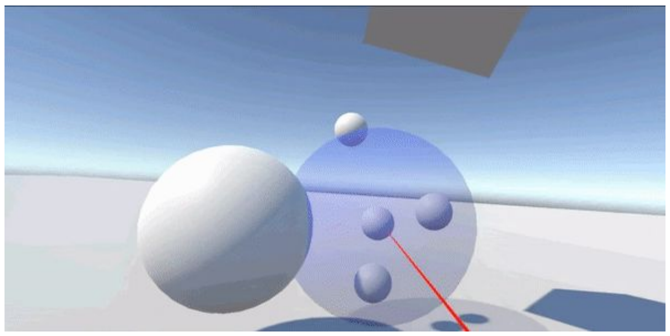
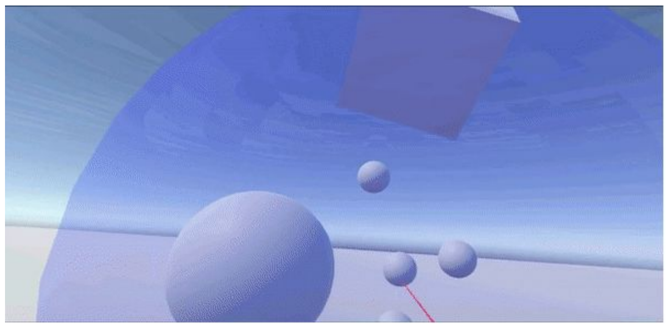
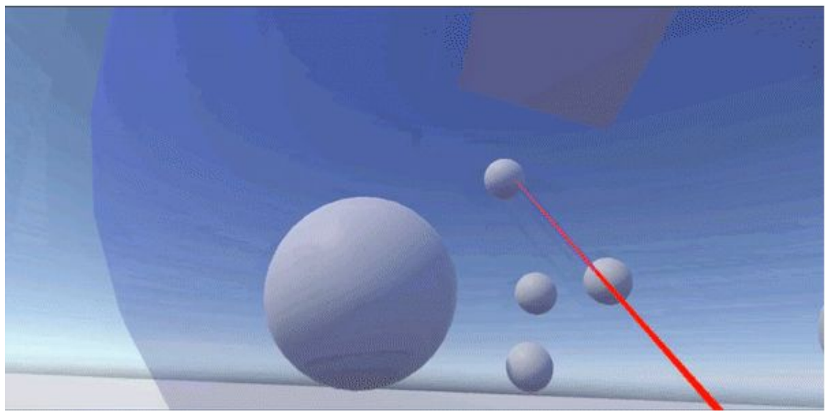
For Serial Selection Script:

Parameter	Default	Description
Intractable Layer	PickableObject	The layers that the technique can interact with
Laser Prefab	Laser	Laser projecting from controller
Controller Picked	Right_Controller	The controller the technique uses
Selected Object()	Empty	Invoked when object is selected

7.19 Sphere-Casting

7.19.1 Design Requirements

- Sphere-casting is a volume based multi selection technique which uses a virtual ray casting out of the controller. (Much like ray-casting)
- A semi-transparent sphere is then attached to the end of the casted ray appearing whenever the ray hits an object.
- The user can change the radius of the semi-transparent sphere with the HTC Vive touchpad or Oculus Rift joystick
- When the user selects, all objects within the semi-transparent sphere are then selected.

7.19.2 Unity Inspector parameters

For Sphere Casting Script:

Parameter	Default	Description

Laser Prefab	Laser	The laser projected from the remtoe
Interaction Type	Selection	When set to manipulation sphere casting can move all objects within it via parenting.
Controller Picked	Right_Controller	The controller this technique uses

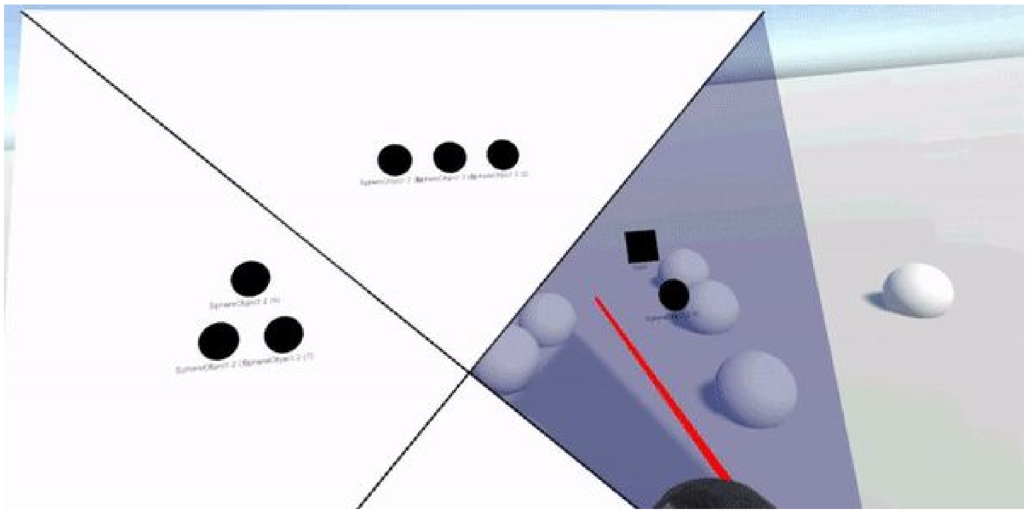
7.20 SQUAD

7.20.1 Design Requirement

- SQUAD is an extension of the sphere-casting technique and can be enabled within the prefab settings.
- Unlike sphere-casting, when multiple objects are selected in the semi-transparent sphere a 2D QUAD menu appears with 2D clones of the original object.
- The 2D cloned objects are scaled based on the original scale of the object / 10.

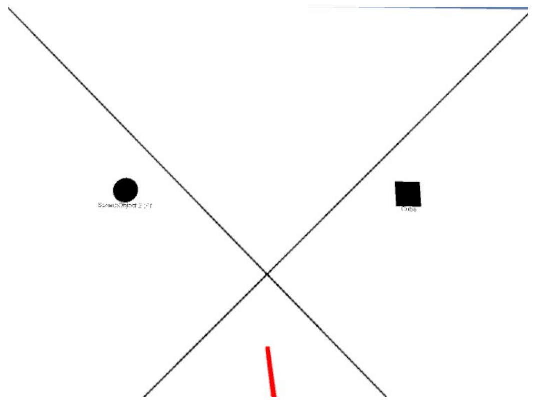
QUAD MENU

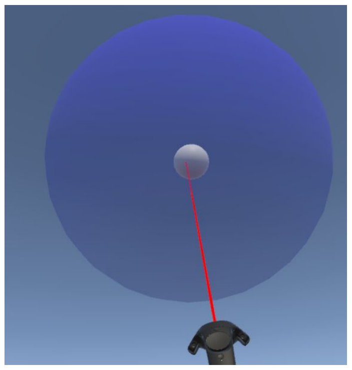
- Once the QUAD menu is enabled, 2D objects are equally distributed into each quadrant.



- The user can then use virtual ray-casting as a selection tool to navigate through the manu.
- A blue outline is displayed when the virtual ray hovers over each of the QUADs.

- Each time a user selects a quad with more than 1 object, the quad progressively refines and re-distributes each 2D object into a new quadrant.
- Eventually, only 1 object will remain inside a quadrant, and the user can select the quadrant and the object will become selected.





7.20.2 Unity Inspector parameters

Uses the same Inspector parameters for selection as Sphere casting here. This is because sphere casting is the selection method for squad except with a public boolean "Squad enabled" to turn it on.

For SQUAD Menu:

Parameter	Default	Description
Prefab Text	PrefabText	Text for object words
QuadrantMaterial	transparent	Material shown when selecting a quadrant
Outline Material	ItemPickupOutline	Outlines an object with this on selection
Triangle Material	TriangleMaterial	Material for quadrant colours

7.21 EXPAND

7.21.1 Design Requirements

- Similar to the QUAD technique, expand also utilizes sphere-casting as a selection technique.
- When multiple objects are selected using sphere-casting, a 3D interface of all the objects are aligned directly in front of the user consisting of 5 objects per row.
- Once the 3D interface is enabled the semi-transparent sphere freezes in place and ray-casting is used to to select the 3D miniature clone object in the menu.
- Once selecting the 3D clone, the physical object will become selected.

7.21.2 Unity Inspector parameters

For Sphere Casting Exp Script:

Parameter	Default	Description
Laser Prefab	Laser	Laser projected out of controller
Interaction Type	Selection	When set to manipulation Sphere casting Exp can move objects via parenting. When pure selection it will just send an event that it has selected and place the object into a public variable.
Controller Picked	Right_Controller	The controller this technique uses

For EXPAND Menu:

Parameter	Default	Description
Panel	menuPanel	The expand menu
Selected Material	ItemPickupOutline	Outlines an object with this on selection

7.22 Spindle

7.22.1 Design Requirements

- Each frame a script will detect the midpoint between the two controllers. This midpoint is the interaction point
- Interaction point is represented by a small transparent cube. Its position will be updated to that calculated position each frame.
- Object can be picked up by colliding the interaction cube with a real object and using the selection (trigger) button.
- The yaw and roll of the object are determined by the location of the hands relative to each other while holding with the spindle.
- The scale of the object is dynamically based on the calculated distance between the controllers each frame. To do this a ratio will be kept from when the object is first picked up. The distance between the two controllers on pickup are mapped to the current scale of the object as 1. So when they get further apart it will grow, closer together shrink, and when the distance originally picked up from it will be the original size.

7.22.2 Unity Inspector parameters

For Spindle Script:

Parameter	Default	Description
Spindle and Wheel	False	When enabled the Spindle and Wheel technique will activate allowing rotation on the pitch axis. More information here.
InteractionObject	InteractionPoint	The point of interaction for spindle

For Spindle Interactor Script:

Parameter	Default	Description
Interaction Layers	PickableObject	The layers that the technique can interact with
Colliding Object	None	The currently Hovered over object
Object In Hand	None	Current object selected
Selected Object()	Empty	Invoked when object is selected
Dropped Object()	Empty	Invoked when object is Dropped
Hovered()	Empty	Event Invoked when an object is hovered over
UnHovered()	Empty	Event Invoked when an object ceases being hovered over

7.23 Spindle + Wheel

7.23.1 Design Requirements

- Spindle and wheel design requirements contain all spindle design requirements.
- In the unity implementation there will be just a spindle prefab with a boolean option to enable the wheel.
- Rotating the primary set controller will affect the pitch of the currently selected object allowing the spindle to have all 7 degrees of freedom. Manipulation/rotation on all axis, and scale.

7.23.2 Unity Inspector parameters

Exactly the same as spindle parameters. See 7.22 Spindle

7.24 World In Miniature

7.24.1 Design Requirements

- World-in-Miniature is an indirect selection and manipulation technique that allows users to scale the world down into their hands.
- The way it works is simply the user presses a button on their controller (Application_Menu for the HTC Vive) and then a virtual miniature representation of the world appears on top of their controller.
- The virtual world is then set as a parent to the controller, so the user can easily move the rotation or position of the virtual miniature world.
- Using the opposite hand, the user can physically manipulate the miniature cloned objects, and it will correctly transform to the real-world object.
- In our implementation, rigidbodies and physics also correctly transform over to the scaled down virtual world, allowing for full functionality between the scaled down and real world.
- The default scale between the physical and miniature world is a 1:20 scale. However, this variable can be customized in the prefab settings to increase or decrease scale.
- Users also have the ability to rotate the scene around their hand using the HTC Vive touchpad or Oculus Rift joystick.

7.24.2 Unity Inspector parameters

For World In Miniature Script:

Parameter	Default	Description
WimActive	false	Shows true when the world in miniature is active
Ignorable Objects String Array	[CameraRig], Directional Light, background	A list of strings of names of specific types of objects not to be included by world in miniature
Scale Amount	20	The scale setting for the size of the miniaturized world
Interactable Layers	PickableObject	The layers that the technique can interact with

Interaction Type	Manipulation_Movement	When set to manipulation Wim can move objects via parenting. When pure selection it will just send an event that it has selected and place the object into a public variable.
ControllerPicked	Right_Controller	The controller that the world in miniature will appear on
Selected object	None	The currently Selected object
Current Object Collided	None	The currently Hovered over object
Tilt speed	2	The speed at which you can rotate Wim via the touchpad

Appendix A: Declaration of Contribution

The following is a declaration of your individual contributions towards this group assessment. If any contribution does not meet the assessment requirements, the course coordinator may adjust individual marks up or down, depending on the level of contribution made.

Team Member 1

Name: Kieran May

I contributed approximately 9000 words **towards this assessment** + diagrams**.**

I worked on the following sections/questions (select whichever is appropriate).

(We worked on all sections together)

Team Member 2

Name: Ian Hanan

I contributed approximately 9000 words **towards this assessment** + diagrams**.**

I worked on the following sections/questions (select whichever is appropriate).

(We worked on all sections together)