VRBOMB: VR-BASED AERIAL BOMB ASSEMBLY SIMULATOR

**วีอาร์บอมบ์: ระบบจำลองการถอดประกอบวัตถุระเบิดชนิดทิ้งทุ่น**

BY

Miss patboke jitthamma 5488049

mr. pramoch viriyathomrongul 5488054

mr. TREECHART TREERATTANAPAKONG 5488207

ADVISOR

DR. MORES PRACHYABRUED

A Senior Project Submitted in Partial Fulfillment of

The Requirements for

THE DEGREE OF BACHELOR OF SCIENCE

(INFORMATION AND COMMUNICATION TECHNOLOGY)

Faculty of Information and Communication Technology

Mahidol University

2015

ACKNOWLEDGEMENTS

There are many people to whom we owe our sincere appreciation for their assistance support. First of all, we would like to express our sincere thanks to senior project advisor, Dr. Mores Prachyabrued for the valuable help, presentation on the topic, and coordination with organizations, recommendations and constantly encouragement throughout this project. We would not have achieved this far and the project would not have been completed without all the support that we have always received from him.

Our special thanks to The Defense Technology Institute (DTI) under the Ministry of Defense for the support of the information, and invitation to the ACDT-2015 Event at Hyatt Regency Hua Hin. We would not have succeeded if we would not get this support from them either.

We also wish to thank our friends, ICT faculty instructors and staff for their additional advice and support. Finally, the most important of all, we would like to express our gratitude to our families for being so supportive of our effort and always give strengthens. We would like to dedicate this senior project to them.

Miss Patboke Jitthamma

Mr. Pramoch Viriyathomrongul

Mr. Treechart Treerattanapakong

**VRbOMB: VR-BASED AERIAL BOMB ASSEMBLY SIMULATOR**

patboke jitthamma 5488049 ITCS/B

pramoch viriyathomrongul 5488054 ITCS/B

TREECHART TREERATTANAPAKONG 5488207 ITCS/B

B.Sc. (INFORMATION AND COMMUNICATION TECHNOLOGY)

PROJECT ADVISOR: DR. MORES PRACHYABRUED

ABSTRACT

Some military procedures are highly dangerous such as bomb assembly and defusing. It is critical that military personnel receive adequate training. Practice in a real environment provides realistic experience but the cost involved can be high as well as the risk. Virtual Reality (VR) has huge potential in training simulation because it provides safe training environment and the cost is generally less than the cost of training with real equipment. Virtual training can be used to increase skill before training in a real environment. In this project, we develop a VR simulator for aerial bomb assembly training (VRBomb). The system provides realistic 3D graphics and supports natural hand interaction. The system will be evaluated with human subjects. We expect that our developed method can be applied to training of other manual assembly tasks.

This project is in collaboration with the Defense Technology Institute (DTI) under the Ministry of Defense, in terms of information and evaluation in ACDT-2015 events at Hyatt Regency Hua Hin.

KEYWORDS: virtual reality / military simulation / air-dropped bomb / OCULUS RIFT2 / LEAP MOTION / MARK 82 / MARK 84

113 P.

วีอาร์บอมบ์: ระบบจำลองการถอดประกอบวัตถุระเบิดชนิดทิ้งทุ่น

พัดโบก จิตธรรมมา 5488049 ITCS/B

ปราโมช วิริยะธำรงกุล 5488054 ITCS/B

ตรีชาติ ตรีรัตนประคอง 5488207 ITCS/B

วท.บ. (เทคโนโลยีสารสนเทศและการสื่อสาร)

อาจารย์ที่ปรึกษาโครงการ: ดร. โมเรศ ปรัชญพฤทธิ์

บทคัดย่อ

ในปัจจุบัน บางขั้นตอนของระบบการทหารจะมีความอันตรายสูง เช่นการประกอบระเบิด หรือการกู้ระเบิด ซึ่งเป็นสิ่งสำคัญที่บุคลากรทางการทหารต้องได้รับการฝึกอบรมที่เพียงพอ ถึงการปฏิบัติงานในสภาพแวดล้อมจริงจะช่วยส่งเสริมให้ได้รับประสบการณ์ที่สมจริง แต่ค่าใช้จ่ายที่เกี่ยวข้องกับการปฎิบัตินี้ก็สูงมากขึ้นเช่นเดียวกันกับความเสี่ยงอันตรายในการปฎิบัติงาน ดังนั้น “ระบบจำลองแบบเสมือนจริง” จึงเข้ามามีบทบาทและสามารถสร้างศักยภาพมากในการจำลองการฝึกอบรมทางการทหารนี้ เพราะสามารถจำลองสภาพแวดล้อมการฝึกอบรมให้เสมือนจริงแต่มีความปลอดภัยและค่าใช้จ่ายที่น้อยกว่าเทียบกับการฝึกอบรมในสถาณที่จริงและใช้อุปกรณ์จริง ระบบจำลองแบบเสมือนจริงนั้นสามารถนำมาใช้เพื่อเพิ่มทักษะก่อนการฝึกอบรมในสภาพแวดล้อมจริง

ในโครงการนี้ทางคณะผู้จัดทำได้ทำการพัฒนาและจำลอง VR สำหรับการฝึกอบรมการประกอบวัตถุระเบิดชนิดทิ้งทุ่น (VRBomb) ในระบบจำลองแบบเสมือนจริงประกอบไปปด้วยระบบกราฟิก 3 มิติที่สมจริงและสนับสนุนการมีปฏิสัมพันธ์กับเครื่องมือเฉพาะของ VR เช่น Leap Motion และ Oculus Rift ระบบจำลองการฝึกอบรมการประกอบวัตถุระเบิดชนิดทิ้งทุ่น จะมีการประเมินผลการทดลองกับทหารฝึกหัด หรือผู้ที่สนใจอีกด้วย ซึ่งเราคาดหวังว่าการพัฒนาของเรานั้นสามารถนำไปใช้ในการฝึกอบรมของงานทางการทหารเกี่ยวกับการประกอบวัตถุอื่นๆได้อีกด้วย

อีกทั้งโครงการนี้ได้ความร่วมมือกับสถาบันเทคโนโลยีป้องกันประเทศ (DTI) ภายใต้กระทรวงกลาโหมด้านการจัดหาข้อมูลระเบิดและการประเมิลผลการทดลองในงาน ACDT-2015 ณ โรงแรม ไฮแอท รีเจนซี่ หัวหิน

113 หน้า

CONTENTS

Page

[ACKNOWLEDGEMENTS ii](#_Toc421576617)

[ABSTRACT iii](#_Toc421576618)

[บทคัดย่อ iv](#_Toc421576619)

[CONTENTS v](#_Toc421576620)

[LIST OF TABLES viii](#_Toc421576621)

[LIST OF FIGURES ix](#_Toc421576622)

1. [Introduction 1](#_Toc421576684)

[1.1 Motivation 1](#_Toc421576685)

[1.2 Problem Statement 6](#_Toc421576686)

[1.3 Objectives of the Project 6](#_Toc421576687)

[1.4 Scope of the Project 6](#_Toc421576688)

[1.5 Expected Benefits 7](#_Toc421576689)

[1.6 Organization of the Document 7](#_Toc421576690)

1. [Background 8](#_Toc421576691)

[2.1 Literature Review 8](#_Toc421576692)

[2.2 Mark82 Bomb (MK-82) 9](#_Toc421576693)

[2.3 Virtual Reality (VR) 13](#_Toc421576694)

[2.4 Oculus Rift DK2 15](#_Toc421576695)

[2.5 Leap Motion 18](#_Toc421576696)

[2.6 Related work 20](#_Toc421576697)

[2.7 Contribution 21](#_Toc421576698)

1. [Analysis and Design 23](#_Toc421576699)

[3.1 System Architecture Overview 23](#_Toc421576700)

[3.2 System Structure Chart 24](#_Toc421576701)

[3.3 Content Design 28](#_Toc421576702)

[3.3.1 MK-83 bomb assembly information 29](#_Toc421576703)

[3.3.2 Leap Motion API 33](#_Toc421576704)

[3.3.3 Oculus Rift DK2 Prefabs 39](#_Toc421576705)

[3.4 Simulator Design 43](#_Toc421576706)

[3.4.1 Gameplay 43](#_Toc421576707)

[3.4.2 Conditions 43](#_Toc421576708)

[3.4.3 Scenes 43](#_Toc421576709)

[3.4.4 Interfaces 48](#_Toc421576710)

[3.4.5 Effects 49](#_Toc421576711)

[3.4.6 Hand Graphic Model 50](#_Toc421576712)

[3.4.7 Models 52](#_Toc421576713)

1. [Implementation 56](#_Toc421576714)

[4.1 Hardware and System Environment 56](#_Toc421576715)

[4.2 Implementation Guide and Techniques 56](#_Toc421576716)

[4.2.1 Setting up the Environment 57](#_Toc421576717)

[4.2.2 Oculus integration in Unity 63](#_Toc421576718)

[4.2.3 Leap Motion integration in Unity 64](#_Toc421576719)

[4.2.4 How to interact (grab) object 67](#_Toc421576720)

[4.2.5 Handle the object out of scene 79](#_Toc421576721)

[4.2.6 Script Flow 80](#_Toc421576722)

[4.2.7 How to detect validity assembly 82](#_Toc421576723)

[4.2.8 Perform Action after Detected Validity 83](#_Toc421576724)

[4.2.9 Check Hand Gestures 89](#_Toc421576725)

[4.2.10 Check Keyboard pressed 91](#_Toc421576726)

[4.2.11 Check Hand in scene 93](#_Toc421576727)

[4.2.12 Control highlight effect 93](#_Toc421576728)

[4.2.13 Highlight held object 95](#_Toc421576729)

[4.2.14 Change graphic hand model 98](#_Toc421576730)

1. [Testing and Evaluation 100](#_Toc421576731)

[5.1 Testing Methods 100](#_Toc421576732)

[5.1.1 Development Testing 100](#_Toc421576733)

[5.1.2 Users Testing 101](#_Toc421576734)

[5.2 Evaluation Results 103](#_Toc421576735)

1. [Conclusions 110](#_Toc421576736)

[6.1 Benefits 110](#_Toc421576737)

[6.1.1 Benefits to Project Developers 110](#_Toc421576738)

[6.1.2 Benefits to Users 110](#_Toc421576739)

[6.2 Problems and Limitations 110](#_Toc421576740)

[6.3 Future Work 111](#_Toc421576741)

[REFERENCES 112](#_Toc421576742)

[BIOGRAPHIES 113](#_Toc421576743)

LIST OF TABLES

Page

[Table 2‑1: Specifications of MK-82 bomb 10](#_Toc421576744)

[Table 2‑2: Example of VR devices in present (VR GEARS) 14](#_Toc421576745)

[Table 2‑3: Specifications of Oculus Rift DK2 17](#_Toc421576746)

[Table 2‑4: Specifications of Leap Motion 19](#_Toc421576747)

[Table 3‑1: The Leap Motion API measures physical quantities 34](#_Toc421576748)

[Table 3‑2: OVRManager.cs contains 41](#_Toc421576749)

[Table 3‑3: All scenes in VRBomb Simulator 44](#_Toc421576750)

[Table 3‑4: All interfaces in VRBomb Simulator 48](#_Toc421576751)

[Table 3‑5: All effects in VRBomb Simulator 49](#_Toc421576752)

[Table 3‑6: All equipment and other model in VRBomb Simulator 50](#_Toc421576753)

[Table 3‑7: All MK-82 parts model which it used in VRBomb Simulator 52](#_Toc421576754)

LIST OF FIGURES

Page

[Figure 1‑1: Screenshot of the Dismounted Solider Training System 4](#_Toc421576818)

[Figure 1‑2: Screenshot of the Dismounted Solider Training System 5](#_Toc421576819)

[Figure 2‑1: All MK-80 series Bomb 11](#_Toc421576820)

[Figure 2‑2: The main elements of MK-82 bomb 11](#_Toc421576821)

[Figure 2‑3: Screenshot of real MK-82 Bomb 12](#_Toc421576822)

[Figure 2‑4: Screenshot of real MK-82 Bomb 12](#_Toc421576823)

[Figure 2‑5: All views of Oculus Rift DK2 (front, side, back, all tools) 16](#_Toc421576824)

[Figure 2‑6: Screenshot of Oculus Rift DK2 POV 16](#_Toc421576825)

[Figure 2‑7: All views of Leap Motion (Front, side, back) 18](#_Toc421576826)

[Figure 2‑8: Screenshot of skeletons tracking 19](#_Toc421576827)

[Figure 2‑9: Screenshot of Planetarium 20](#_Toc421576828)

[Figure 2‑10: Screenshot of Rift Coaster 21](#_Toc421576829)

[Figure 3‑1: System Architecture Overview of VRBomb 23](#_Toc421576830)

[Figure 3‑2: System Structure chart of VRBomb 25](#_Toc421576831)

[Figure 3‑3: Nose Fuze M940 details 30](#_Toc421576832)

[Figure 3‑4: GBU-24B/B details 32](#_Toc421576833)

[Figure 3‑5: The Leap Motion controller’s view of your hands 33](#_Toc421576834)

[Figure 3‑6: The Leap Motion right-handed coordinate system. 34](#_Toc421576835)

[Figure 3‑7: Motion Tracking Data structure chart 35](#_Toc421576836)

[Figure 3‑8: The Hand vectors define the orientation of the hand. 36](#_Toc421576837)

[Figure 3‑9: Fingertip vectors provide the position of a fingertip and the general direction in which a finger is pointing. 37](#_Toc421576838)

[Figure 3‑10: All fingers contain four bones ordered from base to tip 37](#_Toc421576839)

[Figure 3‑11: Screen Tap Gesture 38](#_Toc421576840)

[Figure 3‑12: OVRCameraRig and OVRPlayerController 39](#_Toc421576841)

[Figure 3‑13: OVRCameraRig, expanded in the inspector 40](#_Toc421576842)

[Figure 4‑1: Interface Rift Display Mode 57](#_Toc421576843)

[Figure 4‑2: Interface Leap Motion Control Panel 58](#_Toc421576844)

[Figure 4‑3: Interface Leap Motion Control Panel setting 59](#_Toc421576845)

[Figure 4‑4: Unity 3D project create project 60](#_Toc421576846)

[Figure 4‑5: Unity 3D imported package to project 60](#_Toc421576847)

[Figure 4‑6: Unity 3D Game Interface 61](#_Toc421576848)

[Figure 4‑7: Unity 3D Assets menu 62](#_Toc421576849)

[Figure 4‑8: Unity asset Oculus Rift Prefabs 63](#_Toc421576850)

[Figure 4‑9 Components of OVRCameraRig 64](#_Toc421576851)

[Figure 4‑10: HandController prefab 64](#_Toc421576852)

[Figure 4‑11: Child of CenterEyeAnchor under OVRCameraRig 65](#_Toc421576853)

[Figure 4‑12: Component of HandController 65](#_Toc421576854)

[Figure 4‑13: Rigid Hand Model asset 66](#_Toc421576855)

[Figure 4‑14: Set up physical hand model 67](#_Toc421576856)

[Figure 4‑15: Parameter of InteractionSceneSetup 72](#_Toc421576857)

[Figure 4‑16: Rigid body component of “RigidHand” 73](file:///C:\Users\user\Desktop\VRBomb_Final_Document%20(06-09-15)%2012.45%20AM.docx#_Toc421576858)

[Figure 4‑17: Rigid Body Parameter 73](#_Toc421576859)

[Figure 4‑18: Leap Interaction Parameter 79](#_Toc421576860)

[Figure 4‑19: 6 Objects at different direction with Box Collider 80](#_Toc421576861)

[Figure 4‑20: Script Flow Chart 81](#_Toc421576862)

[Figure 4‑21: Set up trigger object 83](#_Toc421576863)

[Figure 4‑22: Assembly script 88](#_Toc421576864)

[Figure 4‑23: Inspector 90](#_Toc421576865)

[Figure 4‑24: Particle object panel 95](#_Toc421576866)

[Figure 4‑25: Holding Callbacks Script 97](#_Toc421576867)

[Figure 4‑26: All hand model script 99](#_Toc421576868)

[Figure 5‑1: Usability Testing at ACDT2015 event 101](#_Toc421576869)

[Figure 5‑2: Evaluation Form 102](#_Toc421576870)

[Figure 5‑3: Graph of Age 103](#_Toc421576871)

[Figure 5‑4: Graph of Gender 103](#_Toc421576872)

[Figure 5‑5: Graph of Occupation 104](#_Toc421576873)

[Figure 5‑6: Graph of Question 1 105](#_Toc421576874)

[Figure 5‑7: Graph of Question 2 105](#_Toc421576875)

[Figure 5‑8: Graph of Question 3 106](#_Toc421576876)

[Figure 5‑9: Graph of Question 4 107](#_Toc421576877)

[Figure 5‑10: Graph of Question 5 107](#_Toc421576878)

[Figure 5‑11: Graph of Question 6 108](#_Toc421576879)

[Figure 5‑12: Graph of Question 7 109](#_Toc421576880)

CHAPTER 1

# Introduction

This chapter is the introduction of VRBomb project. It includes Motivation, Problem Statement, Project Objective, Scope of the Project and Expected Benefits.

## Motivation

Our motivation is to simulate the specific environment that use to train the soldier trainee. We will focus on create the stable and safe training environment in virtual reality in order to let the trainee gain more accommodate with each training section before going into real training. The reason for created this project is Dangerous military procedures, good model of VR Simulator Training in US military and the problem of equipment shortage in Thailand Military  
  
 First motivation is to simulate the dangerous training into virtual reality. The reason is that because many of the military component that consist part and some chemical that need sensitive and careful method to handle and store them so it not cause the casualty due to component malfunction or personal error. Many part that was used in training also very sensitive and dangerous object so the trainee is needed to have very careful to prepare them with the correct method. If we can simulate them into the virtual reality, it can be able to reduce the casualty rate that may occur in the training.  
  
 Our second motivate is because we have seen the VR training simulation from United Stated military. United States is the country that very keen in the military technology so they very concern about the method that will make there trainee be able to train hard while the casualty is low. United State military develop virtual reality for all the services (Army, Navy and Air force). The training that they produce is focus on training the soldier in combat situation or dangerous setting that where they learn how to react base on the situation they are in, for example engagement with an enemy in an environment in which they experience this but without the real world risks. This has proven to be safer and less costly than traditional training methods.

**Military uses of virtual reality**

These include:

* Flight simulation
* Battlefield simulation
* Medic training (battlefield)
* Vehicle simulation
* Virtual boot camp

Virtual reality can also use for mental treatment in the veteran that suffer from traumatic stress disorder by simulate the situation in virtual reality to reproduce the specific situation to let the patient to learn how to deal with them. The concept of this treatment is to let patients to face with their trauma and gradually adjust their mind to deal with them. Another use is combat visualization in which soldiers and other related personnel are given virtual reality glasses to wear which create a 3D depth of illusion. The results of this can be shared amongst large numbers of personnel.

Find out more about individual uses of virtual reality by the different services, e.g. virtual reality navy training in the separate virtual reality and the military section.

This section discusses the various military applications of virtual reality and the ramifications from using this form of technology. The military may not be an obvious candidate for virtual reality but it has been adopted by all branches army, navy and air force.

This section discusses all aspects of how virtual reality is used by military, from training through to combat situations. It is arranged as follows:

* Virtual reality war
* Virtual reality and the Army
* Virtual reality and the Navy
* Virtual reality and the Air force
* Virtual reality army training
* Virtual reality army exercises
* Virtual reality air force training
* Virtual reality navy training
* Virtual reality combat training
* Virtual reality combat simulation
* Virtual reality military weapons
* Virtual reality military history

The example of VR training in US military, The Dismounted Soldier Training System is a new military sim commissioned by the US government. The sim will be used to train new soldiers as early as next year, and cost a hefty $57 million to make. GamePro report that it's likely that most of this went on development of the new technology that will capture soldiers' movements on a ten foot square pad. New recruits will experience the simulator through a virtual reality headset that will run from a state-of-the-art custom laptop strapped to their back.

The sim is powered by the Crysis 2 engine, and aims to provide a flexible training tool that should ultimately be cheaper than running similar live exercises. Directors running training exercises will be able to mod new missions using intelligence on a target area. This will allow soldiers to move through a mission in virtual reality before attempting it in real life. Once a mission is completed, soldiers receive a score and a report on their actions in the mission.

The sim is being developed by Intelligent Decisions is designed to be able to realistically model helicopters and ground vehicles, as well as environmental effects like footprints, disturbed foliage and realistic weather effects. Multiple soldiers will be able to play in the same instance, allowing troops to practice squad tactics.



Figure 1‑1: Screenshot of the Dismounted Solider Training System

The director of strategic programs at Intelligent Decisions, Floyd West tells GamePro that "a soldier could practice a scenario alone or with squad members, just as they would play an online video game. Soldiers and instructors could also play roles within the simulation. For example, they might play civilians that interact with someone from that squad, or they could play an opposing force, an enemy combatant. They could also control a group of semi-automated forces in the world."

"With Cry Engine 3 being used for Crysis 2 and the capabilities that game engine provides, it allows us to make the most realistic simulation possible. We're able to transport soldiers to accurately recreated locales like Afghanistan and Iraq, where we can simulate everything from visuals to 360-degree sound."



Figure 1‑2: Screenshot of the Dismounted Solider Training System

The virtual reality headsets the trainees wear will run from a backpack unit similar to a top of the range gaming laptop, called the 'Man Wearable Unit'. "While the man wearable units aren't running on an off-the-shelf Alien ware, the internal components themselves are commercial off-the-shelf CPU's and GPU's like NVIDIA graphic cards and whatnot," says west.

Our last motivate is because the military equipment in Thai military is often get shortage that lead to lack of training in the new soldier trainee and also the fund that military receive is not sufficient to afford the new equipment due to high price per unit. The lack of training can lead to casualty in real situation so we can solve this problem if we using the virtual reality because the virtual reality can be use many time and maintain cost is cheaper compare to the actual equipment.

## Problem Statement

Nowadays, in military training section, there are many training that may involve the trainer to participate in danger and the accident can occur if the soldier trainee don’t have enough experience in using or doing the process that may lead to become a fatal accident or losing unnecessary expense to repair the training device. In order to prevent the accident and reduce equipment costs, we are developing the virtual reality that will let the trainee to train how to prepare the MK82 general purpose bomb by using leap motion as the hand tracking device and using Oculus rift as the monitor to display the virtual environment also use Unity game engine as the base to combine Oculus rift and leap motion together.

## Objectives of the Project

The objectives of this project are:

* Develop a VR simulator and training to use the VR device.
* Know the structure and sequence of MK-82 Bomb assembly.
* Evaluate the developed system with human subjects
* Explore emerging VR technology

## Scope of the Project

The scope of our project are:

* To provide the overall picture of the MK82 general purpose bomb.
* Provide the short tutorial about how to prepare and assembly the MK82 GP bomb in part of head and tail.
* Use with VR device (Leap Motion and Oculus Rift)

## Expected Benefits

The benefits of our project are:

* Know the various types of bombs, MK-82.
* Know the step of assembly bombs type MK-82.
* Know the limitations of the VR device Leap motion and Oculus Rift.
* Training to use the VR devices.
* Can be used to develop a training program to other possibilities.

## Organization of the Document

This document consists of 6 chapters including:

1. **Introduction** – Introduction of the project the reason is that the project Inspired by the project aims and limits the benefits of doing this project.
2. **Background** – Background of the project. Inspired by background of Mark 82 bomb (MK-82), background of Virtual reality (VR), background of Oculus Rift, and background of Leap Motion.
3. **Analysis and Design** –Introduction of the project design. Inspired by the explanation of System Architecture Overview, System Structure Chart, Content Design and Game Design.
4. **Implementation** – This chapter explains the Hardware and Software used to implement in the project and the techniques in implementations.
5. **Testing and Evaluation** – This chapter explains the Testing method and Charts of Evaluation results
6. **Conclusion** – This chapter explains the conclusion of the project including Benefits, Problems and Limitation and Future Work

CHAPTER 2

# Background

This chapter is the Background of VRBomb project. It includes background of Mark 82 bomb (MK-82), background of Virtual reality (VR), background of Oculus Rift, and background of Leap Motion

## Literature Review

In last year, ICT faculty had modified the lesson in Multimedia track by added the new course, Virtual Reality, that teaching about definition of VR, VR devices and technic and implementation. After we studied and learned by ourselves about VR. We felt interested in it. We thought this is a new education topic in Thailand and its challenging. Before we developed the field in the senior project, we ever worked with VR devices in multimedia track project. The device that we used are sponsored by ICT faculty, on list below:

• Leap motion

• VR Hand controller

• Razor Hydra

• Kinect

• Steel series camera tracking

So we have the ability and familiarity with VR devices. We use the knowledge to apply in senior project. By choosing the suitable tool, leap motion, and increase the challenge by adopting with another device called Oculus Rift, including knowledge about creating a 3D model from the department of Computer animation and many more. Until, it was a project VRBomb eventually.

## Mark82 Bomb (MK-82)

This bomb are usually use in the mission that want to use maximum blast and explosive range. The Mk82 is generally use in air drop mission also known as free-fall, this type of bomb normally will equip with M904 at the nose fuze and M905 at tail part of the bomb. The case that use to cover the inner bomb body is considerately light with only 45% of their total body weight.  
  
 The MK82 with BSU-49/B high drag tail assembly. BSU-49/B is the ballute air bag that use to provide the bomb high speed with low attitude, the purpose of the air bag is to quickly slow the bomb drop speed so that the aircraft that carry and drop the bomb won’t get in the blast. The MK82 bomb tail part is also needed to assembly the ballute part that contain the balloon and parachute and the lanyard that will use to control the ballute part. The lanyard will active the balloon and parachute whenever it get drop from aircraft.

The MK80 series are invented in 1950 from the idea of to get the bomb that has less aerodynamic. All of the bomb in this series have similar architecture and have the cylinder shape also equip with the conical fins or retarder. The retarder can be attach either head part (nose) or tail part to guarantee the area of blast effect. The MK80 bomb can be used in many mission type and also effectively use for many type of target from small vehicle to antiaircraft artillery. Some of MK80 series are come along with the thermal protection to use in aircraft, these type of MK80 are develop to increase the time before bomb start to function and decrease time before the bomb reacted to fire and such.

Table 2‑1: Specifications of MK-82 bomb

|  |  |
| --- | --- |
| Specifications | |
| Class | 500 lb. General Purpose Bomb, Blast/Fragmentation |
| Guidance | Ballistic |
| Autopilot: | None |
| Propulsion: | None |
| Weight | 241 kg / 500 lbs. |
| Length | 2.21 m / 66.15 in. |
| Diameter | 10.75 in. |
| Warhead | 500 lbs. |
| Explosive | 89 kg / 192 lbs Tritonal, Minol II, or H-6 |
| Fuze | Variety for nose and tail. |
| Stabilizer | MAU-93/B, BSU49/B AIR, MK-15 Snakeye |
| Contractor | Nad Crane |
| Unit Cost | $268.50 |
| Aircraft | [A-10A](http://fas.org/man/dod-101/sys/ac/a-10.htm) [B-1B](http://fas.org/nuke/guide/usa/bomber/b-1b.htm) [B-2](http://fas.org/nuke/guide/usa/bomber/b-2.htm) [B-52](http://fas.org/nuke/guide/usa/bomber/b-52.htm) [F-4G](http://fas.org/man/dod-101/sys/ac/f-4.htm) [F-15A-E](http://fas.org/man/dod-101/sys/ac/f-15.htm) [F-16A-D](http://fas.org/man/dod-101/sys/ac/f-16.htm) [F-111D-F](http://fas.org/man/dod-101/sys/ac/f-111.htm) [F-117A](http://fas.org/man/dod-101/sys/ac/f-117.htm) |



Figure 2‑1: All MK-80 series Bomb

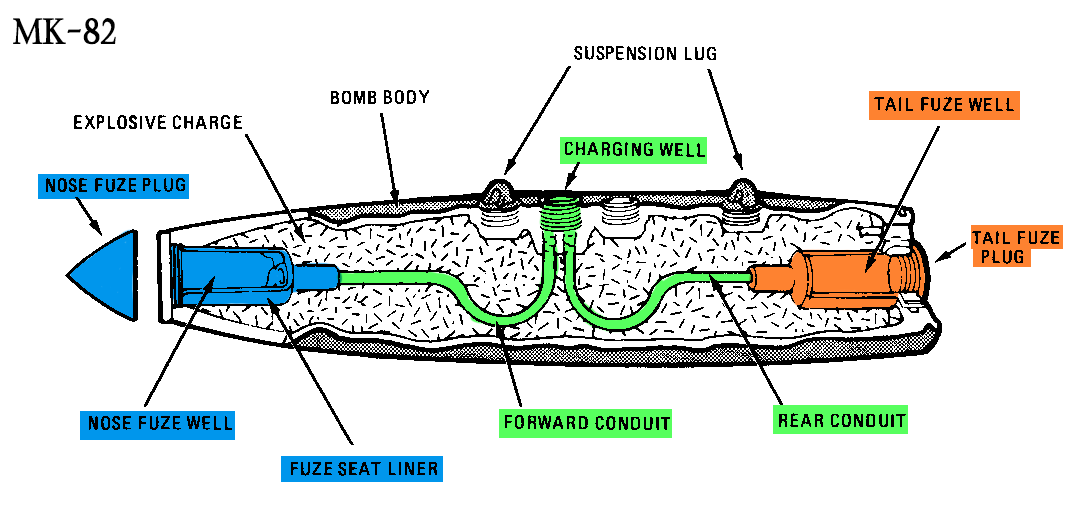


Figure 2‑2: The main elements of MK-82 bomb

****

Figure 2‑3: Screenshot of real MK-82 Bomb



Figure 2‑4: Screenshot of real MK-82 Bomb

We focus assembly at nose fuze, body section and tail fuze which are significant parts in Mark82 architecture for initiate the explosion.

## 2.3 Virtual Reality (VR)

Virtual reality is the system in the computer that create to fulfill the purpose of reproduce the actual experience, sense or specific situation in real world into computer program that user can interact with them by any mean.

In these day, the majority of the virtual reality environment are using computer screen or other stereoscopic to display the virtual reality environment and some may consist other component to make the user be able to interact with the system or make the user able to use more sensory in the system such as using headphone to sending sound in system instead of letting user hear thing in real world. The virtual reality can be reproduce some of the real world experience that normally not available to common people such as plane pilot simulation or something that can’t experience in real world such as VR game. In the present day, it still very hard to develop high level virtual reality that can immerse the user to absolutely believe that it real due to limitation of current technology , graphic or communication bandwidth.  
  
 Virtual reality can represent many type of application that normally associated with 3D environment, immersive, graphic, head mount display, data glove. In the book “The Metaphysics of Virtual Reality by Michael R.Heim” have classified the virtual reality into 7 main category that is simulation, interaction, artificiality, immersion, telepresence, full-body immersion and network communication but many people are confuse the virtual reality will only consist of using head mount display and data suit.

Table 2‑2: Example of VR devices in present (VR GEARS)

|  |  |
| --- | --- |
| Device Name | Picture |
| Oculus Rift DK1, DK2 | https://d3nevzfk7ii3be.cloudfront.net/igi/JJLiwOaQNSH3FRKN.large  http://cdn.arstechnica.net/wp-content/uploads/2014/03/9464.jpg |
| Leap Motion | http://www2.pcmag.com/media/images/393268-leap-motion-controller.jpg?thumb=y |
| Samsung Gear VR | https://cdn3.pcadvisor.co.uk/cmsdata/features/3529642/Samsung-Gear-VR-release-date-3.jpg |
| Razor Hydra | http://im.ziffdavisinternational.com/ign_me/screenshot/w/win-a-razer-hydra-motion-controller/win-a-razer-hydra-motion-controller_2ubw.jpg |
| Kinect Camera | http://ecx.images-amazon.com/images/I/516DD%2BTj73L._SL1300_.jpg |
| VR Hand controller | http://blogs-images.forbes.com/lorikozlowski/files/2014/06/Control-VR.jpg |

We also use Oculus Rift DK2 in our virtual reality simulator, by let the user wears it while using VRBomb simulator to turn the head around and perceive realistic 3D objects.

## 2.4 Oculus Rift DK2

Oculus rift DK2 is the head mount display, this model are consist of the 2nd model that develop by Oculus VR. This model are release for developer who interest to develop the program for the simple VR. The DK2 model are using the same screen that use in the Samsung smart phone.  
 The consumer version of the rift are expected to be release at some point in 2015 but the founder of Oculus “Palmer Lackey” said nothing about this topic. Oculus have release their developer version for 2 model since they active, DK1 at late 2012 and DK2 at mid-2014, so that the developer can grasp the basic concept to develop the program for Rift device. Even though their early release model target is developer but many normal user also purchase them to use in general purpose of playing game or test the program from the developer. There won’t be release anymore DK series but they will release consumer version instead, from the word of one of the Oculus founder “Palmer Luckey” they will develop a new head mount display that can be effective use while have a low price for the normal gamer to have a chance to experience them.



Figure 2‑5: All views of Oculus Rift DK2 (front, side, back, all tools)

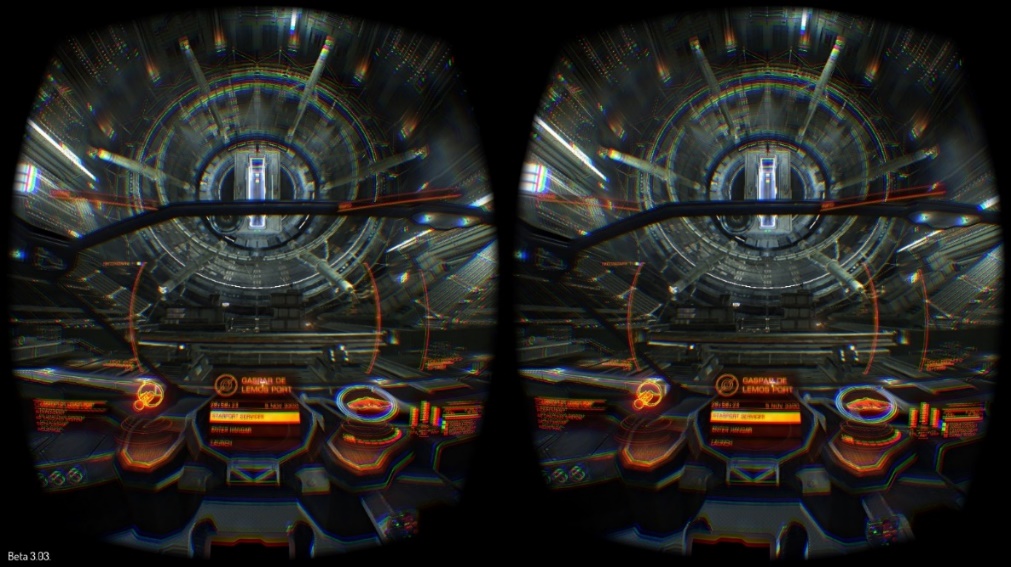
****

Figure 2‑6: Screenshot of Oculus Rift DK2 POV

Table 2‑3: Specifications of Oculus Rift DK2

|  |  |
| --- | --- |
| Specification |  |
| Resolution: | 960x1080 per eye |
| Refresh Rate: | 75 Hz, 72 Hz, 60 Hz |
| Persistence | 2 ms, 3 ms, full |
| Viewing Optics: | 100 degree field of view (nominal) |
| Cable: | 10’ (detachable) |
| HDMI: | HDMI 1.4b |
| USB Device: | USB 2.0 |
| Sensors (Internal): | Gyroscope, Accelerometer, Magnetometer |
| Update Rate: | 1000 Hz |
| Sensors (Positional): | Near Infrared CMOS Sensor |
| Update Rate: | 60 Hz |
| Weight: | 0.97 lbs. (without cable) |
| Includes: | HDMI to DVI Adapter  DC Power  Adapter  International Power Plugs  Nearsighted lens cups  Lens cleaning cloth |
| Software: | Oculus SDK which includes source code |
| Language Software: | OculusSDK [Unity/C#] |

We also use Oculus Rift DK2 in our virtual reality simulator, by let the user wears it while using VRBomb simulator to turn the head around and perceive realistic 3D objects

## 2.5 Leap Motion

Leap Motion, Inc. is the company from American that doing business that consist of hardware sensor that detect the hand and finger movement as input, analogous to a mouse without using hand contract.  
 They coming into market quietly until they release their first product in the name of The Leap in May 21, 2012 and release the software developer program in October 2012, they can distributed around 12,000 units to developers.  
 In May 2014, they succeed to launch the consumer beta version of their Leap motion in the name of “The Leap Motion Controller” that contain small USB peripheral device that can use in physical desktop in facing upward direction. This model are consist as 2nd version of their product, it use 2 monochromatic IR camera and 3 infrared LED as a sensor. The area that this can detect are roughly 1 meter radius in hemispherical area. The 3 LED will map the 3D IR light and the 2 camera will generate 300 frames in 1 second of reflected data that send from device via USB cable to PC that will calculate the data by the Leap Motion control program that using some method to produce the result.

Figure 2‑7: All views of Leap Motion (Front, side, back)

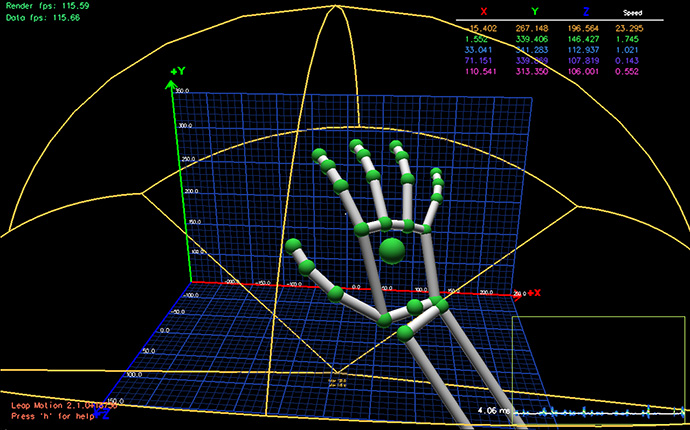


Figure 2‑8: Screenshot of skeletons tracking

Table 2‑4: Specifications of Leap Motion

|  |  |
| --- | --- |
| Specification |  |
| Height: | 0.5 inches |
| Width: | 1.2 inches |
| Depth: | 3 inches |
| Weight: | 0.1 pounds |
| System Requirement: | Windows 7 or 8 or Mac OS X 10.7 Lion ADM PhenomTMII or Intel CoreTM i3, i5, i7 |
| Processor: | 2 GB RAM |
| USB: | 2.0 port |
| Internet connection : | Yes |
| Includes: | Controller  USB Cables  Information Guide |
| Software: | Leap motion software and airspace |
| Language Software: | LeapSDK [Unity/C#] |

We also use Leap Motion controller to enable dexterous manipulation of virtual objects in our virtual reality simulator, by let the user motions hand over the device which attach to the Oculus Rift.

## 2.6 Related work

Before we created the project about visual reality, we has studied and found information about the other projects earlier, and has applied to our project. From all VR devices we had chosen the examples on leap motion and oculus rift over each project.

2.6.1 Planetarium

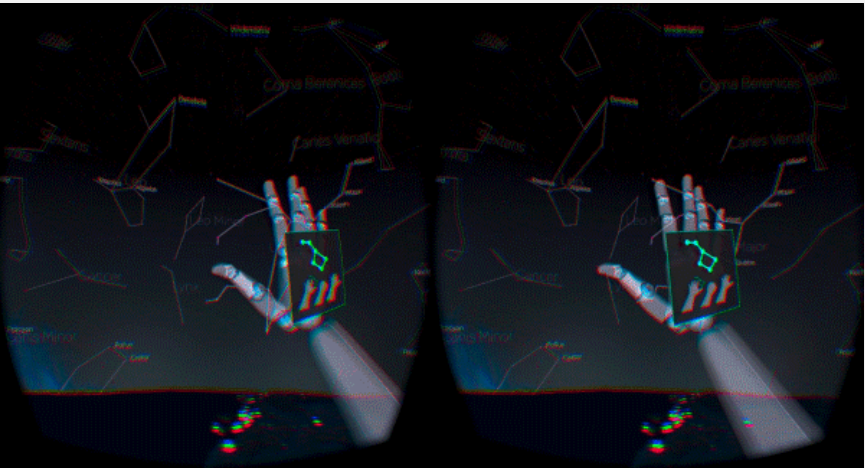


Figure 2‑9: Screenshot of Planetarium

Planetarium is the project that using Oculus rift and leap motion to let the user be able to see the virtual world and interact with them. This project is created to let the user see and learn about the star and many widget that consist time. This project is provide us the example of the virtual reality project that using Oculus rift and Leap motion. This project also give us roughly concept and limitation of the device.

2.6.2 Rift Coaster

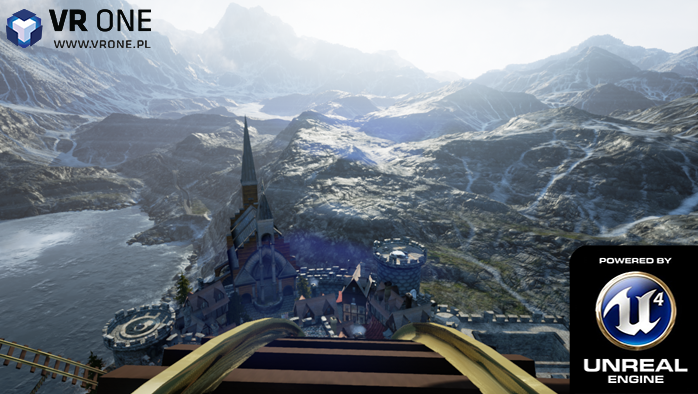


Figure 2‑10: Screenshot of Rift Coaster

Rift Coaster is the virtual reality system that designed to examine the potential of Oculus rift. It provide a 1st person perspective user can see 360 degrees all around.

This project is the example how to use the Oculus rift in virtual reality system and how to create the environment of virtual reality.

## 2.7 Contribution

Our project is create by the motivation to create the teaching section of basic operation and process of assembly the bomb. Our project have many similar concept to many other program but our project is the first project that using virtual reality as teaching media to teach about bomb assembly and our project also can adapt into other teaching media too.

For the architecture limits of our software, to create other assembly simulation (e.g. gun assembly), you can utilize some resources from our project. For example, you can copy our user’s hand prefab (leap motion), user’s camera prefab (oculus rift) and other controller to your project. You also get the idea of user interaction (grabbing) or validating assembly (i.e. trigger), and then port to your simulation.

However, there are the limitations that you should build in your own way to complete the other assembly simulation such as: creating own model, creating own sequence assembly code and creating own set of triggers.

CHAPTER 3

# Analysis and Design

This chapter will show the explanation of System Architecture Overview, System Structure Chart, Content Design and Game Design.

## 3.1 System Architecture Overview

The System Architecture of VRBomb is shown in Figure 3-1 below. There are three main components including Perception, Interaction and Processing.

|  |  |
| --- | --- |
| Oculus Rift DK2  Leap Motion  Positional Tracking  VRBomb Simulator | |
| Project : VRBomb  System : VRBomb | Major Advisor : Dr. Mores Prachyabrued |
| Description: The System Architecture of VRBomb is shown in Figure 3-1 below. There are three main components including Perception, Interaction and Processing. | |

Figure 3‑1: System Architecture Overview of VRBomb

1. Perception

Oculus Rift provides stereoscopic 3D visuals. Positional tracking device tracks the user’s viewpoint.

1. Interaction

Leap Motion tracks the user’s hands and enables dexterous manipulation of virtual objects.

1. Processing.

Unity3D provides software simulation and real-time analysis of the validity of the assembly.

## System Structure Chart

The System Structure Chart of VRBomb is shown in Figure 3-1 below. There are three main components including Assembly Head Part and Assembly Tail Part.

|  |  |
| --- | --- |
|  | |
| Project : VRBomb  System : VRBomb | Major Advisor : Dr. Mores Prachyabrued |
| Description : The System Structure Chart of VRBomb consists of two main Part including Head Part and Tail Part of MK-82 | |

Figure 3‑2: System Structure chart of VRBomb

1. Tutorial

At the start of the simulator the program must leads the user to tutorial scene for teach the user how to use and interact with device and simulator.

Support gesture interactions:

* Moving hands and Moving head
* Grabbing objects

1. Head Assembly

When the user selects this part, it directly goes to assembly scenes and displays all the model of MK-82’s head part.

* 1. Install stop screw

The first step of head assembly part

Support interactions:

* Moving hands and Moving head
* Grabbing objects
* Connecting 2 elements together
* Screen tapping gesture
  1. Install delay element

The second step of Head assembly part

Support interactions:

* Moving hands and Moving head
* Grabbing objects
* Hovering an object
* Connecting 2 elements together
  1. Install arming wire

The third step of Head assembly part

Support interactions:

* Moving hands and Moving head
* Grabbing objects
* Connecting 2 elements together
  1. Install retaining clip

The forth step of Head assembly part

Support interactions:

* Moving hands and Moving head
* Grabbing objects
* Connecting 2 elements together
* Screen tapping gesture
  1. Install tray

The fifth step of Head assembly part

Support interactions:

* Moving hands and Moving head
* Grabbing objects
* Covering an object with another object
  1. Restart

When user finish the part, the program allow user to re-assembly of the part again.

* 1. Tail Assembly

When user finish the part, the program allow user go to assembly another part.

1. Tail Assembly

When the user selects this part, it directly goes to assembly scenes and displays all the model of MK-82’s tail part.

* 1. Install safety element

The first step of Tail assembly part

Support interactions:

* Moving hands and Moving head
* Grabbing objects
* Connecting 2 elements together
* Screen tapping gesture
  1. Install clip

The second step of Tail assembly part

Support interactions:

* Moving hands and Moving head
* Grabbing objects
* Connecting 2 elements together
  1. Install arming pin

The third step of Tail assembly part

Support interactions:

* Moving hands and Moving head
* Grabbing objects
* Connecting 2 elements together
* Screen tapping gesture
  1. Restart

When user finish the part, the program allow user to re-assembly of the part again.

* 1. Head Assembly

When user finish the part, the program allow user go to assembly another part.

## Content Design

This simulator is about training the user to know the assembly steps of the MK-82 Bomb. So we received the information about Bomb and procedures for assembly from DTI to study and applied to the program including all of API and library of our devices as follow.

### MK-83 bomb assembly information

Bomb must be built to be able to withstand heat to a certain point and be able to ignore the impact of the ordinary handling also be able to drop from aircraft in safe condition when in-flight emergencies occur.  
 Bomb detonation will be handle by the fuse. Fuse is the device that use to control the bomb detonation when the certain time or condition are met to detonate depend on the setting that is set beforehand. A bomb fuse can be mechanical or electronic. The type of fuse is determine by the action that each type of fuse use to detonate the bomb, mechanical is use mechanical action such as slug release and the electrical will use the electric to start the detonation. It has the sensitive explosive elements (the primer and detonator) and the necessary mechanical/electrical action to detonate the main buster charge. The primer-detonator explosion is relayed to the main charge by a booster charge. This completes the explosive train.

Mk80 (series) has many component in it body but we are lacking the knowledge of the overall component and many of them are not accurate produced and some of the component may different from the real models and functions. These are the component that we are able to produce them.

3.3.1.1 Head part   
 Nose Fuse Well: The container of nose fuse, it located in the front of the bomb.

The M904 (series) fuse is a mechanical type fuse that use in MK80 (series) bomb. This fuse is act as detonator-safe that prevent the accidently detonate, it contain 2 observation window which can use to determine the safe/arm condition of the fuse. These fuse does not have a special locking feature designed into the fuse for shear safety if the bomb is accidentally dropped. However, detonation is unlikely if the collar (forward end of the fuse) is sheared off by the accidental drop before arming is complete.  
 The fuse may need to setup the condition time for activate the arming or functioning depend on the use for each mission. There are 2 to 18 number with 2 increment each step to determine the delay time in the time unit second along with any combination of six function delays from instantaneous to 250 milliseconds may be selected. The fuse are driven by permanent mounted arming vane that make the arming time at release speed range from 170 to over 525 knots.  
 Functioning times are determined by the installation of an M9 delay element. Any one of six delay elements may be installed. Each delay element is identified by the functioning delay time stamped on the element body—NONDELAY (instantaneous), 0.01, 0.025, 0.05, 0.1, or 0.25 second.

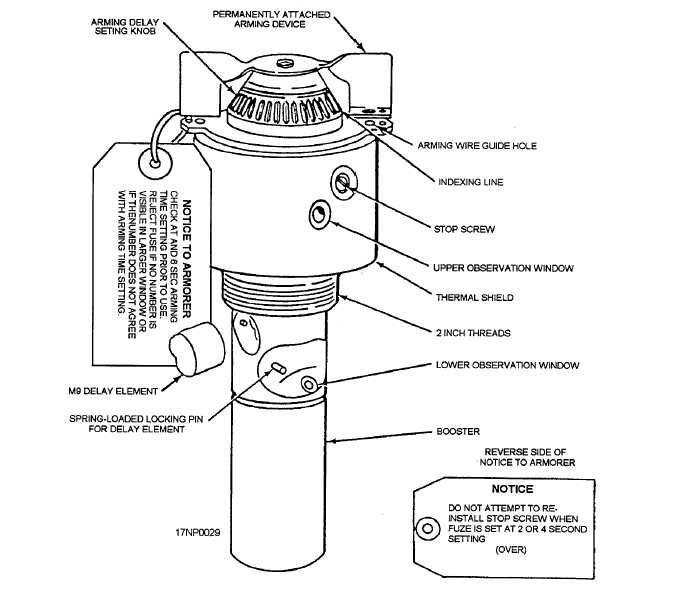
The M904 (series) fuse contains approximately 1 1/2 ounces of tartly in the booster, which is located at the base of the fuse body. The entire fuse weighs about 2 1/3 pounds and is 9 1/4 inches long. The M904E4 is a thermally protected fuse. . It is especially designed for use with the thermally protected Mk 80 (series) general-purpose bombs and the thermally protected M148E1 adapter booster. This will make the start-up time require for the bomb to start function took significant longer than usual.

Figure 3‑3: Nose Fuze M940 details

M904E2/E3/E4 Mechanical Impact Nose Fuze preparation:

1) First step is to select the delay time that necessary for each mission through the upper observer window.

2) After setting the time delay, the next step is to use screwdriver to tighten the stops crew that locate near the upper observer window.

3) Next is to set the other delay time at the lower observer window.

4) After set the time on lower observer window, the next step is to hold the stick button that located near the lower observer window and then put the delay unit into the hole.

5) Next step is to guide the arming wire through the guide hole that located on upper part of the nose fuze.

6) After guide the arming wire through the hole then you need to use the retainer clip to restrain the arming wire from going off the position.

3.3.1.2 Tail Part

FMU-143E/B Electric Tail Fuse: The FMU-143E/B fuse is used with the GBU-24B/B. It is initiated by the FZU-32B/B initiator that act as the power generator and power supply to arm the fuse. This fuse safe condition will determine by the presence of the safety pin and arming wire through the pop-out pin (gag rod).

FMU-143E/B Electric Tail Fuze preparation:

1. First is to attach the safety gag, put the safety gag on the safe pin.
2. After put the safety gag onto safety pin then suing the clip to restrain them.
3. After restrain the safety pin and gag then attach the arming pin and arming wire that connect to nose fuze.

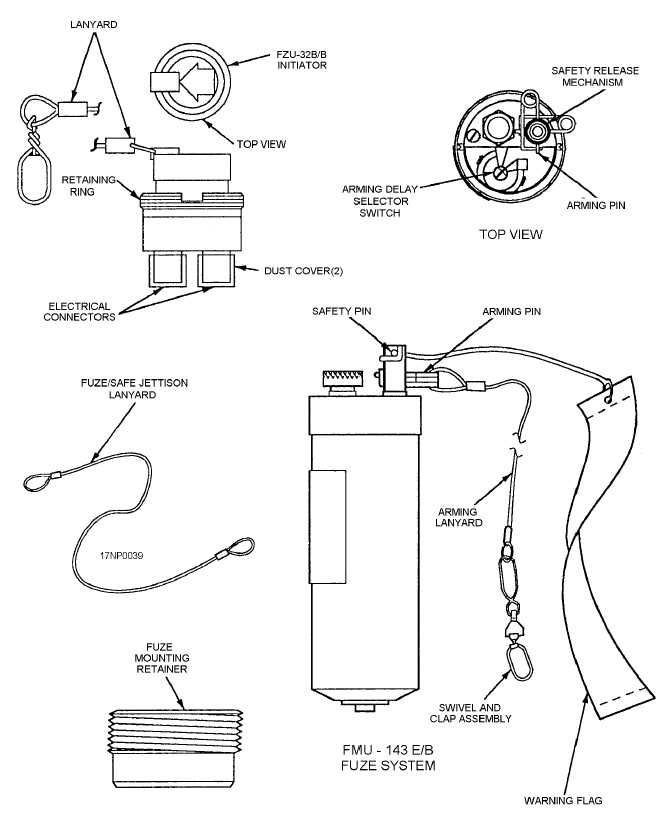


Figure 3‑4: GBU-24B/B details

### Leap Motion API

The Leap Motion system acknowledge and tracks hands, or finger-like tools. The device perform in an intimate proximity with high precision and tracking frame rate and reports discrete positions, gestures, and motion.

The Leap Motion controller uses optical sensors and infrared light. The sensors are regulate along the y-axis (upward) when the controller is in its standard operating position and have a field of view of about 150 degrees. The proficient range of the Leap Motion Controller extends from about 25 to 600 millimeters above the device.



Figure 3‑5: The Leap Motion controller’s view of your hands

Detection and tracking will work well when the controller has a clear, high-contrast view of an object’s shadow. The Leap Motion software combines its sensor data with an intramural model of the human hand to help manage with challenging tracking conditions.

* + - 1. Coordination System

The Leap Motion system use a right-handed Cartesian coordinate system. The origin is at the center at the top of the Leap Motion Controller. The x-axis and z-axis are lie in the horizontal plane, with the x-axis running parallel to the long edge of the device. The y-axis is vertical, with positive values increasing upwards. The z-axis has positive values increasing toward the user.

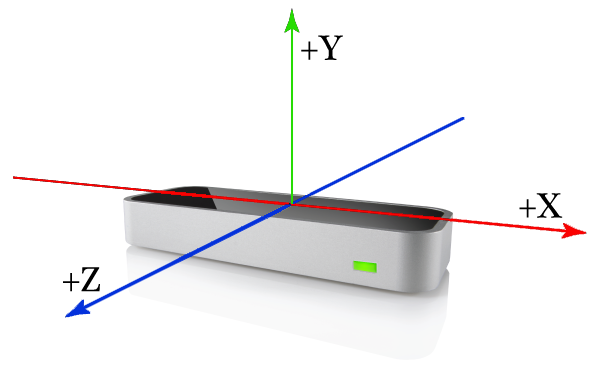


Figure 3‑6: The Leap Motion right-handed coordinate system.

Table 3‑1: The Leap Motion API measures physical quantities

|  |  |
| --- | --- |
| The Leap Motion API measures physical quantities with the following units | |
| Distance: | millimeters |
| Time: | microseconds (unless otherwise noted) |
| Speed: | millimeters/second |
| Angle: | radians |

* + - 1. Motion Tracking Data

The Leap Motion controller tracks hands, fingers, and tools in its point of view, it allow updates as a set or frame of data. Each Frame object show a frame including lists of tracked entities, such as hands, tools and fingers, as well as accepted gestures and factors explaining the overall motion in the scene. The Frame object is importantly the root of the Leap Motion data model.

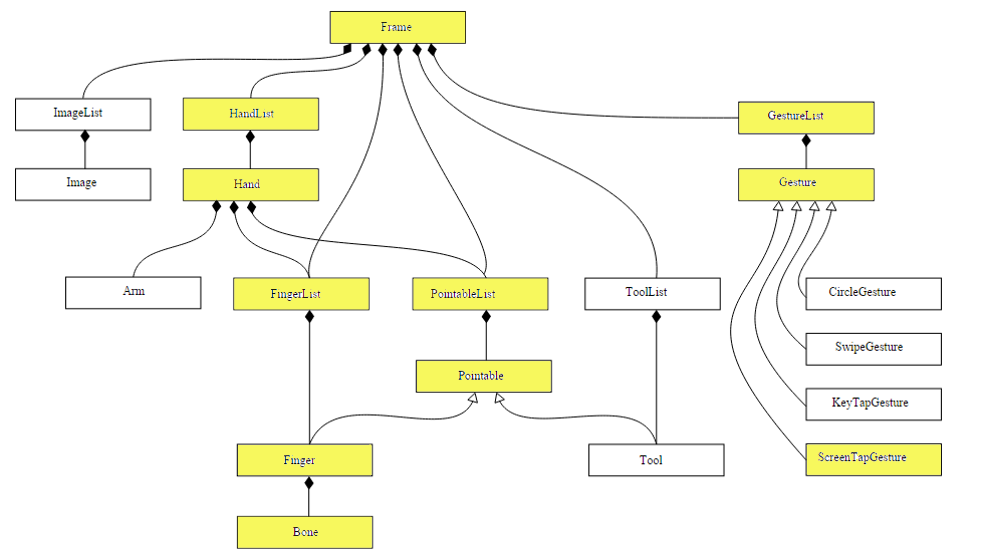


Figure 3‑7: Motion Tracking Data structure chart

The Tracking Data Following:

1. Hands
2. Fingers
3. Gestures
4. Hands

The hand model come up with information about the position, and other characteristics of a detected hand and lists of the fingers related with the hand.

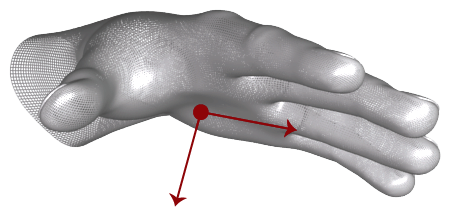


Figure 3‑8: The Hand vectors define the orientation of the hand.

More than two hands can appear in the hand list for a frame. However, we recommend using at most two hands in the Leap Motion Controller’s point of view for best motion tracking quality.

1. Fingers

The Leap Motion controller come up with information about each finger on a hand. If part of a finger is not visible, the finger characteristics are estimated based on recent observations and the anatomical model of the hand. Fingers are identified by type name, i.e. thumb, index, middle, ring, and pinky.



Figure 3‑9: Fingertip vectors provide the position of a fingertip and the general direction in which a finger is pointing.

A Finger object provides a Bone object describing the position and orientation of each anatomical finger bone. All fingers contain four bones ordered from base to tip.

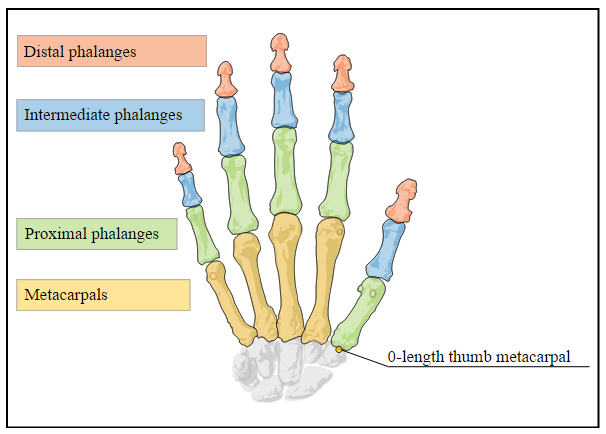


Figure 3‑10: All fingers contain four bones ordered from base to tip

The bones are identified as:

* Metacarpal – the bone inside the hand connecting the finger to the wrist (except the thumb)
* Proximal Phalanx – the bone at the base of the finger, connected to the palm
* Intermediate Phalanx – the middle bone of the finger, between the tip and the base
* Distal Phalanx – the terminal bone at the end of the finger

1. Gestures

The Leap Motion software realize movement patterns as gestures, this could indicate a user intent or command. Gestures are observed for each finger or tool one by one. The Leap Motion software uses gestures observed in a frame in the same way that it reports other motion tracking data like fingers and hands.

In this project we use only “Screen Tap” gesture. The ScreenTapGesture class displays a tapping gesture by a finger. A screen tap gesture is activate when the tip of a finger pokes forward to the position and then springs back to the original position.

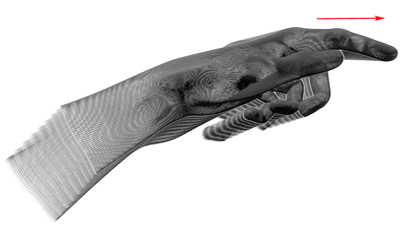


Figure 3‑11: Screen Tap Gesture

### Oculus Rift DK2 Prefabs

We use the oculus rift DK2 on the project in order to provide a more realistic visualization software which oculus rift DK2 has been a complex and diverse. However, we have studied only the most needed and can be applied to our work there. There is Oculus Rift DK2 Prefabs. The current integration of the Rift into Unity applications focuses around two prefabs that can be added into a scene:



Figure 3‑12: OVRCameraRig and OVRPlayerController

To use, simply drag and drop one of the prefabs into your scene.

* + - 1. OVRCameraRig

OVRCameraRig replaces the native Unity camera within a scene. You can drag an OVRCameraRig into your scene and you will be able to start viewing the scene with the Rift. Make sure to turn o any other camera in the scene to ensure that OVRCameraRig is the only one being used.



Figure 3‑13: OVRCameraRig, expanded in the inspector

OVRCameraRig contains two Unity cameras, one for each eye. It is meant to be attached to a moving object (such as a character walking around, a car, a gun turret, etc.) This replaces the conventional camera.

The following scripts (components) are attached to the OVRCameraRig prefab:

* **OVRCameraRig.cs**

OVRCameraRig is a component that controls stereo rendering and head tracking. It maintains three child "anchor" Transforms at the poses of the left and right eyes, as well as a virtual "center" eye that is half-way between them. This component is the main interface between Unity and the cameras. This is attached to a prefab that makes it easy to add a Rift device into a scene.

**Important:** All camera control should be done through this component. You should understand this script when integrating a camera into your own controller type.

* **OVRManager.cs**

OVRManager is the main interface to the Oculus Rift hardware. It includes wrapper functions for all exported C++ functions, as well as helper functions that use the stored Oculus Rift variables to help configure camera behavior.

This component is added to the OVRCameraRig prefab. It can be part of any application object. However, it should only be declared once, because there are public members that allow for changing certain Rift values in the Unity inspector.

OVRManager.cs contains the following public members:

Table 3‑2: OVRManager.cs contains

|  |  |
| --- | --- |
| Native Texture Scale | Each camera in the camera controller creates a Render Texture that is the ideal size for obtaining the ideal pixel deity (a 1-to-1 pixel size in the center of the screen post lens distortion). This field can be used to permanently scale the cameras' render targets to any multiple ideal pixel deity, which gives you control over the trade-o  between performance and quality. |
| Virtual Texture Scale | This field can be used to dynamically scale the cameras render target to values lower than the ideal pixel deity, which can help reduce GPU usage at run-time if necessary. |
| Use Position Tracking | If disabled, the position detected by the tracker will stop updating the HMD position. |
| Use Rotation Tracking | If disabled, the orientation detected by the tracker will stop up- dating the HMD orientation. |
| Mirror to Display | If the Oculus direct-mode display driver is enabled and this option is set, the rendered output will appear in a window on the desktop in addition to the Rift. Disabling this can slightly improve performance. |
| Time Warp | Time warp is a technique that adjusts the on-screen position of rendered images based on the latest tracking pose at the time the user will see it. Enabling this will force vertical-sync and make other timing adjustments to minimize latency. |
| Freeze Time Warp | If enabled, this illustrates the effect of time warp by temporarily freezing the rendered eye pose. |
| Reset Tracker On Load | This value defaults to True. When turned off, subsequent scene loads will not reset the tracker. This will keep the tracker orientation the same from scene to scene, as well as keep magnetometer settings intact |
| Mesoscopic | If true, rendering will try to optimize for a single viewpoint rather than rendering once for each eye. Not supported on all platforms. |
| Eye Texture Format | Sets the format of the eye Render Textures. Normally you should use Default or DefaultHDR for high-dynamic range rendering. |
| Eye Texture Depth | Sets the depth precision of the eye Render Textures. May fix z-fighting artifacts at the expense of performance. |

## Simulator Design

This explain about how the user interact with the simulator including all of components design in the project.

### Gameplay

At the start of the simulator, the program will lead the user to the tutorial scene that shows the tips how user reactions with device to make the best interaction. Then the simulator will allows user to choose the part of MK-82 bomb (Head Assembly or Tail Assembly) for assembly. In the scene of Assembly MK-82, on each part will contains all of elements on each part which allow user to training the assembly step by step. When the user finishes on each part, the simulator will allow the user to re-assembly the part or go to assembly another part of MK-82 Bomb.

### Conditions

* User can chooses between Head Assembly part and tail Assembly part
* User must follows the instruction which it provided by the simulator.
* User must finished the last step before go to another part.

### Scenes

This simulator provides 3 tutorial scenes and 2 assembly scenes.

Table 3‑3: All scenes in VRBomb Simulator

|  |  |
| --- | --- |
| Tutorial scene 1  Guide the user to use the Oculus Rift DK2 |  |
| Tutorial scene 2  Guid the user to interaction with leap motion effectively |  |
| Tutorial scene 3  Guid the user how to grab the objects effectively |  |
| Tutorial scene 4  Guid the user for the interface in the simulator |  |
| MK-82 Head Assembly scene  The area for assembly the head part of MK-82, provides overall elements that use in the part |  |
| MK-82 tail Assembly scene  The area for assembly the tail part of MK-82, provides overall elements that use in the part |  |

### Interfaces

Table 3‑4: All interfaces in VRBomb Simulator

|  |  |
| --- | --- |
|  | Hint Panel  Show the step of assembly the bomb on each part. *(Only the current step, user must finish the current step then the panel will show the other step)* |
|  | **Menu Panel**  Allow user to should the action by tapping the button |
|  | **Game Message**  Display the object’s name, while user hover it. |

### Effects

Table 3‑5: All effects in VRBomb Simulator

|  |  |
| --- | --- |
|  | Particle effects Indicate which spot user have to assembly. |
|  | **Hover Effect**  Indicate which the model that the hand hover over. |

### Hand Graphic Model

*\*this models are attached in the Leap motion API, we not created by ourselves.*

Table 3‑6: All equipment and other model in VRBomb Simulator

|  |  |
| --- | --- |
|  | Minimal Hand  User can chooses the type of hand to show in the simulator. |
|  | **WhiteCutHand**  User can chooses the type of hand to show in the simulator. |
|  | **RobotCutHand**  User can chooses the type of hand to show in the simulator. |
|  | **PolygonHand**  User can chooses the type of hand to show in the simulator. |
|  | **HumanCutHand**  User can chooses the type of hand to show in the simulator. |
|  | **WhiteFullHand**  User can chooses the type of hand to show in the simulator. |
|  | **RobotFullHand**  User can chooses the type of hand to show in the simulator. |
|  | **HumanFullHand**  User can chooses the type of hand to show in the simulator. |

### Models

Table 3‑7: All MK-82 parts model which it used in VRBomb Simulator

|  |  |  |
| --- | --- | --- |
|  | | Arming Wire  Use for connect the nose fuse and tail fuse. |
|  | | **Arming Wire Retainer**  Located on the nose fuse head part. Use to restrain the arming wire that come through the wire guide hole on nose fuse. |
|  | | **Nose Fuze Delay**  Located on nose fuse body. Use to prevent the accidently configuration of delay time before release the fuse. |
|  | | **Nose Fuze Wells**  This component are act as the container for nose fuse to load in the bomb body. |
|  | | **Nose Fuze**  The main component for the bomb. This component are acting as a detonator for the bomb. |
|  | | **Stop Screw**  Located on nose fuse. This component is use to prevent accidently configuration delay time on upper section. |
|  | | **Clip**  Located on tail fuse. This component is use to bind the safety pin with tail fuse main body. |
|  | | **Safety Pin**  Located on tail fuse. This component is use to shift the status of the fuse to safety when its place on tail fuse. |
|  | **Tail Fuze**  One of the main component of the bomb. This component usually act as the safety device to ensure that the bomb won’t be detonate when not in use. | |

CHAPTER 4

# Implementation

This chapter explains the hardware and software used to implement in VRBomb and also including the techniques are used in this project.

## Hardware and System Environment

* Operating System and Utilities Applications
  + Microsoft Windows 7: An Operating System used to implement the project.
  + Photoshop: Software for creating the entire graphic in the simulator.
  + Blender: Software for creating the models and scenes in the simulator.
* Hardware
  + Oculus Rift DK2: Virtual Reality gear used for created the real 360 degree environment in the simulator.
  + Leap Motion: Virtual Reality gear used for created the real hands in the simulator.
* Editor
  + Unity 3D: A project builder and runner.
* Programming Tools
  + Oculus Rift SDK: A plug-in that used for connect and implement between Oculus Rift DK2 and Unity 3D.
  + Leap Motion SDK: A plug-in that used for connect and implement between Leap Motion and Unity 3D.

## Implementation Guide and Techniques

This topic will explain about how to set up an environment before implementations and the important techniques used in this project

### Setting up the Environment

The instructions below will explain the main steps of setting up environment before starting the implementation of VRBomb Simulator.

1. Download and install Unity4.6.1.f1
2. Download and install latest version of Leap Motion Tracking Software (<https://www.leapmotion.com/setup>)
3. Download and install Oculus Rift Runtime 0.4.4-beta (<https://developer.oculus.com/downloads/#version=pc-0.4.4-beta>)
4. Attach Leap Motion to Oculus Rift (<https://community.leapmotion.com/t/unofficial-vr-mount-thread/2296>)
5. Connect both devices to PC via USB port
6. Setup Oculus Rift Display Mode
   1. Open Oculus Configuration Utility.exe
   2. Select Tools > Rift Display Mode…
   3. Select Extend Desktop to HMD
   4. Click Apply

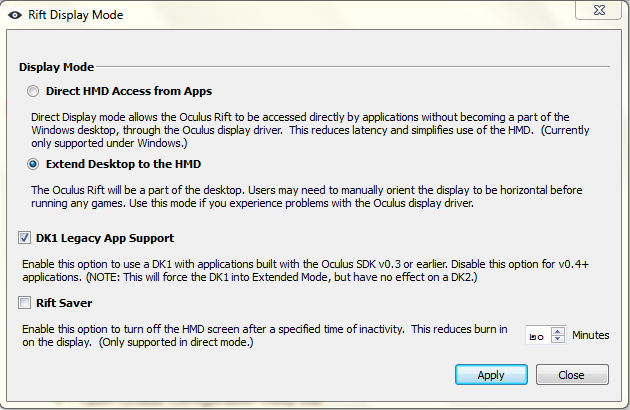


Figure 4‑1: Interface Rift Display Mode

1. Setup Leap Motion Control Panel
   1. Open Leap Motion Control Panel.exe
   2. Configure General page like these



Figure 4‑2: Interface Leap Motion Control Panel

* 1. Configure Tracking page like these

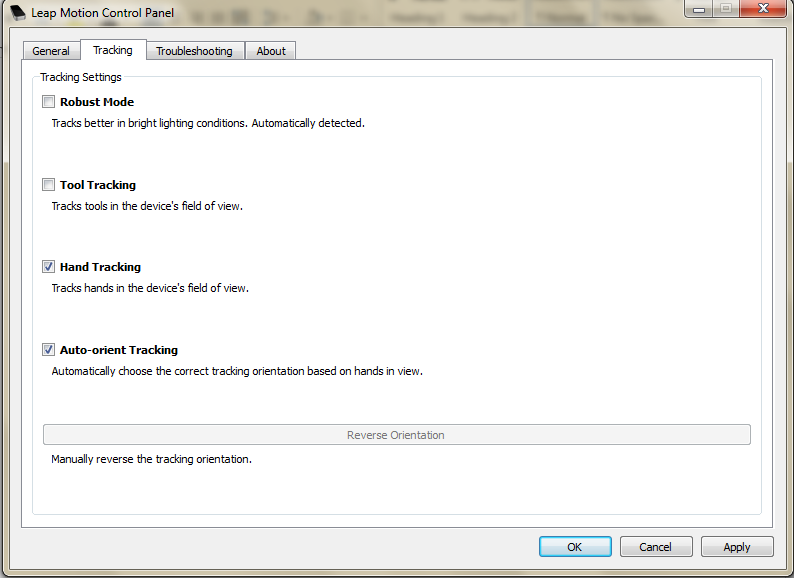


Figure 4‑3: Interface Leap Motion Control Panel setting

* 1. Click OK

1. Download and extract Leap Motion VR Asset 2.2.2.24469 (Pro)

(<https://developer.leapmotion.com/downloads/unity/4.6>)

1. Open and activate Unity4.6.1.f1 to pro version
2. Create a new 3D project without import any packages

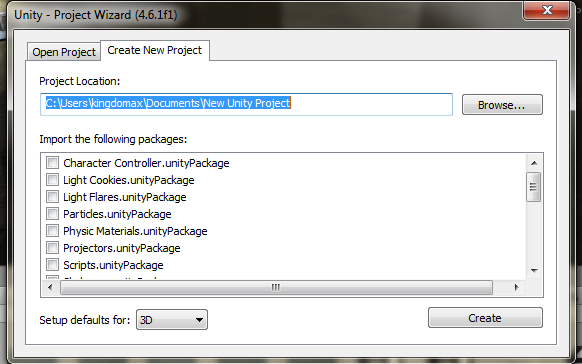


Figure 4‑4: Unity 3D project create project

1. Import Leap Motion VR Asset to the project
   1. At menu command, select the **Assets > Import Package > Custom Package**

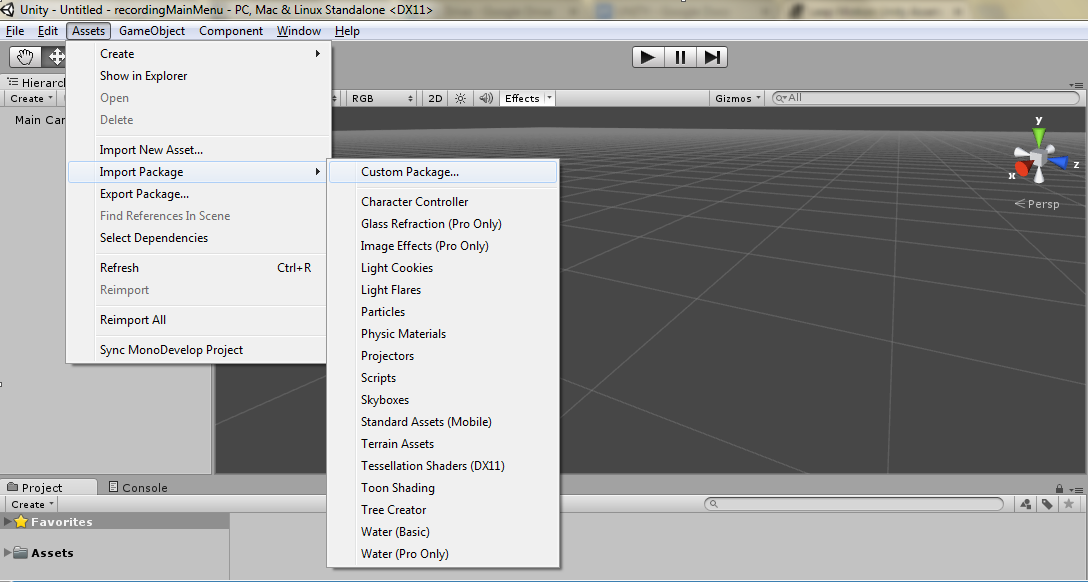


Figure 4‑5: Unity 3D imported package to project

* 1. Locate the downloaded asset package and click open

1. For debug the project with Oculus Rift
   1. Select Game Tab
   2. Toggle Maximize on Play



Figure 4‑6: Unity 3D Game Interface

* 1. Drag Game Tab from Unity Editor to Oculus Rift Display
  2. Also click the maximize button on the top right of Game Tab (to the left of X button).

1. Try to run the project with sample scene
   1. Select Asset in Project Tab

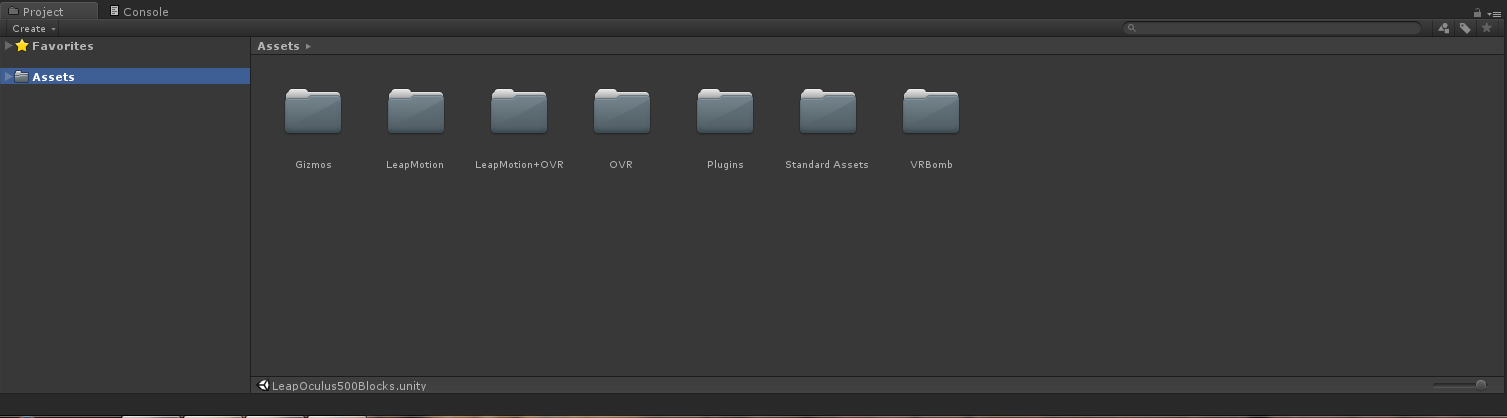


Figure 4‑7: Unity 3D Assets menu

* 1. Select LeapMotion+OVR > Scenes
  2. Double click LeapOculus500Blocks.unity
  3. Press Play Button

### Oculus integration in Unity

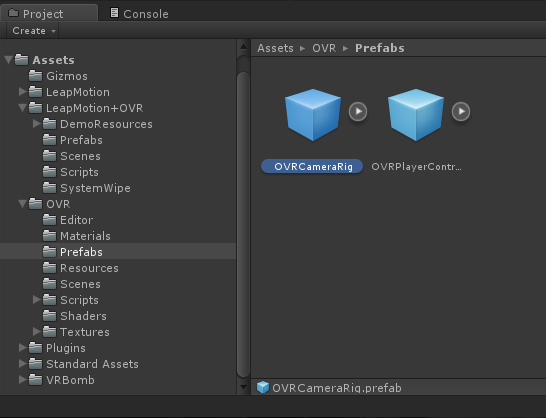
Assume that we’re already integrated “LeapMotionVRAsset” in this project. Then, import “OVRCameraRig” prefab to the scene. 

Figure 4‑8: Unity asset Oculus Rift Prefabs

Fill out component’s variables of OVRCameraRig. We recommend to set value 3 at position Y axis variable because it is appropriate height of camera view from the ground and also enable using position tracking device.



Figure 4‑9 Components of OVRCameraRig

Lastly, don’t forget to turn off any other camera object in the scene to ensure that this camera prefab is the only one being used.

### Leap Motion integration in Unity

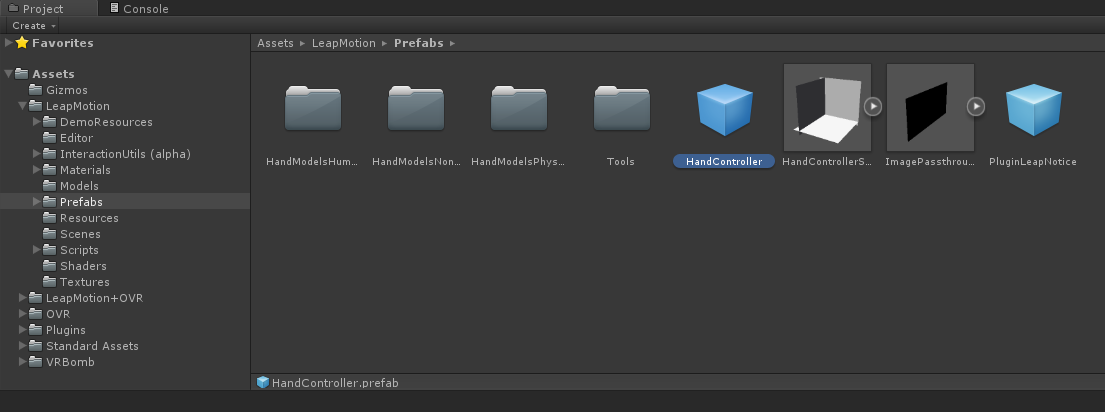
Import “HandController” prefab from “LeapMotionVRAsset” to the scene. Then, place “HandController” under the “CenterEyeanchor” in order to move HandController along with camera position 

Figure 4‑10: HandController prefab

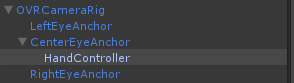


Figure 4‑11: Child of CenterEyeAnchor under OVRCameraRig

Set up component’s variables of HandController. We have rotate the position of HandController in the same way of hardware device, scale hand size a little bit down and also enable IsHeadMounted variable to optimize tracking in different axis.

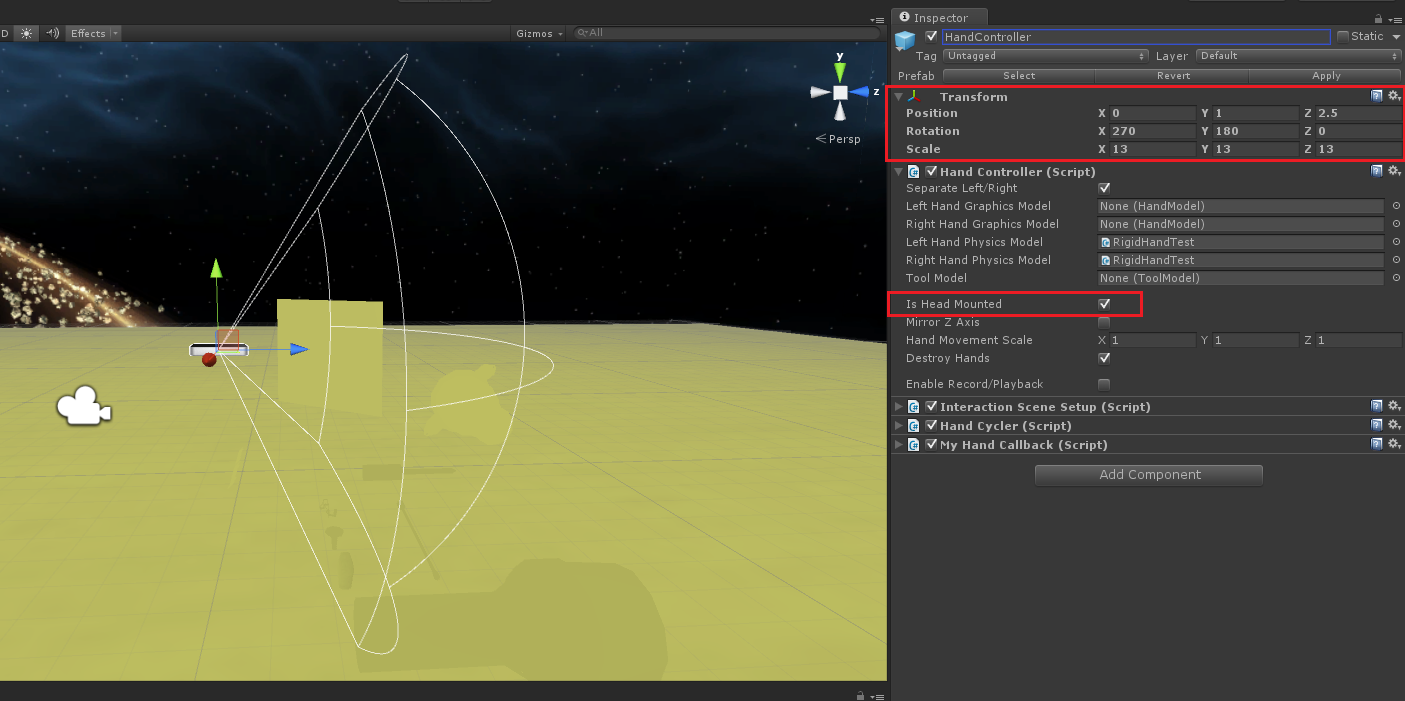


Figure 4‑12: Component of HandController

Select RigidHand prefab in imported asset and drag to the scene

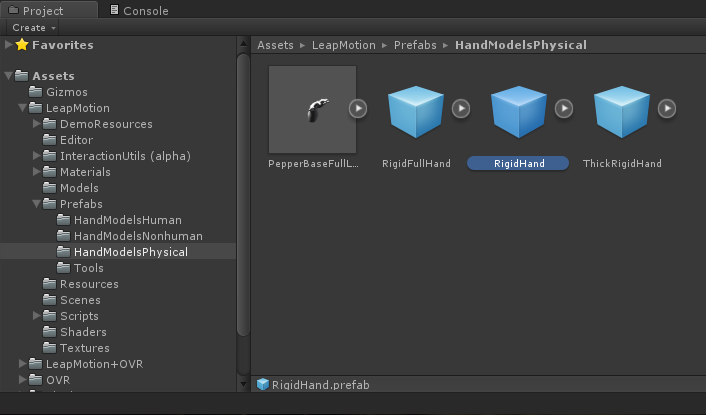


Figure 4‑13: Rigid Hand Model asset

Again, set up variable at HandController to tell that Rigid Body is your physical hand and let HandController maintain it

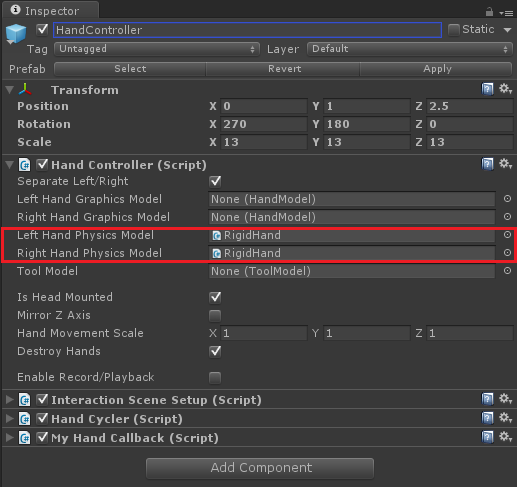
Setup HandController script of HandController object

Figure 4‑14: Set up physical hand model

### How to interact (grab) object

The very significant spot of overall implementation is grabbing the object. Attach InteractionSceneSetup.cs script to HandController object.

|  |
| --- |
| using UnityEngine; |
| using System.Collections; |
| using System.Linq; |
| using Leap.Interact; |
|  |
|  |
| /// <summary> |
| /// Configuration & initialization scripts to setup Leap Interaction & reflect the Unity scene in it. |
| /// </summary> |
| public class InteractionSceneSetup : MonoBehaviour { |
|  |
| public const float MM\_PER\_M = 1000.0f; |
|  |
| /// <summary> |
| /// Defines the relative scale of the Unity and Leap worlds. Leap unity is equivalent to one millimeter. |
| /// This variable specifiers the length of the Unity unit in millimeters. |
| /// </summary> |
| [HideInInspector] |
| public float clientUnitLengthInMillimeters = 50f; |
|  |
| /// <summary> |
| /// Specifies where the Leap device is located in relation to the reference object, or other object used as a reference for |
| /// positioning the Leap world. This is specifierd in Leap unity (e.g. in real-world millimeters). |
| /// By default, set this to [0,0,0] when using HandController as the reference object. Set it to [0, -200, 200] when using the camera. |
| /// </summary> |
| [HideInInspector] |
| public Vector3 OffsetCameraToLeap = new Vector3(0, 0, 0); |
|  |
| /// <summary> |
| /// The refernce object that the leap device is 'attached' to. As this object moves & reorients, the Leap Interaction hands will move along. |
| /// </summary> |
| [HideInInspector] |
| public HandController ReferenceObject; |
|  |
| /// <summary> |
| /// Shows the exact hand-proxy shapes used by Leap Interaction. You can also use those shapes to interact/collide/push regular rigidbodies in the scene. |
| /// </summary> |
| [HideInInspector] |
| public bool ShowInteractionHands = false; |
|  |
| [HideInInspector] |
| public bool ShowVisualizer = false; // disabled flag |
| [HideInInspector] |
| public bool EnableInteractionCollisions = false; |
|  |
| public bool AllowPinchWithAnyFinger = false; |
| public bool DisableHoldingOnPointingIndexFinger = false; |
|  |
| static public int LayerForHands = 30; |
| static public int LayerForHeldObjects = 31; |
| static public int LayerForReleasedObjects = 0; // Default |
|  |
| private bool initialized = false; |
|  |
| public static void EnsureInstanceInitialized() { |
| // find instance. |
| InteractionSceneSetup leap = GameObject.FindObjectOfType<InteractionSceneSetup>() as InteractionSceneSetup; |
| leap.EnsureInitialized(); |
| } |
|  |
| public void EnsureInitialized() |
| { |
| if (!initialized) { |
| Initialize(); |
| initialized = true; |
| } |
| } |
|  |
| void Awake() { |
| EnsureInitialized(); |
| } |
|  |
| /// <summary> |
| /// Performs initialization of Leap Interaction. |
| /// </summary> |
| public void Initialize () { |
| clientUnitLengthInMillimeters = MM\_PER\_M / transform.localScale.x; |
| ReferenceObject = GetComponent<HandController>(); |
|  |
| m\_scene = new Scene (); |
| Scene.ClientUnitLengthInMillimeters = clientUnitLengthInMillimeters; |
| PositionCamera(); |
|  |
| m\_scene.RunCollisionDetection = EnableInteractionCollisions; |
| m\_scene.ResolveCollisions = true; //EnableInteractionCollisions; |
| m\_scene.AlwaysRunCollisionForHandsVsHeldObjects = true; |
|  |
| m\_scene.HandDistanceMultiplier = 1.0f; |
| m\_scene.DestroyClustersWhileNotHolding = true; |
|  |
| // Don't create update the hand from the interact dll |
| m\_scene.UseBodyCallbacks = ShowInteractionHands; |
| m\_scene.UseHoldingCallbacks = true; |
|  |
| // Start internal visualizer |
| if (ShowVisualizer) { m\_scene.OpenVisualDebgger (); } |
|  |
| m\_scene.AllowPinchWithAnyFinger = AllowPinchWithAnyFinger; |
| m\_scene.DisableHoldingOnPointingIndexFinger = DisableHoldingOnPointingIndexFinger; |
|  |
| m\_unityUtil = new UnityUtil (m\_scene); |
| m\_unityUtil.InitLeap (); |
|  |
| UnityUtil.LayerForHands = LayerForHands; |
| UnityUtil.LayerForHeldObjects = LayerForHeldObjects; |
| UnityUtil.LayerForReleasedObjects = LayerForReleasedObjects; |
|  |
| // Optional: Initializatio of holding callbacks. |
| // m\_scene.OnHoldingHoverOver += OnHoldingHovers; |
| // m\_scene.OnHoldingStarts += OnHoldingStarts; |
| m\_scene.OnHoldingUpdates += DisableHandCollisions; |
| m\_scene.OnHoldingEnds += EnableHandCollisions; |
| } |
|  |
| public void DisableHandCollisions(Holding holding) { |
| Body body = holding.Body; |
| GameObject game\_object = null; |
| if (body != null && body.IsValid()) |
| game\_object = UnityUtil.BodyMapper.FirstOrDefault(x => x.Value.BodyId.ptr == body.BodyId.ptr).Key; |
|  |
| ReferenceObject.IgnoreCollisionsWithHands(game\_object); |
| } |
|  |
| private void EnableHandCollisions(Holding holding) { |
| Body body = holding.Body; |
| GameObject game\_object = null; |
| if (body != null && body.IsValid()) |
| game\_object = UnityUtil.BodyMapper.FirstOrDefault(x => x.Value.BodyId.ptr == body.BodyId.ptr).Key; |
|  |
| ReferenceObject.IgnoreCollisionsWithHands(game\_object, false); |
| } |
|  |
| /// <summary> |
| /// Updates Leap device position based on the ReferenceObject. Updates Leap Intearction. |
| /// </summary> |
| void FixedUpdate () { |
| PositionCamera(); |
| m\_unityUtil.StepLeap (Time.deltaTime); |
| } |
|  |
| public void PositionCamera() { |
| Transform referenceObject = transform; |
| if(referenceObject == null) { |
| HandController tmp = GameObject.FindObjectOfType<HandController>(); |
| if (tmp) referenceObject = tmp.transform; |
| } |
| if(referenceObject == null) { |
| GameObject tmp = GameObject.Find("Main Camera"); |
| if (tmp) referenceObject = tmp.transform; |
| } |
| if(referenceObject == null) { |
| return; |
| } |
|  |
| LeapTransform referenceTransform = new LeapTransform(referenceObject.position, referenceObject.rotation); |
| referenceTransform.Position = referenceObject.position; |
| referenceTransform.Rotation = referenceObject.rotation; |
|  |
| Vector3 offsetCameraToLeap = OffsetCameraToLeap \* 1.0f/clientUnitLengthInMillimeters; |
| offsetCameraToLeap = referenceObject.rotation \* offsetCameraToLeap; |
|  |
| referenceTransform.Position = referenceObject.position + offsetCameraToLeap; |
|  |
| Scene.LeapOriginInClient = referenceTransform; |
| } |
|  |
| Leap.Interact.Scene m\_scene; |
| Leap.Interact.UnityUtil m\_unityUtil; |
| } |

Setup parameter of InteractionSceneSetup

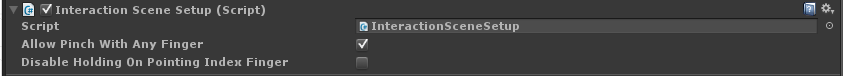
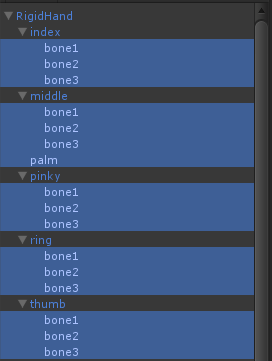


Figure 4‑15: Parameter of InteractionSceneSetup

Decrease mass variable of every bone and palm of physical hand less than other intractable objects



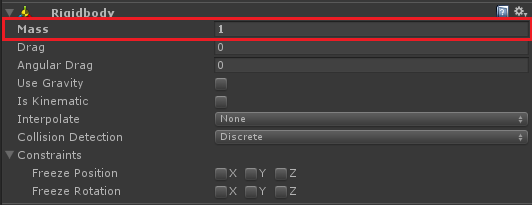
****

Figure 4‑16: Rigid body component of “RigidHand”

Create the object that we want to grab it and then attach UNITY collider and UNITY rigid body component to the object. We have to fine tune the appropriate value for rigid body. We set mass of every object in scene greater than mass of each bone in hand. The reason is that we don’t want the object bouncing off too far when colliding with many fingers at the same time. Drag and angular drag set to high value. So it will slow down by amount of air resistance when the object get bounced and enable simulate gravity system to object.

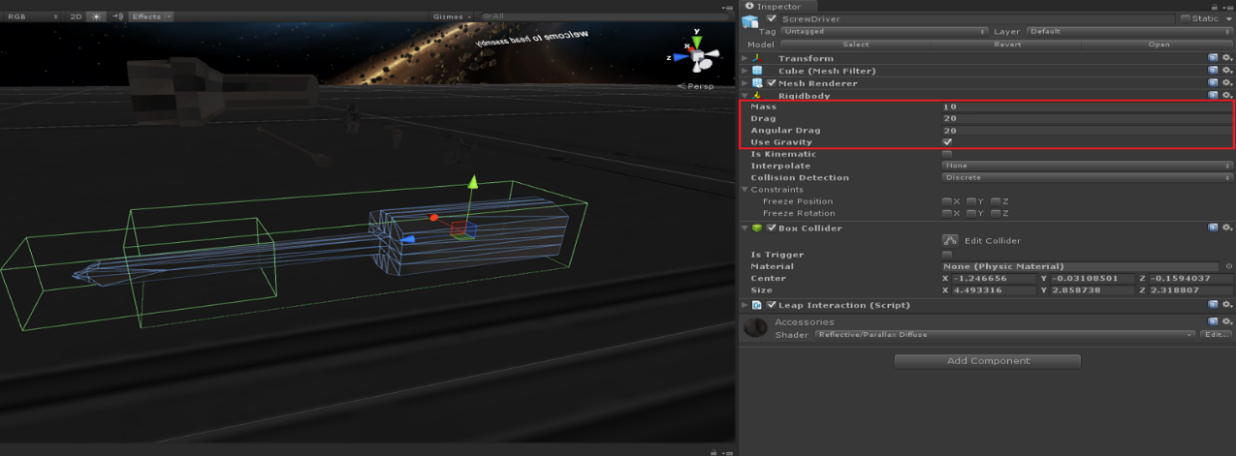


Figure 4‑17: Rigid Body Parameter

Attach LeapInteraction.cs to object.

|  |
| --- |
| using UnityEngine; |
| using System.Collections; |
|  |
| namespace Leap.Interact |
| { |
| [RequireComponent(typeof(Rigidbody))] |
| [RequireComponent(typeof(Collider))] |
| public class LeapInteraction : MonoBehaviour { |
|  |
| public enum HandlingModeEnum { |
| Fixed, |
| RotateWithOtherHand, |
| Rotate, |
| RotateAndScale, |
| Averaged |
| } |
|  |
| public HandlingModeEnum HandlingMode = HandlingModeEnum.Rotate; |
|  |
| [HideInInspector] |
| public bool LockRotation = true; |
|  |
| // Use predefined holding orientation. |
| public bool PredefinedRotation = true; |
|  |
| // Maximum distance for magnetic grabbing. |
| public float MagneticDistance = 50.0f; // default in millimeters ? // units ? // tested |
|  |
| // Determines how strongly the object is pulled towards the hodling hand. Any value. |
| public float StrengthFactor = 1.0f; |
|  |
| // Leap-space distance for starting a pinch. |
| public float MinPinchDistnace = 70.0f; |
|  |
| // Leap-space distance for maximum strength on a pinch. |
| public float FullPinchDistance = 40.0f; |
|  |
|  |
| [HideInInspector] |
| public bool EnableCollisionShapeHolding = false; |
| [HideInInspector] |
| public bool EnableAnchorBasedHolding = true; |
|  |
| // Use palm's positon to position the held body. |
| [HideInInspector] |
| public bool PositionFromPalm = false; |
| // Use palm's rotation to orient the held body. |
| [HideInInspector] |
| public bool RotationFromPalm = true; |
|  |
|  |
| // Enable rotation while holding with two hands. |
| [HideInInspector] |
| public bool EnableAnchorRotation = false; |
| // Enable scaling while holding with two hands. |
| [HideInInspector] |
| public bool EnableScaling = false; |
| // Enable reorientation on hand collision while holdling. |
| [HideInInspector] |
| public bool EnableHandCollision = false; |
| // Average transforms on multi-hand holding |
| [HideInInspector] |
| public bool AverageTransforms = false; |
|  |
| // Enable extra rotation from relative movement of holding fingers. |
| public bool RotateWithFingers = false; |
|  |
| // Not applicable to averaged holding. |
| // Set newer holdings as primary holidings (where the held body is attached). |
| public bool ToggleMainHolding = false; |
|  |
| [HideInInspector] |
| public bool EnableClicking = true; // not exposed |
|  |
| [HideInInspector] |
| public bool UseVelocity = false; // |
|  |
| [HideInInspector] |
| public bool GenerateDefaultClickAnchors = false; |
|  |
| // Always enable smooth grabbing |
| [HideInInspector] |
| public bool EnableSmoothGrabbing = true; |
|  |
|  |
| // Rotational constraint is only going to be enforced at this strength (0-1 scale). |
| [HideInInspector] |
| public float MinStrengthToLockRotation = 0.0f; |
|  |
| [HideInInspector] |
| public bool CollisionsWithHandFilteredOut = false; |
|  |
| public Body.HandAnchorType HandAnchorType = Body.HandAnchorType.DefaultHoldingAnchor; |
|  |
| public bool GenerateAnchors = false; |
|  |
|  |
| // Temp per-body values used by the UnityUtil or Scene scripts. |
| [HideInInspector] |
| public Vector3 tmpVelocity = Vector3.zero; |
| [HideInInspector] |
| public Vector3 tmpAngularVelocity = Vector3.zero; |
| [HideInInspector] |
| public bool velocityToTransfer = false; |
| [HideInInspector] |
| public float scale = 1.0f; |
|  |
|  |
| private bool m\_started = false; |
|  |
| public void ApplyToBody(Body body) |
| { |
| switch(HandlingMode) |
| { |
| case HandlingModeEnum.Fixed: |
| EnableAnchorRotation = false; |
| EnableScaling = false; |
| EnableHandCollision = false; |
| AverageTransforms = false; |
| break; |
| case HandlingModeEnum.RotateWithOtherHand: |
| EnableAnchorRotation = false; |
| EnableScaling = false; |
| EnableHandCollision = true; |
| AverageTransforms = false; |
| break; |
| case HandlingModeEnum.Rotate: |
| EnableAnchorRotation = true; |
| EnableScaling = false; |
| EnableHandCollision = true; |
| AverageTransforms = false; |
| break; |
| case HandlingModeEnum.RotateAndScale: |
| EnableAnchorRotation = true; |
| EnableScaling = true; |
| EnableHandCollision = true; |
| AverageTransforms = false; |
| break; |
| case HandlingModeEnum.Averaged: |
| EnableAnchorRotation = false; |
| EnableScaling = false; |
| EnableHandCollision = true; |
| AverageTransforms = true; |
| break; |
| } |
|  |
| MinStrengthToLockRotation = LockRotation ? 0.0f : 1.0f; |
|  |
| body.MotionType = Body.MotionTypeEnum.Dynamic; |
|  |
| body.EnableCollisionShapeHolding = EnableCollisionShapeHolding; |
| body.EnableAnchorBasedHolding = EnableAnchorBasedHolding; |
|  |
| body.UsePalmPositionForHoldings = PositionFromPalm; |
| body.UsePalmRotationForHoldings = RotationFromPalm; |
|  |
| body.EnableReorientationOnMultiHolding = EnableAnchorRotation; |
| body.EnableScalingOnMultiHolding = EnableScaling; |
| body.EnableHandCollisionWhileHolding = EnableHandCollision; |
| body.EnableRotationWithFingersWhileHolding = RotateWithFingers; |
| body.OvertakeControlWithNewerHoldings = ToggleMainHolding; |
|  |
| body.EnableAveragingTransformsOnMultiHolding = AverageTransforms; |
|  |
| body.MaxMagneticGrabDistance = MagneticDistance; |
|  |
| body.EnableClicking = EnableClicking; |
|  |
| if (GenerateAnchors) { body.GenerateDefaultAnchors(); } |
| if (GenerateDefaultClickAnchors) { body.GenerateDefaultClickAnchors(); } |
| if (EnableSmoothGrabbing) { body.EnableSmoothGrabbing(); } |
|  |
| body.UseCurrentRelativeRotationWhenLockingRotation = !PredefinedRotation; |
| body.LockRotationAboveThisStrength = MinStrengthToLockRotation; |
|  |
| body.PinchDistanceThresholdForMinStrength = MinPinchDistnace; |
| body.PinchDistanceThresholdForMaxStrength = FullPinchDistance; |
|  |
| body.SetPalmAnchor(HandAnchorType); |
|  |
| body.SetMagneticStrength(StrengthFactor); |
| } |
|  |
| void Start () { |
| InteractionSceneSetup.EnsureInstanceInitialized(); |
| AddRemoveBodyUtil.Instance.AddBodyToLeapFromUnity(rigidbody); |
| m\_started = true; |
| } |
|  |
| void Update () { |
| } |
|  |
| void OnEnable() { |
| } |
|  |
| void OnDisable() { |
| if (m\_started) |
| AddRemoveBodyUtil.Instance.RemoveBodyFromLeap(rigidbody); |
| } |
| } |
| } |

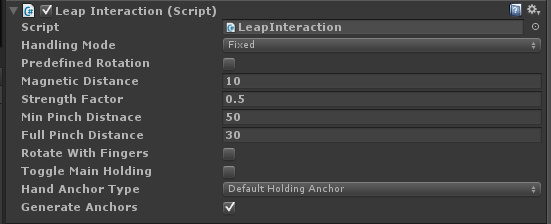
Setup parameter of Leap Interaction

Figure 4‑18: Leap Interaction Parameter

### Handle the object out of scene

Create 6 Objects at different direction. (Front, back, right, left, up, down)

Attach Box Collider component to all of them.

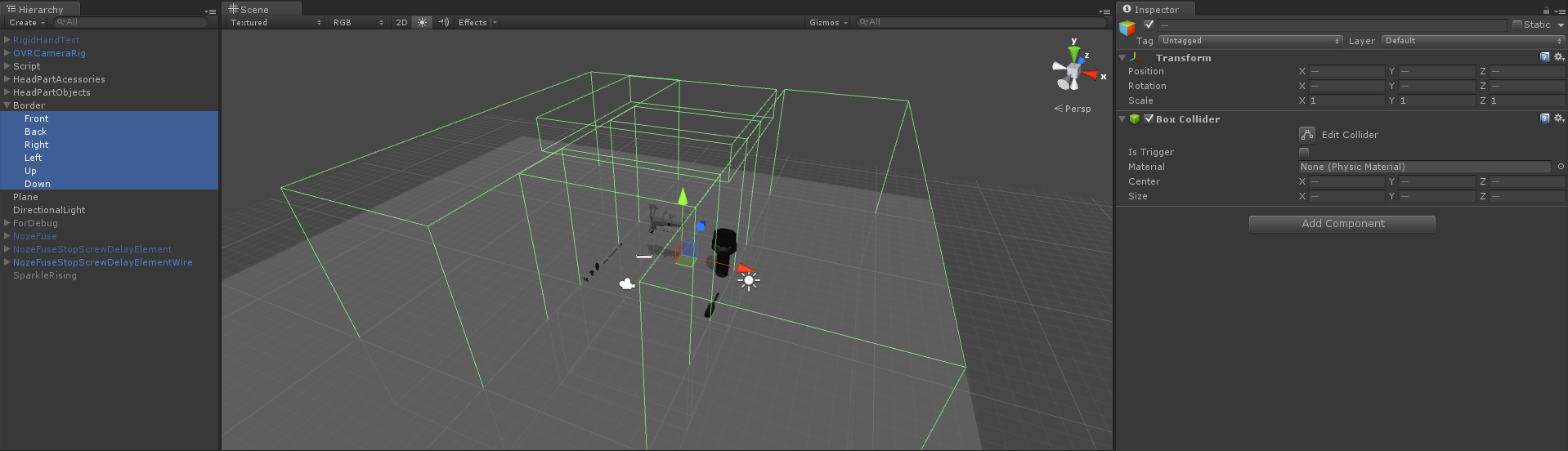
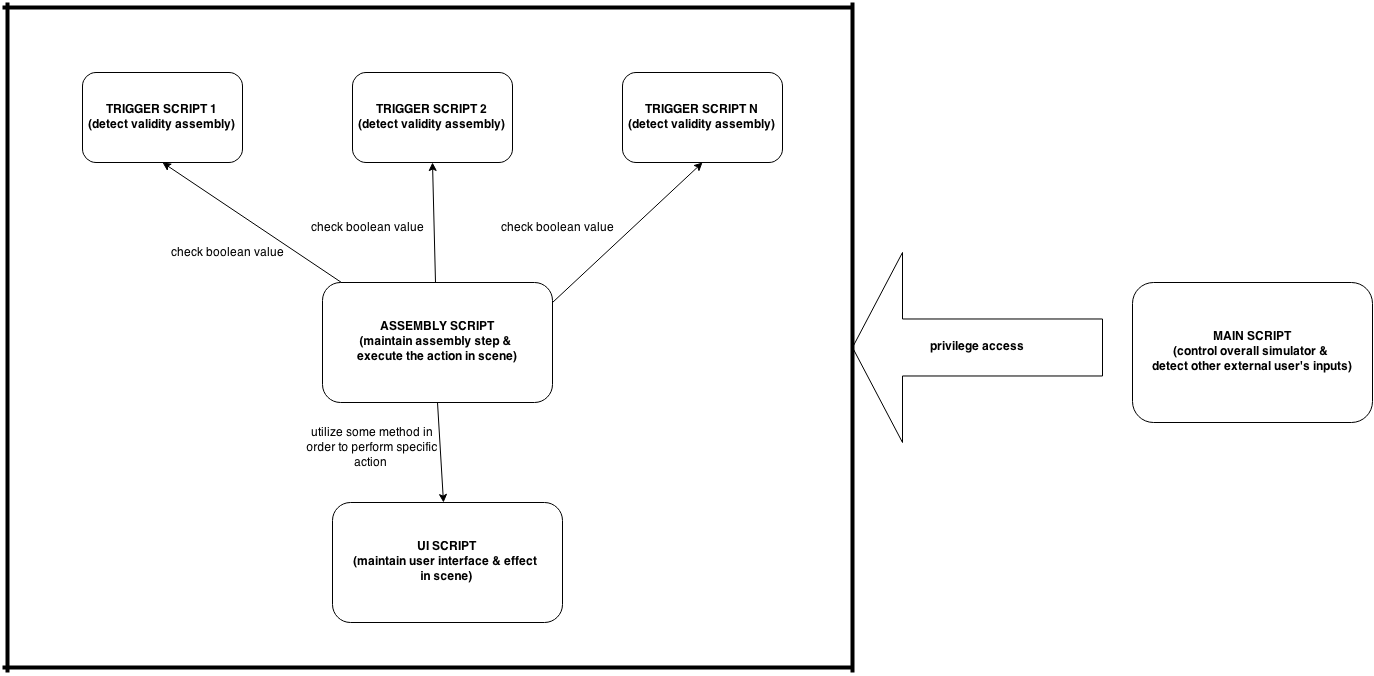


Figure 4‑19: 6 Objects at different direction with Box Collider

### Script Flow

We have many Trigger scripts which be active at its own assembly step.

When, trigger script detects the correct assembly, it will change its Boolean value. Assembly script automatically knows, when the Boolean was changed, and performs specific action regarding current step. In the performing specific action, Assembly script may invoke the method of UI script. We have the Main script which detect other user input such as: hand gestures, keyboard pressed and etc., to perform specific action. Main script also has privilege to access all other script.

Figure 4‑20: Script Flow Chart

### How to detect validity assembly

This example code shows how to detect correct assembly and change its Boolean value.

|  |
| --- |
| *TriggerDelayElement.cs* |
|  |
| public class TriggerDelayElement : MonoBehaviour |
| { |
| [HideInInspector] |
| public bool DelayElementConnected; |
|  |
|  |
| void Start () |
| { |
| DelayElementConnected = false; |
| } |
|  |
|  |
| void OnTriggerEnter(Collider other) |
| { |
| if(other.gameObject.name=="ColliderOfDelayElement"){ |
| Debug.Log ("TriggerDelayElement.cs\_HeadAssembly.Step2.DelayElementConnected"); |
| DelayElementConnected = true; |
| } |
| } |
| } |

Attach specific trigger script to the object in scene which you want to check the validity. Also add Collider component to that object and check Is Trigger.

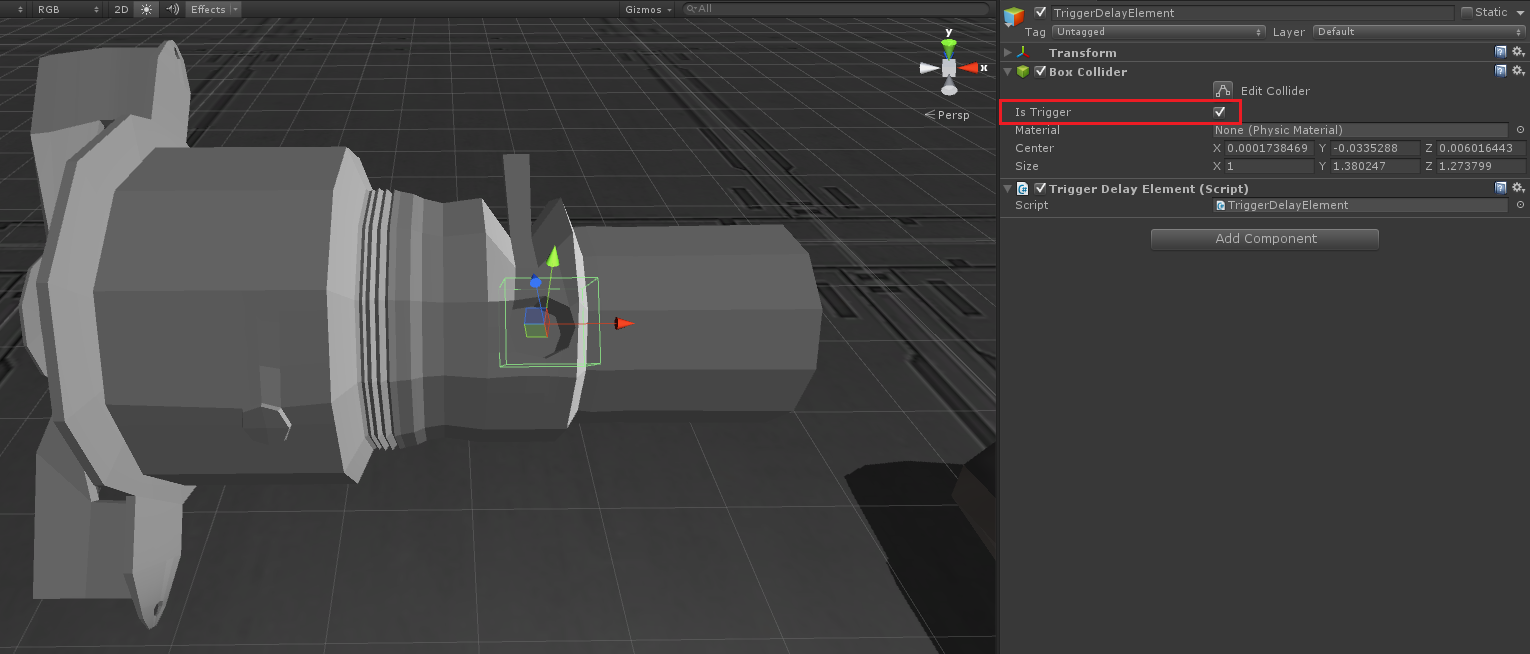


Figure 4‑21: Set up trigger object

### Perform Action after Detected Validity

This example code shows how to check Boolean of each trigger script and executes some action regarding current step such as: delete object, activate object, change step, display hint panel etc.

|  |
| --- |
| *HeadAssembly.cs* |
| void Start() |
| { |
| CurrentStep = 1; |
| CurrentMainObject = NozeFuse; |
| TriggerStopScrew.SetActive(true); // Active 1st triggered in HeadAssembly.scene |
| } |
|  |
|  |
| void Update() |
| { |
| switch(CurrentStep){ |
| case 1 : Step1(); break; |
| case 2 : Step2(); break; |
| case 3 : Step3(); break; |
| case 4 : Step4(); break; |
| case 5 : Step5(); break; |
| } |
| } |
| void Step1() |
| { |
| // If trigger sub1, perform something........ |
| if(TriggerStopScrew.GetComponent<TriggerStopScrew>().StopScrewConnected){ |
|  |
| // First, set boolean to false, for preventing these statements run again. |
| TriggerStopScrew.GetComponent<TriggerStopScrew>().StopScrewConnected = false; |
|  |
| StopScrew.SetActive(false); |
| TriggerStopScrew.SetActive(false); |
| StopScrewClone1.SetActive(true); |
| } |
|  |
| // If trigger sub2, perform something........ |
| if(StopScrewClone1.GetComponent<TriggerScrewDriver>().ScrewDriverConnected){ |
|  |
| StopScrewClone1.GetComponent<TriggerScrewDriver>().ScrewDriverConnected = false; |
|  |
| StopScrewClone1.transform.GetChild(0).gameObject.SetActive(false); // deactive highlight particle |
| ScrewDriver.SetActive(false); |
| ScrewDriverClone1.SetActive(true); |
| HeadMain.EnableCheckGesture = true; |
| } |
|  |
| // If trigger sub3, perform something........ |
| if(TriggerScreenTap.AlreadyScreenTapAtScrewDriverClone1){ |
|  |
| TriggerScreenTap.AlreadyScreenTapAtScrewDriverClone1 = false; |
|  |
| HeadMain.EnableCheckGesture = false; |
| ScrewDriverClone1.transform.GetChild(0).gameObject.SetActive(false); |
| StartCoroutine(RotateObject(ScrewDriverClone1, Vector3.right, 3.0f, 3.0f)); |
| // Put next 3 statements in BruteForceStep1\_3(), because we want those to run after rotate coroutine |
| Debug.Log ("HeadAssembly.cs\_Invoke.BruteForceStep1\_3()"); |
| // ScrewDriverClone1.SetActive(false); |
| // StopScrewClone1.SetActive(false); |
| // StopScrewClone2.SetActive(true); |
|  |
| // We don't setup new main object in the transition of step1 - step2. |
| CurrentStep++; |
| Debug.Log ("HeadAssembly.cs\_Finish.Step1"); |
| } |
| } |
|  |
|  |
| void Step2() |
| { |
| // If trigger sub1, perform something........ |
| if(Lockpin.GetComponent<TriggerHover>().HoverAtLockpin){ |
| Lockpin.transform.GetChild(0).gameObject.SetActive(false); |
| TriggerDelayElement.SetActive(true); |
| }else{ |
| Lockpin.transform.GetChild(0).gameObject.SetActive(true); |
| TriggerDelayElement.SetActive(false); |
| } |
|  |
| // If trigger sub2, perform something........ |
| if(TriggerDelayElement.GetComponent<TriggerDelayElement>().DelayElementConnected){ |
|  |
| // First, set boolean to false, for preventing these statements run again. |
| Lockpin.GetComponent<TriggerHover>().HoverAtLockpin = false; |
| TriggerDelayElement.GetComponent<TriggerDelayElement>().DelayElementConnected = false; |
|  |
| Lockpin.GetComponent<BoxCollider>().enabled = false; |
| Lockpin.transform.GetChild(0).gameObject.SetActive(false); |
| TriggerDelayElement.SetActive(false); |
| DelayElement.SetActive (false); |
|  |
| // Setup next step(transition of step2 - step3) |
| SetNewMainObject(NozeFuseStopScrewDelayElement); |
| TriggerArmingWire.SetActive(true); |
| CurrentStep++; |
| Debug.Log ("HeadAssembly.cs\_Finish.Step2"); |
| } |
| } |
|  |
|  |
| void Step3() |
| { |
| if(TriggerArmingWire.GetComponent<TriggerArmingWire>().ArmingWireConnected){ |
|  |
| // First, set boolean to false, for preventing these statements run again. |
| TriggerArmingWire.GetComponent<TriggerArmingWire>().ArmingWireConnected = false; |
|  |
| TriggerArmingWire.SetActive(false); |
| ArmingWire.SetActive(false); |
|  |
| // Setup next step(transition of step3 - step4) |
| SetNewMainObject (NozeFuseStopScrewDelayElementWire); |
| TriggerRetainingClip.SetActive(true); |
| CurrentStep++; |
| Debug.Log ("HeadAssembly.cs\_Finish.Step3"); |
| } |
| } |

Attach assembly script to empty object in scene and completely direct the reference.

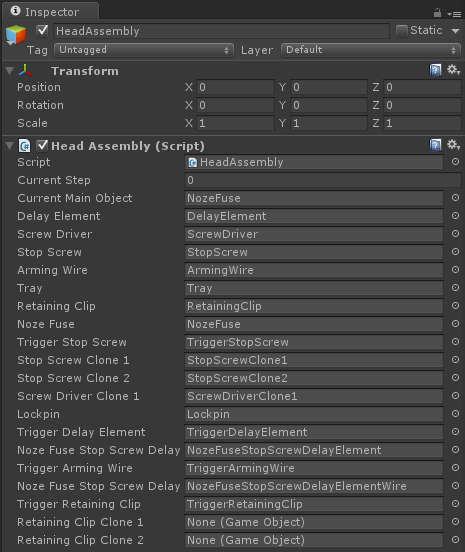


Figure 4‑22: Assembly script

### Check Hand Gestures

This code shows how to enable leap motion gesture and detect gesture.

|  |
| --- |
| *HeadMain.cs* |
| *void Start()* |
| *{* |
| *Debug.Log ("HeadMain.cs\_Initialize.HeadAsemblyScene");* |
| *EnableCheckGesture = false;* |
| *LeapMotion = new Controller();* |
|  |
| *LeapMotion.EnableGesture(Gesture.GestureType.TYPE\_SCREEN\_TAP);* |
| *LeapMotion.Config.SetFloat("Gesture.ScreenTap.MinForwardVelocity", 20.0f); // default = 30.0f* |
| *LeapMotion.Config.SetFloat("Gesture.ScreenTap.HistorySeconds", 0.5f);* |
| *LeapMotion.Config.SetFloat("Gesture.ScreenTap.MinDistance", 0.5f); // default = 1.0f* |
|  |
| *LeapMotion.Config.Save();* |
| *}* |
| *// Check if there is any screen tap gesture existed on screen* |
| *void CheckScreenTap()* |
| *{* |
| *if(EnableCheckGesture)* |
| *{* |
| *GestureList GL = LeapMotion.Frame ().Gestures();* |
|  |
| *for(int i=0; i<GL.Count; i++)* |
| *{* |
| *if(GL[i].IsValid && GL[i].Type == Gesture.GestureType.TYPE\_SCREEN\_TAP && GL[i].State == Gesture.GestureState.STATE\_STOP && new ScreenTapGesture(GL[i]).Progress == 1.0f)* |
| *{* |
| *Debug.Log ("HeadMain.cs\_Detected.ScreenTapGesture");* |
|  |
| *if(TriggerScreenTap.HandAtScrewDriverClone1){* |
| *TriggerScreenTap.HandAtScrewDriverClone1 = false;* |
| *Debug.Log ("HeadMain.cs\_Detected.ScreenTapGesture.ScrewDriverClone1");* |
| *TriggerScreenTap.AlreadyScreenTapAtScrewDriverClone1 = true;* |
| *break;* |
| *}* |
| *}* |
| *}* |
| *}* |
| *}* |
|  |

Attach this script to empty object in scene and completely direct the reference.

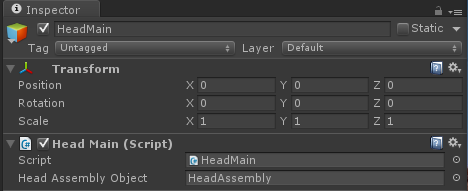


Figure 4‑23: Inspector

### Check Keyboard pressed

This code shows how to detect keyboard pressed and executes some action.

|  |
| --- |
|  |
| *HeadMain.cs* |
| // Move main object in scene by pressing 2,4,6,8,+,- |
| void CheckMoveObject() |
| { |
| if(Input.GetKeyDown(KeyCode.KeypadPlus)){ |
| Debug.Log ("HeadMain.cs\_KeyDown.KeypadPlus"); |
|  |
| // acces to other gameobject's script, for using methods |
| HeadAssembly TempScript = (HeadAssembly)HeadAssemblyObject.GetComponent(typeof(HeadAssembly)); |
| Vector3 NewPosition = TempScript.GetObjectPosition(); |
| NewPosition.z -= 0.5f; |
|  |
| TempScript.MoveObjectTo(NewPosition); |
| } |
| if(Input.GetKeyDown(KeyCode.KeypadMinus)){ |
| Debug.Log ("HeadMain.cs\_KeyDown.KeypadMinus"); |
| HeadAssembly TempScript = (HeadAssembly)HeadAssemblyObject.GetComponent(typeof(HeadAssembly)); |
| Vector3 NewPosition = TempScript.GetObjectPosition(); |
| NewPosition.z += 0.5f; |
| TempScript.MoveObjectTo(NewPosition); |
| } |
| if(Input.GetKeyDown(KeyCode.Keypad4)){ |
| Debug.Log ("HeadMain.cs\_KeyDown.Keypad4"); |
| HeadAssembly TempScript = (HeadAssembly)HeadAssemblyObject.GetComponent(typeof(HeadAssembly)); |
| Vector3 NewPosition = TempScript.GetObjectPosition(); |
| NewPosition.x -= 0.5f; |
| TempScript.MoveObjectTo(NewPosition); |
| } |
| if(Input.GetKeyDown(KeyCode.Keypad6)){ |
| Debug.Log ("HeadMain.cs\_KeyDown.Keypad6"); |
| HeadAssembly TempScript = (HeadAssembly)HeadAssemblyObject.GetComponent(typeof(HeadAssembly)); |
| Vector3 NewPosition = TempScript.GetObjectPosition(); |
| NewPosition.x += 0.5f; |
| TempScript.MoveObjectTo(NewPosition); |
| } |
| if(Input.GetKeyDown(KeyCode.Keypad8)){ |
| Debug.Log ("HeadMain.cs\_KeyDown.Keypad8"); |
| HeadAssembly TempScript = (HeadAssembly)HeadAssemblyObject.GetComponent(typeof(HeadAssembly)); |
| Vector3 NewPosition = TempScript.GetObjectPosition(); |
| NewPosition.y += 0.5f; |
| TempScript.MoveObjectTo(NewPosition); |
| } |
| if(Input.GetKeyDown(KeyCode.Keypad2)){ |
| Debug.Log ("HeadMain.cs\_KeyDown.Keypad2"); |
| HeadAssembly TempScript = (HeadAssembly)HeadAssemblyObject.GetComponent(typeof(HeadAssembly)); |
| Vector3 NewPosition = TempScript.GetObjectPosition(); |
| NewPosition.y -= 0.5f; |
| TempScript.MoveObjectTo(NewPosition); |
| } |
| } |
|  |
| // Rotate main object by pressing 5, stop by releasing 5 again |
| void CheckRotateObject() |
| { |
| if(Input.GetKeyDown(KeyCode.Keypad5)){ |
| Debug.Log ("HeadMain.cs\_KeyDown.Keypad5"); |
| HeadAssembly TempScript = (HeadAssembly)HeadAssemblyObject.GetComponent(typeof(HeadAssembly)); |
| TempScript.StartCoroutine(TempScript.RotateMainObject()); |
| } |
| if(Input.GetKeyUp(KeyCode.Keypad5)){ |
| Debug.Log ("HeadMain.cs\_KeyUp.Keypad5"); |
| HeadAssembly TempScript = (HeadAssembly)HeadAssemblyObject.GetComponent(typeof(HeadAssembly)); |
| TempScript.StopAllCoroutines(); |
| } |
| } |

### Check Hand in scene

|  |
| --- |
|  |
| *HeadMain.cs* |
| // Is it leap motion's hand? |
| public static bool IsHand(string ObjName) |
| { |
| if(ObjName.Equals ("bone1") || ObjName.Equals ("bone2") || ObjName.Equals ("bone3") || ObjName.Equals ("palm")){ |
| return true; |
| }else{ |
| return false; |
| } |
| } |

### Control highlight effect

This example code shows how to control the display time of particle effect

|  |
| --- |
| *LoopingHightlight.cs* |
| public class LoopingHighlight : MonoBehaviour |
| { |
| ParticleEmitter PE; |
| IEnumerator coroutine; |
|  |
|  |
| void OnEnable() |
| { |
| PE = this.gameObject.GetComponent<ParticleEmitter>(); |
| coroutine = Highlight(); |
| StartCoroutine(coroutine); |
| } |
|  |
|  |
| void OnDisable() |
| { |
| StopCoroutine(coroutine); |
| } |
|  |
|  |
| IEnumerator Highlight() |
| { |
| while(true){ |
| PE.emit = false; |
| yield return new WaitForSeconds(3.0f); |
|  |
| PE.emit = true; |
| yield return new WaitForSeconds(0.75f); |
| } |
| } |
| } |

Also attach this script to particle object.

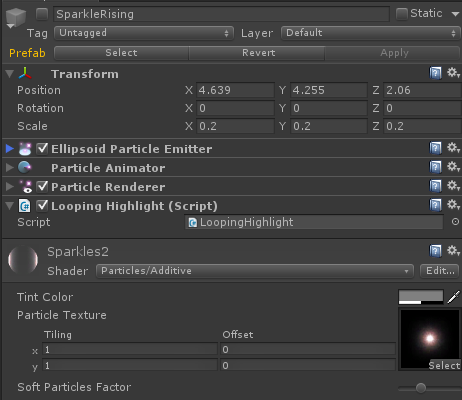


Figure 4‑24: Particle object panel

### Highlight held object

In the simulator, when user is hovering or grabbing the intractable object.

The object will change the material and simulator will display object’s name on the screen.

Simply attach “HoldingCallback.cs” script to “HandController”. This code shows how to change material of hovering or held object.

|  |
| --- |
| *HoldingCallbacks.cs* |
|  |
| using System.Collections; |
| using System.Collections.Generic; |
| using System.Linq; |
| using Leap.Interact; |
|  |
| public class HoldingCallbacks : MonoBehaviour |
| { |
|  |
| public Material freeMaterial; |
| public Material hoverMaterial; |
| public Material heldMaterial; |
|  |
| private GameObject lastHovered = null; |
|  |
| void Start () { |
| UnityUtil.Scene.OnHoldingHoverOver += new Scene.HoldingNotification(OnHoldingHovers); |
| UnityUtil.Scene.OnHoldingStarts += new Scene.HoldingNotification(OnHoldingStarts); |
| UnityUtil.Scene.OnHoldingUpdates += new Scene.HoldingNotification(OnHoldingUpdates); |
| UnityUtil.Scene.OnHoldingEnds += new Scene.HoldingNotification(OnHoldingEnds); |
| } |
|  |
| void Update () { |
| } |
|  |
| public void OnHoldingHovers(Holding holding) { |
| Body body = holding.Body; |
| GameObject gameObject = null; |
| if (body != null && body.IsValid()) |
| gameObject = UnityUtil.BodyMapper.FirstOrDefault(x => x.Value.BodyId.ptr == body.BodyId.ptr).Key; |
| if (lastHovered != gameObject) |
| { |
| if (lastHovered) |
| lastHovered.renderer.material = freeMaterial; |
| if (gameObject) |
| gameObject.renderer.material = hoverMaterial; |
| lastHovered = gameObject; |
| } |
| } |
|  |
| public void OnHoldingStarts(Holding holding) { |
| Body body = holding.Body; |
| GameObject gameObject = null; |
| if (body != null && body.IsValid()) |
| gameObject = UnityUtil.BodyMapper.FirstOrDefault(x => x.Value.BodyId.ptr == body.BodyId.ptr).Key; |
| //Debug.Log("holding started " + gameObject); |
| if (gameObject) |
| gameObject.renderer.material = heldMaterial; |
| } |
|  |
| public void OnHoldingUpdates(Holding holding) { |
| } |
|  |
| public void OnHoldingEnds(Holding holding) { |
| Body body = holding.Body; |
| GameObject gameObject = null; |
| if (body != null && body.IsValid()) |
| gameObject = UnityUtil.BodyMapper.FirstOrDefault(x => x.Value.BodyId.ptr == body.BodyId.ptr).Key; |
| if (gameObject) |
| gameObject.renderer.material = freeMaterial; |
| } |
| } |

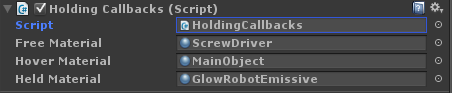
Set material reference.

Figure 4‑25: Holding Callbacks Script

### Change graphic hand model

We attach HandCycle.cs script at HandController to maintain hand graphic model and also allow user to change hand graphic model by pressing left and right keyboard arrow.

|  |
| --- |
| *HandCycle.cs* |
|  |
| *using UnityEngine;* |
| *using System.Collections;* |
|  |
| *public class HandCycler : MonoBehaviour {* |
|  |
| *public HandModel[] leftHands;* |
| *public HandModel[] rightHands;* |
|  |
| *private int hand\_index\_ = 0;* |
|  |
| *void Start() {* |
| *SetNewHands();* |
| *}* |
|  |
| *protected void SetNewHands() {* |
| *HandController controller = GetComponent<HandController>();* |
| *controller.leftGraphicsModel = leftHands[hand\_index\_];* |
| *controller.rightGraphicsModel = rightHands[hand\_index\_];* |
| *controller.DestroyAllHands();* |
| *}* |
|  |
| *void OnGUI() {* |
| *Event current\_event = Event.current;* |
| *if (current\_event.type == EventType.KeyDown) {* |
| *if (current\_event.keyCode == KeyCode.LeftArrow) {* |
| *hand\_index\_ = (hand\_index\_ + leftHands.Length - 1) % leftHands.Length;* |
| *SetNewHands();* |
| *}* |
| *else if (current\_event.keyCode == KeyCode.RightArrow) {* |
| *hand\_index\_ = (hand\_index\_ + 1) % leftHands.Length;* |
| *SetNewHands();* |
| *}* |
| *}* |
| *}* |
| *}* |

Also attach this script to HandController object and set hand model reference, as much as you want.

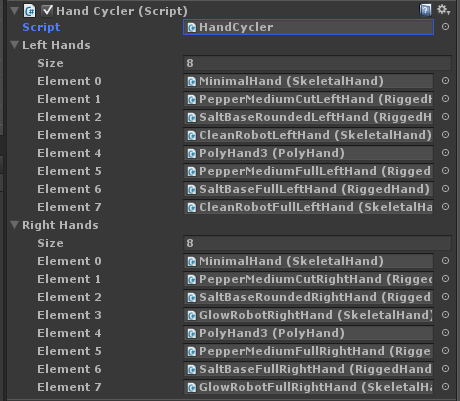


Figure 4‑26: All hand model script

CHAPTER 5

# Testing and Evaluation

This chapter explains the testing method and evaluation results of VRBomb also including the charts of the results.

## Testing Methods

* Implement the questionnaires relating on the simulator.
* Every participants have to try the simulator by him/herself before doing the questionnaires.
* All participants are the guest in ACDT2015 event.
* Number of participants are 15.

### Development Testing

We tested our project from the head part step by step until the tail part.

#### Head Part

* Have problem with the gravity of model, because some model such as arming wire is too small but the tracking hand is too big then it hard to manipulation and object always rebound.

#### Tail Part

* Have problem with the gravity of model, because some model such as arming wire is too small but the tracking hand is too big then it hard to manipulation and object always rebound.

### Users Testing

We making the questionnaire in assessment form. In the form include age gender occupation and 7 questions of Basic Virtual reality and our simulation, it’s let the participants check the symbol in the box that have 3 levels (Bad, Fair and Good) for each question. Our Target is the guest in ACDT2015 event at Hyatt Regency Hua Hin Hotel on 23-24th May 2015, every guest have to try the simulation by themselves before doing the evaluation and the number of target for our project is 15 persons.



Figure 5‑1: Usability Testing at ACDT2015 event

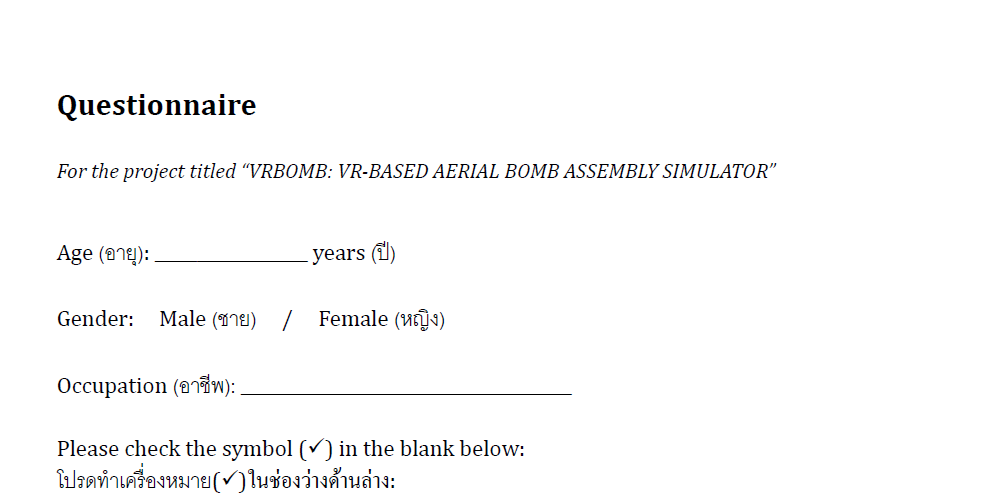
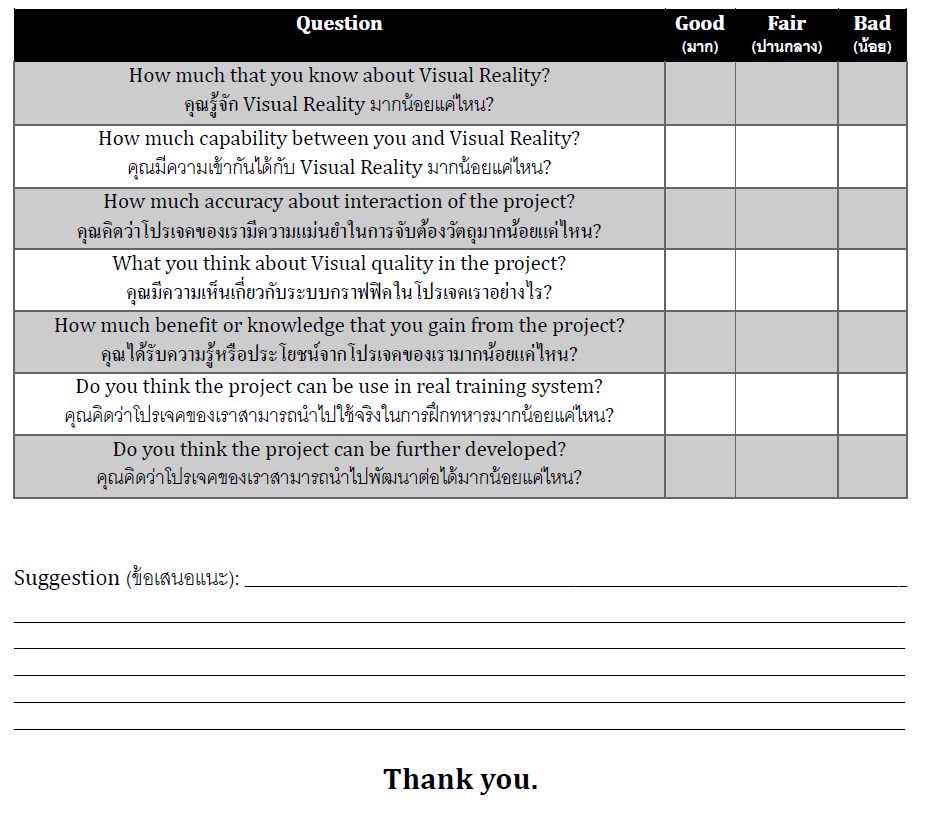
* Evaluation Form

Figure 5‑2: Evaluation Form

## Evaluation Results

Figure 5‑3: Graph of Age

This graph show the percent of age of the guest in the event who interested in our project, try simulation and made the evaluation for us. Both age from 10 to 25 and 26 to 40 is 40 percent and 41 to 55 is only 20 percent.

Figure 5‑4: Graph of Gender

This graph show the percent of gender of the guest in event, Male is 60 percent and following by the female in 40 percent.

Figure 5‑5: Graph of Occupation

This graph show the percent of occupation of the guest in event and the most occupation is student from many university by 47% and the least occupation is Engineer and Instructor in 13 percent.

Figure 5‑6: Graph of Question 1

This graph show the first question of the evaluation “Did the guest know what is VR stand for” and the result is 67% is fair, 27% is good and 6% is bad.

Figure 5‑7: Graph of Question 2

This graph show the second question of the evaluation “how are they familiar with VR device”. The result is 60% is fair. 27% is good and 13% is bad.

Figure 5‑8: Graph of Question 3

This graph show the third question of the evaluation “how much accuracy in interaction with object in our program”. The result is 60% are fair. 33% is good and 7% is bad.

Figure 5‑9: Graph of Question 4

This graph show the fourth question of the evaluation “how are they think about graphic in the simulator”. The result is 53% is good, 47% is fair and bad is none.

Figure 5‑10: Graph of Question 5

This graph show the fifth question of the evaluation “How much benefit or knowledge that you gain from the simulator”. The result is 60% is good, 40% is fair and bad is none.

Figure 5‑11: Graph of Question 6

This graph show the sixth question of the evaluation “Do you think the project can be use in real training system?” The result is 80% is good, 20% is fair and bad is none.

Figure 5‑12: Graph of Question 7

This graph show the last question of the evaluation “Do you think the project can be further developed?” The result is very positive is 100 % is good.

Feedback

* Let graphic look more interesting
* Good job!
* Recondition about device interaction
* Very fun and exciting!
* Grabbing object is hard
* Make not only assembly the MK-85 such as boxing game?

CHAPTER 6

# Conclusions

This chapter will explain about the conclusions of this project including benefits, problems and future work.

## Benefits

This project can give benefit to both developer and the user.

### Benefits to Project Developers

* We have learned how to create the virtual reality simulator.
* We know how to integrate the virtual reality SDK into the project.
* We learn how to work as a team.
* We learn the architectures and how to assembly of MK80 bomb series..
* We gain more duty consciousness to finish our work on time and keep the work flow up to schedule.

### Benefits to Users

* Users can learn how to assembly MK80 bomb series.
* Users will get the experience of using the virtual reality simulator.
* User may get inspire by our project to create their own virtual reality system

## Problems and Limitations

* There are the limitation of hand tracking device hence Leap motion device that can’t track smoothly and there can get interfere very easy from the change of environment brightness or even in dusty environment also if something is in the line of sight that can refract the light, it will also cause the error in the leap motion tracking.
* There are problem with the model that import into Unity from Blender model file, some of them aren’t show the accurate model compare to the original and the texture from the blender can’t import into the Unity so we have to manually correct them again in Unity.
* The models that are very small or have many component in them are very hard to interact with due to the inaccurate tracking and sometime they can cause some error in system.
* The software developer kit for both Oculus rift and Leap motion aren’t compatible with each other and some function aren’t support by the Unity.
* There are the limitation to Leap motion that contain too few gesture that can use and some gesture has some serious bug that we can’t applied to this project

## Future Work

* To improve the user knowledge in training assembly the bomb on each part, the simulator should add the function to open-close the hint panel.
* To improve the realistic in of assembly, the program should allow the user to grabbing the main object on each part.
* This project can be used as fundamental system to develop into other simulation that consist the hand and head movement such as gun assembly or shooting simulation.
* This project can still add or change the content of the component model to the better and more accurate model also we can change the hand tracking into the better device to give more accurate tracking and better experience.

REFERENCES

1. MK82 GENERAL PURPOSE BOMB Retrieved From http://www.fas.org/man/dod-101/sys/dumb/mk82.htm

2. General-purpose bomb Retrieved From http://en.Wikipedia.Org/wiki/general-purpose\_bom

3 Virtual reality Retrieved From http://en.wikipedia.org/wiki/Virtual\_reality

4 Oculus Rift Retrieved From <http://en.wikipedia.org/wiki/Oculus_Rift>

5 Leap Motion Retrieved From <http://en.wikipedia.org/wiki/Leap_Motion>

6 MK-80 Series Bomb Information From <http://www.vrs.org.uk/virtual-reality-military/index.html>

7 Military VR Training From <http://www.pcgamer.com/us-military-commission-57m-virtual-reality-training-sim-powered-by-cryengine-3/>

8 Leap Motion API from <https://developer.leapmotion.com/documentation/java/devguide/Leap_Overview.html>

9 Body and Tail Fuze Information From <http://www.globalsecurity.org/military/>

10 Oculus Rift VR, LLC Integration Guide (SDK Version 0.4.4), Authors: Peter Giokaris, David Borel, and Date: December 2, 2014

11 Chapter 4: Bombs, General Purpose, and 500-Pound MK82/BLU-111; 2000-Pound MK84/BLU-117; Bombs, Practice, 500-Pound MK82 Inter BDU-50 Series and 2000-Pound MK84 Inert BDU-56/B, Authors: The Defense Technology Institute Documentation

12 The example related work Planetarium from http://blog.leapmotion.com/introducing-planetarium-design-science-behind-vr-widgets-showcase/

13 The example related work Rift Coaster from https://share.oculus.com/app/riftcoaster-hd

BIOGRAPHIES

|  |  |
| --- | --- |
| NAME | Miss Patboke Jitthamma |
| DATE OF BIRTH | 21 September 1992 |
| PLACE OF BIRTH | Bangkok, Thailand |
| INSTITUTIONS ATTENDED | Rajinibon School, 2010:  High School Diploma  Mahidol University, 2014 :  Bachelor of Science (ICT) |
|  |  |
| NAME | Mr. Pramoch Viriyathomrongul |
| DATE OF BIRTH | 11 October 1992 |
| PLACE OF BIRTH | Bangkok, Thailand |
| INSTITUTIONS ATTENDED | Mattayom wat benjamaborpit School, 2010:  High School Diploma  Mahidol University, 2014 :  Bachelor of Science (ICT) |
|  |  |
| NAME | Mr. Treechart Treerattanapakong |
| DATE OF BIRTH | 5 January 1993 |
| PLACE OF BIRTH | Bangkok, Thailand |
| INSTITUTIONS ATTENDED | Mathayom Satit Radjhaphat Bansomdejchaopraya School, 2010:  High School Diploma  Mahidol University, 2014 :  Bachelor of Science (ICT) |