**Department of Applied Physics**

**Final Year Project Report**

**Project Title**

**Virtual reality in physics experiment: Mechanics**

**Student Name: Wang Zhikang**

**Student Number: 13102247d**

**Programme code: 11439**

**2016 – 2017**

Abstract

As virtual reality (VR) is developing fast in recent years, researchers have been investigating its future application in different fields, especially for education. However, little work was done about physics. Therefore, this project was implemented to investigate how physics can be presented in virtual reality to make it a prospective learning tool. Through this project, a program was built to present mechanical demonstrations on Google Cardboard VR on Android middle-end smartphones, and the results and quality were evaluated and discussed. It was found generally better than conventional planar physics demonstrations, while it still had some deficiencies. Further optimization and deployment to other platforms, more content in the program, and other program design on better VR devices may be done in the future.

Acknowledgement

I would like to acknowledge the support of the Department of Applied Physics, the Hong Kong Polytechnic University, in particular, my supervisor prof. Lau Shu Ping, for his valuable advice and support. I would also like to thank my friends, students Wang Jia Jun, Huang Yu Yang, Wu Qi Lin, Wong Kin Ping and Man Yui Kuen for their help with program tests. I hereby acknowledge the use of Unreal Engine 4 and Autodesk 3ds Max for educational purposes in this project, and the collision sound effect in the developed program is from <http://www.zapsplat.com/music/snooker-or-pool-cue-ball-strikes-another-ball-reverberant/> .

INDEX

[Abstract II](#_Toc481102397)

[Acknowledgement II](#_Toc481102398)

[INDEX III](#_Toc481102399)

[CHAPTER 1 INTRODUCTION 1](#_Toc481102400)

[CHAPTER 2 LITERATURE REVIEW 2](#_Toc481102401)

[CHAPTER 3 METHODOLOGY 5](#_Toc481102402)

[3.1 General Information 5](#_Toc481102403)

[3.2 Scene Arrangement 6](#_Toc481102404)

[3.3 Asset Preparation 7](#_Toc481102405)

[3.4 Motion Control System 10](#_Toc481102406)

[3.5 Programming Language 12](#_Toc481102407)

[3.6 Program Design 14](#_Toc481102408)

[3.7 Optimization 19](#_Toc481102409)

[3.8 Packaging for Distribution 23](#_Toc481102410)

[CHAPTER 4 RESULTS 28](#_Toc481102411)

[CHAPTER 5 DISCUSSION 33](#_Toc481102412)

[CHAPTER 6 CONCLUSIONS AND RECOMMENDATIONS 36](#_Toc481102413)

[References 38](#_Toc481102414)

[Appendix 42](#_Toc481102415)

CHAPTER 1 INTRODUCTION

The virtual reality (VR) industry is developing fast in recent years. In 2016, three commercial products were released, which are HTC Vive, Oculus Rift and PlayStation VR, and this marked the first year when commercial VR end products on PCs and consoles appeared on the market. The estimated sales volume of these three are all a few hundred thousand sets in year 2016 [1]. Also, for low-end VR devices, Google Cardboard, which works on smartphones, was said to have shipped 5 million sets till January 2016 [2], occupying most of the low-end VR market.

Regarding the definition of VR, it is "a realistic and immersive simulation of a three-dimensional 360-degree environment, created using interactive software and hardware, and experienced or controlled by movement of the body" [3]. Its development is reasonable, since the display of VR, stereoscopic 3D, is just the display of two images of the same world from different viewpoints, which can be easily achieved using current 3D technology. It has a great deal of potential use in simulation and education as a tool for immersive learning, and it has already been investigated for anatomy, medical simulation, nursing, aviation training etc. [4-7], however, there are still few about physics. Therefore, in order to exploit potential use of VR on physics, especially for teaching, this project was implemented. In this project, how physics experiments can be presented in a VR setting was investigated through the development of a prototype physics demonstration program, and the result was relatively satisfactory though some flaws did exist. The motivation is that, physics learning values intuitiveness and engagement, which can be achieved by a VR environment. Much other work was done to enhance engagement and interaction, by putting concrete physical phenomena displayed on a 2D screen surface and making them interact with user’s adjustment, such as the PhET Interactive Simulations project at the University of Colorado Boulder for science and math [8]. However, it is not interactive or intuitive enough if considering that it involves many parameters as numerical values to set as initial conditions [9], and it only involves 2D shapes, which is hardly realistic. Both problems are expected to be solved by the VR technology. This report will introduce other work done by previous researchers, and will describe how the physics demonstration program was developed during this project, and will examine the result, discuss the advantages and disadvantages, and reach a conclusion.

CHAPTER 2 LITERATURE REVIEW

There were not many recent researches exploring possibilities of combining physics demonstrations and VR. In one research by Savage et al., where the name “virtual reality” was used but only meant planar physics simulation, some physical phenomena on special relativity and quantum physics were visualized and presented on computer screens in 2D to help students to understand [10]. It involved visible objects under the influence of physical phenomena to help students to develop a sense of what was happening. It reported “a small but statistically significant difference” of students’ score improvement, and has “presented evidence that three-dimensional interactive simulations can enhance students’ understanding of abstract areas of physics”, probably because it would be no longer difficult for students to imagine how the physical behaviour should be after learning from the concrete demonstrations. This provides evidence for the importance of this project.

Nevertheless, a research by Hannes Kaufmann and Bernd Meyer in 2009 is very close to the expected result of this project [11]. It indeed used a VR headset even though no commercial one was available at that time. A 3D pen was used as a controller to pick objects to place in a 3D world, and adapters configured with a customized time-variant force input could be attached to an object to influence its behaviour, and data of different properties could be recorded by attached different adapters. Objects with links and joints and different shapes could be placed in a 3D world to simulate any situation, which is like a sandbox setting, and therefore experiments could be conducted using it with adapters. However, according to the pictures shown in that paper, this system might have been too complicated to learn for students, because many different adapters were used to track or add different data, and then it might be questionable whether it could really help to learn physics efficiently. Still, another problem is present: it used PhysX as the physics engine to simulate all physical behaviours of objects. However, it is a physics engine made for games to have only convincing physical behaviour but not precise or reliable physical behaviour, and it can have a significant amount of error when operating physical experiments because it emphasizes the speed to handle a large number of objects [12]. For example, during this project, it was tested to have roughly 10% error per period for harmonic oscillation, which should not be tolerated if the physics experiment needs to be consistent and stable. Therefore, it may not be suitable for a sandbox setting, where the user should be able to place anything he/she wants and record correct data from experiments designed inside. Also, a low-precision physics engine can explode a pile of objects due to accumulated error when there are too many objects touching each other in a sandbox setting.

As 3D technology and VR technology are developing, many software companies in these fields claim that they welcome educational use, and provide privileges for educational purposes, such as Epic Games Inc., Unity Technologies Inc. and Autodesk Inc. [13-16]. However, as the software is mainly oriented to entertainments when developed, many functionalities, such as storing and drawing graphs, and many other requirements, such as a good precision, are not provided, and it is not easy to use. Therefore, this field still needs development and the software needs to be adapted to use easily. On the other hand, a precise enough physics engine still cannot be found as they are often designed to handle thousands of objects efficiently at the same time at the expense of precision [12], using a first-order Euler method. However, a precise physics engine specially for this purpose may also be built so that a free sandbox setting can be achieved. This is possible since current ones are not originally designed for precise purposes, and this may be in greater demand as VR is getting more popular and more physically stable surroundings are needed compared to current unstable ones, especially for connected systems such as ragdolls.

CHAPTER 3 METHODOLOGY

3.1 General Information

In this project, a game engine called Unreal Engine 4 was used as the platform to build a program, in the form of a packaged software, to provide an interactive 3D world that one can view on VR devices. In this virtual world, the user can choose a certain topic of physics demonstration and adjust parameters to see the corresponding physical results. As a starting point, the target VR device is one of the cheapest possible devices, Google Cardboard viewed on a smartphone, which typically costs 20 to 200 HKD [17]. This device was chosen because it might be unreasonable to suppose all users can afford or be provided with high-end VR devices of a few thousand Hong Kong dollars, especially if the users are mainly students.

The drawback of this VR device compromise is also significant. As the users are not expected to have a motion controller at hand, the input channels are limited and it is difficult for the program to have an interactive design, and freedom inside the 3D world is thus reduced, and a sandbox setting is not expected. In the program, the input channels are supposed to be one gyroscope, which should automatically work to change the view angle displayed on the smartphone when the phone is rotated, and one button on the device, which should simulate a touch action on the phone screen. This supposition generally holds true for current cheap VR devices including those products derived from Google Cardboard under names of different companies. As a result of a low degree of freedom of input channels, much content in the program is displayed in 2D graphical user interface (GUI) panels with texts, buttons and sliders, which is similar to common 2D physics demonstrations on websites, and far from ideal as a VR program. Despite these disadvantages, user control in this program was designed as intuitive as possible for easy use, and user engagement for learning is enhanced by the 3D VR view.

The computer used for this program development was HP Pavilion Notebook 14-al087tx, produced in 2016, with a low voltage two-core CPU intel i7-6500U, 8 GB DDR4-2133 RAM, a 940mx NVidia graphic card, one 128GB SSD hard drive and one 1TB SATA hard drive [18]. The tools involved in this project were Unreal Engine 4.15.1, which is free for educational purposes [13], Autodesk 3ds Max 2017, which was validated through a student license for three years [15], and GoldWave using my own personal license. To deploy this program to Android platform, the Android Software Development Kit (SDK) and related packages were also installed. Necessary details on how these were used will be mentioned later.

3.2 Scene Arrangement

The constructed 3D world in a game engine is called a scene. To construct the scene for physics demonstration in VR using Unreal Engine 4, a round static table was put at the centre of the floor; all designed physics demonstrations were displayed on it. The dimensions of the table seen in VR mode are shown in Fig. 1 (a), where 80 cm indicates the height of the table, and the white spherical point above the table indicates the user’s start position where the user camera will be placed. These dimensions were determined by trial-and-error so that the user could feel like sitting in front of a table with a typical distance of paper work or a laptop computer. The distance between the start position and the centre of the table is roughly 36.2 cm, and the eyesight to the centre is approximately 50.6 degrees downward. The main menu was put vertically at the centre above the table, and during a physics demonstration, the control panels controlling physical parameters were put horizontally on the table near the user as Fig. 1 (b) shows. These are all planar 2D user interfaces (UIs) and can be clicked by pressing the button on the VR device when they are at the centre of the user’s view. To indicate where the click action would exactly happen, a red point was also drawn at this view centre as the click location. Physics demonstrations were put at the centre of the table. When a diagram was needed to show changing physical quantities, a diagram was placed behind physics demonstrations and was vertical or tilted upward to be viewed more easily. To help motion control, two icons as buttons were put 5 cm above the centre of the table, which will be discussed later.

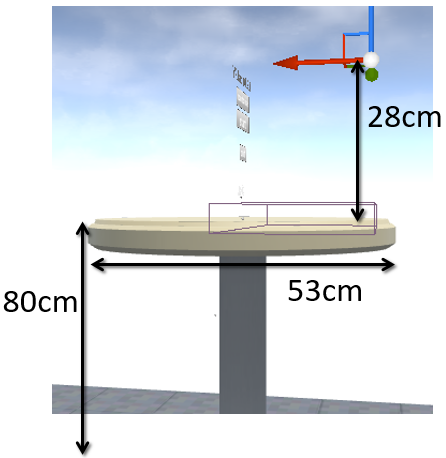


FIG. 1. Screenshots taken on the computer of the start menu from the left (a), and of a physics demonstration from the default start location (b).

(a)

(b)

3.3 Asset Preparation

The source of the table is the table static mesh from the Unreal Engine Mobile Starter Content pack and modified using Autodesk 3ds Max 2017 (3ds Max). The static mesh was exported from Unreal Engine content pack as an FBX file, imported to 3ds Max and edited, and exported as FBX again to put into Unreal Engine. During the modification, the depression on the table surface was removed and a new surface was created using the mesh editor to seam it and make it flat as shown is Fig. 2 (a); the UV map, which is the mapping from the object surfaces to a texture image file, was recreated to optimize shadow quality around the table; the curved legs were deleted and replaced by a cylinder to reduce complexity.

The slope used in a physics demonstration was created using 3ds Max. It was created by drawing a spline object of a right triangle with a 30-degree angle applied an extrude modifier. Also, a UV map was assigned and modified so that a lightmap can be generated in Unreal Engine, otherwise the object would appear black if set to receive precomputed static lighting. It was then exported to be an FBX file to import into Unreal Engine.

Two 3D arrows were created using 3ds Max from a box geometry as a shaft and a triangular prism as an arrowhead that was made from another box geometry applied a taper modifier. However, because surfaces of both the shaft and the arrowhead would have a transparency when the object was assigned as transparent, the overlapping surfaces at the connection part of the shaft and the arrowhead would be visible with a resultant different colour. This is because the shaft and the arrowhead were treated as two separate objects and their total transparency when seen through both was lower. To eliminate this artefact, the internal surfaces were deleted and this part was seamed manually. A UV map was not assigned to this object because a colour determined by static lighting was not expected on it, as it is only an indicator in demonstrations.

2D icons were created by the rendering functionality of 3ds Max to capture 2D images of 3D objects. 2D arrows were created in this method with an arrow made as above. The two icons used to control motion were from another three objects created by 3ds Max. One was a sphere surrounded by a doughnut and two triangular prisms to express the motion of orbiting as Fig.2 (d) shows. Other two were distorted arrows using a prism and a curvedly tapered box to express the motion of going forward and backward. Since both 3ds Max and Unreal Engine provide methods to adjust colour and brightness, and transparency (or, Alpha Channel) can be output in captured 2D images of 3ds Max, a 2D image editing software such as photoshop was not needed.

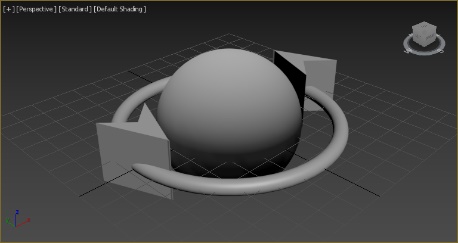
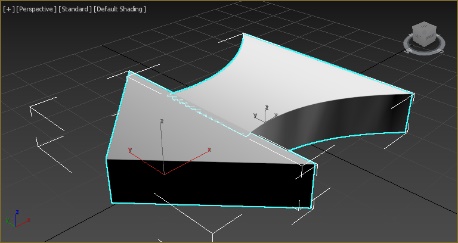
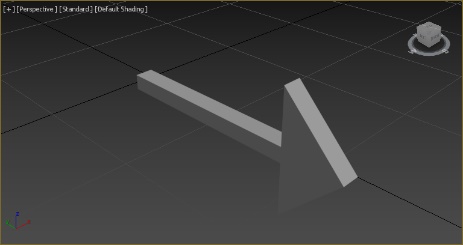
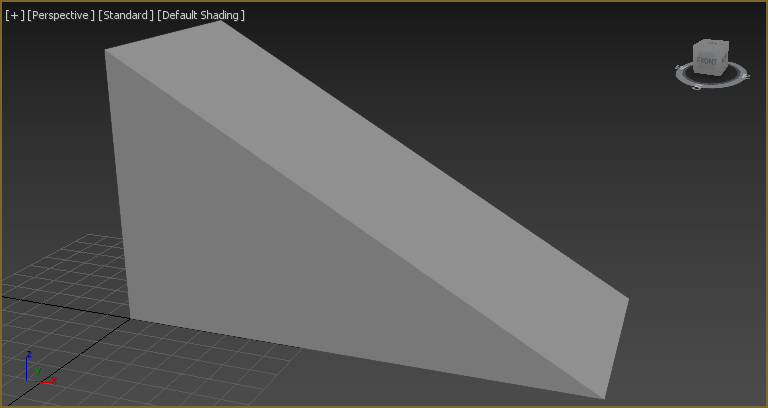
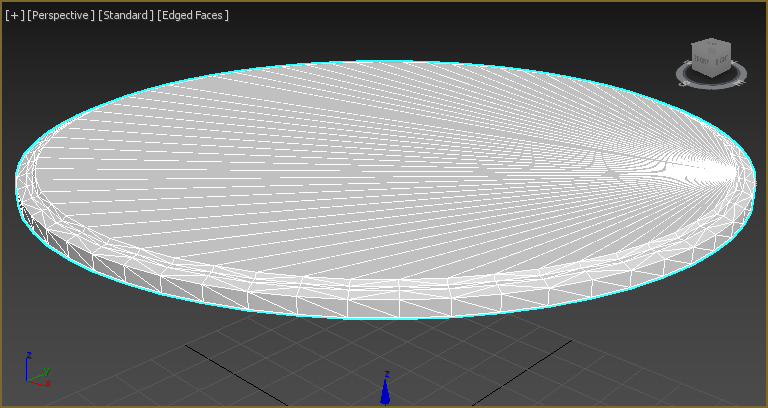


FIG. 2. Screenshots of edited objects in 3ds Max, respectively the table, the slope, an arrow, the orbiting object, and the forward arrow.

(a)

(b)

(c)

(d)

(e)

In order to better inform the user what is happening, four sound effects were added. One is from Unreal Engine editor sound assets, and was originally the sound of end test playing in Unreal Engine editor. It was then used as the sound of the viewer’s smooth automatic motion on a determined path. For example, the sound will be played when the viewer resets his/her position back to the default start position. Two sounds were used to indicate the press and release of the button on the VR device. These press and release sounds were separated using GoldWave from one whole button click sound effect obtained from <https://www.soundjay.com/button-sounds-1.html> for free. Another sound was used to indicate collision of spheres, obtained and modified from <http://www.zapsplat.com/music/snooker-or-pool-cue-ball-strikes-another-ball-reverberant/> for free with an attribution or credit requirement. The modifications silenced background noise and deleted one minor noise after the major collision sound, which was possibly an echo, and quickened its sound attenuation of reverberation. The minor noise was removed by copying another part of reverberant sound to overwrite it, adjusting the volume to make it consistent with its context, and using the functionality of removing pops and clicks to connect the sound wave naturally with before and after.

3.4 Motion Control System

Since it is a VR environment, the user is supposed to be given the ability to move around in the virtual world. Unlike high-end VR devices, usually the Google Cardboard VR does not expect the user to walk around in the real world to move in the virtual world, and when this program was designed, the user was also expected to be seated. Therefore, motion control was realized with the use of the single button on the expected VR device. After incorporating some ideas from 3D design programs, a relatively convenient control system was developed, with two buttons placed at the centre of the scene to help.

This control system provides three means of movement. The first is the most common one, which is linear motion towards the front or the back, and defaults to front. This functionality can be triggered by pressing and holding the VR device button for over 0.45 second, and the direction of movement is determined by the front direction when triggered, and this direction will not change even if the front direction of the user changes during this motion progress, so that the user can view towards any direction while moving linearly. This motion stops when the VR button is released. In order to change the mode of moving forward to moving backward, the user should press the VR button to click once at the left icon shown in Fig. 3 (a). This icon indicates whether front direction or back direction is used for this linear motion and it can be switched when clicked by the user as shown in Fig. 3 (b). In principle, the user can go to any possible place with this functionality by repeatedly choosing a direction to move. However, it could be inconvenient if the user simply wanted to change an angle to view the same content on the table. With this functionality only, the user would need to aim at a correct location exactly to achieve a desired angle and a comfortable view distance at the same time, which almost always fails and needs relocation. Therefore, a second means of movement was introduced.

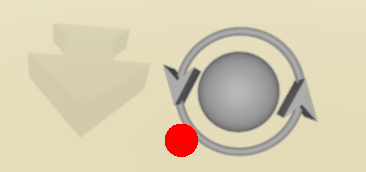
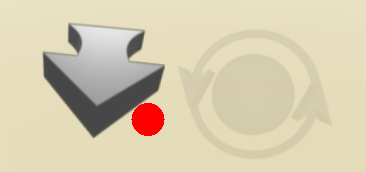
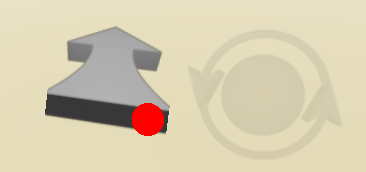


FIG. 3. Screenshots of the forward icon (a), the backward icon (b), and the orbiting icon (c).

(a)

(b)

(c)

The second was from typical 3D design programs such as 3ds Max and Maya, and this method is often used to view an object. This method rotates the viewer around a static centre with a fixed radius to view the thing at the centre from another angle quickly, like moving on a spherical coordinate. To implement this, a button with an icon representing orbiting motion was placed at the centre on the right as shown in Fig. 3 (c). When the user presses and tries dragging it by rotating the view angle, the user will drag him/herself around it, so that the user can move while keeping a fixed distance to the centre easily and quickly, with no need to wait for motion to progress like the previous linear movement. This method actually provides an alternative to roam around the VR world. The user can use forward or backward linear motion to increase or decrease the radius and use orbiting to adjust the azimuth angle, which encompasses the whole space. This movement method based on a spherical coordinate turned out to be much more convenient and is expected to be the major method used by the user.

A third movement means was designed to reset the user’s location to the default start. This is necessary because the user may move somewhere where the two icons controlling movement cannot be seen, or the user may get stuck or have moved too far away and want to return. This reset functionality is triggered by successively clicking the VR button for four times. It temporarily disables other means of movement and disables the collision geometry of the user object so that the user can penetrate through other things in the scene. Then, the user is moved to the destination point in 0.5 second in a predetermined smooth path with a sound effect. Some minutiae related may also be reset. This functionality was also designed as an emergency tool, and thus supposed to work well under any condition.

3.5 Programming Language

In order to implement the functionalities mentioned above, the visualized programming language provided by Unreal Engine was used. The name of the language is Blueprints. It is comparatively easy to learn, because it does not involve syntax, since all actions and data are visualized as nodes and modules connected by lines. However, except these, it has only small differences with C++, and it can be rewritten into C++ word by word. Epic Games Incorporation provides valuable documentation on Blueprints Visual Scripting [19]. This documentation is considered better than most of the books introducing Blueprints on the market, according to the results after many of them were read during this project. However, although this documentation may seem long, it is not perfect and many problems encountered were solved elsewhere. When a question or a problem appears, searching it on Google often lead to a link to Unreal Engine Answer Hub or forums where someone have asked a similar question and been answered [20, 21]. Almost all negligence, deficiencies and obsolescence of the official documentation have their complements there. Thus, online support is considered sufficient, though it is probably time-consuming to read through.

Since an official documentation exists, how Unreal Engine works or how Blueprints is written will not be described in this report. However, to provide a general understanding on how the program in this project works, some general concepts and ignored necessary details in the official documentation will be introduced. Some content below will suppose the reader to have basic knowledge about programming.

Blueprints programming language is an object-oriented programming (OOP) language, and is an event-driven programming language [19]. The main thread of the developed program cannot be accessed directly, and the developer can only access how different classes work and put instanced classes into the scene or make them referenced by the scene, so that the main thread can be accessed when scripts on instanced classes are executed. Because of this structure, an event-driven programming style was adopted. Most typical events are Begin Play, Tick (every frame update), End Play, On Overlap Begin and On Hit. These events are triggered under corresponding conditions and are self-explanatory. Also, customized input channels used in the developed program can create corresponding events, and they are valid only if the implemented class is set to receive input. Scripts are written behind these events to be executed when these events are triggered. Therefore, a class can be controlled and react to different stimulations, and can behave continuously using the Tick event.

3.6 Program Design

Because the language itself is object-oriented, an Object-Oriented Architectural Style was adopted in this developed program. In order to implement functionalities mentioned before, and specific functionalities needed for physics demonstrations, a few classes were created and defined, and their responsibilities and interconnections will be introduced, after three most important parts of this program are explained.

Considering the structure of the whole program, the inputs are the user’s VR device orientation, or view angle, and the triggering of the VR button; the output is a 3D view showing a satisfactory physics phenomenon expected by the user in a reasonable world environment. The environment was constructed as introduced previously, and showing the world in 3D, or stereoscopic two-screen rendering, is the responsibility of Unreal Engine and the Google VR plugin. Therefore, the rest needs to be handled by the developed program. To reaffirm what are needed in the program, how it is expected to be used by a user is summarized here. The user should run the program, see a start menu, choose a desired physics demonstration, watch the physics demonstration and adjust parameters on the control panel that influence the demonstration, and probably watch a diagram displaying data relationships, and can move at any time using the motion control system, and can return to the start menu to see another demonstration or quit the program. To divide the programming work, there are three most important parts, respectively responsible for user’s own motion control and interaction with the world, or in other words user interface, and control panels and widgets with the objects controlled showing a satisfactory physics phenomenon, and systems to extract data from the objects simulating physics and display them on a diagram.

To implement these three parts, five important classes were used. This first class was named Player, derived from Default Pawn class. It is a movable object in the scene used to represent the user, from where the user sees the world, and was assigned a collision shape to prevent itself from penetrating through or getting too close with another object. This class is responsible for handling inputs from the VR device, moving itself, and sending messages to other objects in the scene to interact with them. The second class is Control Panel, derived from User Widget class. It is shown as a 2D panel where buttons, texts, sliders and images can be placed, using the Unreal Motion Graphics (UMG) system [22], and this planar panel can be shown in a 3D scene by attaching to another actor class as a reference of a widget component [23]. This class is responsible for storage of data for physics demonstration, displaying the data, adjusting them according to user’s input with designs such as snapping, and controls over the progress of physics demonstration and what to display on the diagram through buttons. This class varies according to different physics demonstrations. The third class is Sphere, derived directly from the Actor root class, having a spherical visual representation, and is used as an object to simulate physics phenomena. The scripts written inside this class are used to initialize its physical behaviour according to data fed by the control panel, stabilize its physical behaviour due to flaws in the default physics engine PhysX using ad hoc methods, or control its behaviour if PhysX is not used, and pass data to display. It varies per demonstrations. The fourth class is Diagram, derived from Actor class. It was designed to not vary per demonstrations. It takes in three input values, x, y and id to draw a point on itself using Paper Grouped Sprite components, with id used to specify which group and colour this point is. For the diagram’s independency on the meaning of its displayed data, input x ranges from 0 to 1 and y ranges from -1 to 1, and therefore, data are rescaled before fed into it. The diagram also provides functionalities to show numbers and texts on the vertical axis as markers, to clear graphs drawn on it, to automatically clear points that have been drawn for too long, and to show two coordinates to draw two graphs at the same time. The last class is Data Proxy, derived from Actor, working with the corresponding Control Panel and Sphere to process the data passed from physics demonstrations to feed to the diagram and sometimes for storage. It also responds to the request of changing what kind of data to display, controls the lifetime of drawn points, resizes the graph if the numerical value of data is too large or small, and adjusts the texts displayed as markers on the diagram. It varies per demonstrations. Because it highly depends on what demonstration is used, it is handled as affiliated to its Control Panel, spawned when its Control Panel is constructed, and destroyed when destructed. Besides these five classes, many other classes are also used as complements, but usually not as important or complicated, such as a simple class to show a main menu, and therefore they will not be discussed.

This is the architecture of the program. Most implementation of functionalities described above is easy and trivial, or at least can be done through trial-and-error of parameter adjustment such as resizing the diagram or stabilizing physical behaviour under PhysX engine. Therefore, most of the implementation will not be elaborated further, except how user input is handled, and some rare problems encountered during development, of which the solutions may be hard to find.

Input events are received and consumed by Player class. The rotation of the VR device will always automatically result in the orientation change of the user camera placed in the scene so that a different view angle is displayed. Therefore, the user camera orientation should be accessed as an input data indicating where the user is looking at. The orientation of Player is aligned to the camera every frame so that the front direction of Player can match the view direction. A Widget Interaction component is attached to Player’s front direction to simulate a mouse cursor there when a 2D UI is present [24], and because Player’s front direction is the same as the view direction, the mouse cursor is simulated at the centre of view. This Widget Interaction component works by using a line tracing method to find a collision point if one goes along a certain direction in the scene, and use this hit point to simulate a cursor; mouse pressing and releasing events at that point can be triggered from Blueprints, and how the 2D UI reacts to these mouse events are defined in respective User Widget classes. However, this method only works with 2D UI components and does not support clicking on 3D objects. Therefore, besides sending a message of the mouse event through the Widget Interaction component when the user presses or releases the VR button, another message through a declared interface, Click Interface, is also sent to the object returned by line tracing. Thus, objects can receive click events through this Click Interface when they are pressed or released, and corresponding reactions are defined for respective classes, so that they can be held and dragged. In addition, whether the user is aiming at an object can be judged from whether the central red point in user’s view and the object are overlapping.

The next is how the motion control system was implemented. First, since the input channel is relatively limited, the number of successive clicks is considered a part of input information, and thus double-click, single-click, holding the button, four clicks are all considered differently. This was implemented with a successive click counter integer, which increments every time a button press is triggered, and clears itself to zero when the last button action has passed for over 0.25 second. With this counter as a switch argument, different commands can be distinguished when the user presses the button. As mentioned before, holding the button results in linear motion. To distinguish it from other multiple-click commands, only when the counter is one can the press event trigger a timer of 0.45 second to start linear motion, which can be nullified by a button release action. Also, the release event stops the motion if linear motion is in progress. Then, double-click was designed to trigger a special movement through an event dispatcher, which can be assigned and defined in respective demonstrations, and four clicks trigger resetting the position to default. It can be easily seen that if only these were implemented, these would work while interacting with the world. For example, clicking and holding a button on a 2D UI would result in linear motion inevitably. This was solved by a work-around, which was to decrement the successive click counter integer inside the Player object every time a button or a clickable object receives a press. Therefore, whenever the user presses a UI button, the click counter is decremented and then incremented, cancelling all effects, and no action can be triggered by the resultant zero counter. Thus, the orbiting motion can function correctly though the button is held. However, since the functionality of resetting the position to default is supposed to work under any condition, this functionality uses a separate counter which cannot be modified by other classes, and will always be triggered when four successive clicks are completed. This is all about how input is handled.

Other details which may result in difficult problems will also be introduced here. The first one is that, due to round-off error, when a 2D UI is placed exactly on a planar surface of an object, even though the UI may be seen on the screen, it may not function properly as line tracing can return the surface of the object under the UI but not the UI itself, because these two are too close. The planar UI must be placed a small distance above another surface to function well. The second is that a planar UI defaults to one-sided, and looking at it or working from an incorrect side has no effect. The third is that Begin Play events on different objects are executed sequentially. When one object’s initialization on Begin Play involves resultant data of other objects’ Begin Plays, a conflict can happen due to an undesired order of Begin Play execution. It can be solved by changing the program design to avoid such sequence requirement or use one object to spawn another so that the order can be controlled. The last is that, the default PhysX engine always handles two colliding objects with a relative velocity less than 2 m/s to stick together after collision, which is a large relative velocity in VR. This is bypassed by enlarging the scale of the world by hundreds of times, and then set the displayed scale on VR devices to be hundreds of times smaller to cancel the effect. In this way, the physics engine can treat a speed of 2 cm/s as over 2 m/s, but displayed as 2 cm/s on a VR device.

3.7 Optimization

After program implementation and debugging, optimization is also important for this VR program, because it is intended for students as smartphone users. It must be able to at least execute on an average smartphone on the current market with an acceptable framerate, since a low framerate can quickly incur discomfort in less than one minute, such as dizziness, a headache or disgust. Sony PSVR refuses to put any VR software lower than 60 frames per second (fps) on its market [25, 26]. However, a normal smartphone usually does not support very high framerates, and a typical high framerate is already 60 fps. As a framerate between 40 and 50 fps is already unacceptable according to tests, the framerate in this program is expected to be more than 50 fps.

The device used for tests in this project was Huawei Nova CAZ-AL10, roughly 2000 HKD when the report was written. The CPU is an eight-core 2000 MHz CPU, ARM Cortex-A53; the GPU is Qualcomm Adreno 506; the RAM is 4 GB 933 MHz [27]. Although the screen resolution of this device is 1920\*1080, when testing for optimization, it was run in the 1280\*720 mode, and therefore it should be faster than true speed. However, as it was only a test device to help optimization decisions, the validity of optimizations implemented should not be affected.

The purpose of optimization is to improve the quality of the program and reduce the computational cost or sometimes storage cost, and keep a good balance between them when there is a trade-off. The most important quality of this program to consider is the graphic quality on the screen, and the computational cost should be low enough for the program to operate above 50 fps. Important optimizations that increased the framerate by over 3 fps in the tests will be first introduced below.

Considering rendering principles, since the legs of the table asset in the Starter Content pack were very complicated, composed of many surfaces, and usually not seen by the user, the legs were a waste of computation. They were then deleted and replaced by a simple cylinder. This caused the framerate to increase from 58 to 62 fps.

Considering rendering principles, all 2D UI images are redrawn every frame by default, which is not necessary [28]. It is because the computational cost of this redrawing is not significant on most computer or console platforms. However, it turned out to be different on mobile platforms. By adding Invalidation Boxes to wrap and cache texts and images on UIs, the framerate on the test device increased by roughly 3 fps. To show reactions towards user’s action, the redrawing of texts and buttons was then implemented manually through programming.

By default, the material to display the floor is Default Material of Unreal Engine. However, it was found to be one of the most computationally expensive features. The framerate dropped to 47 fps when the view was filled with this material, and instructions of rendering this material were found complicated. Therefore, it was targeted and replaced. If the floor became a simple colour block, the user would not be able to feel a distance from it, as there would be nothing discernible, which would make it confusing and unnatural. Therefore, a new material to apply to the floor was designed, with a squared colour pattern created from a Cellnoise function, desaturation and colour multiplication, and a normal map modified from Default Material to show the outline of tiles. The colour pattern was then baked to be a texture, so that the colour of the material would not be computed during runtime but read from a texture file [29]. Runtime computation of colour created from noise is expensive, because the noise is computed from formulas and thus the resolution is infinite, and thus it cannot be cached but calculated per pixel every time it is displayed, which is usually too computationally expensive for a mobile device. The framerate increase after this optimization was about 3 fps when the user was looking at the table, and over 10 fps when the user was only looking at the floor. Other performance optimizations were minimal and will not be mentioned.

Next, some important trade-offs and compromises involved when developing this program will be introduced. The Mobile HDR display mode, which means high dynamic range, was disabled due to its high cost, though it must be enabled to support post-process light effects such as blooming [30]. When it was enabled, the framerate reduced by nearly 30 fps. Regarding 2D UIs, if a large image size of a high resolution is scaled down in the scene to be displayed small, the image can appear shaper and clearer than a low-resolution one. Although this method can also be used for UIs in the program to have better images against aliasing (or, jaggies), it turned out to cost around a 12-frame reduction per second, because the image size accessed during rendering became much larger to handle. Therefore, finally the UI was created by enlarging a low-resolution image with small font sizes to improve efficiency.

Finally, some adopted optimizations for quality will be introduced. The first is anti-aliasing (AA). Temporal AA method was chosen in project configuration in Unreal Engine, and the sampling rate of anti-aliasing was set to 8 in device profile configuration. It was chosen because it had the best quality during tests. Indirect Lighting Quality was set to 3.0, higher than the default, so that shadows and colours on the table can appear clear and normal, and lightmap resolutions was adjusted by trial-and-error to show a clear undistorted shadow [31]. All colours displayed in the scene were a result of repeated trial-and-error, the same as sizes and locations of all objects, for a comfortable view and natural appearances as possible. Physical parameters displayed on UIs were also designed to be simple integers if possible, because a value with complicated decimals and units can be hard to display on a low-resolution VR device. Lastly, because hitches can incur discomfort in VR, for smooth transitions between different physics demonstrations, a sublevel system was used [32]. In Unreal Engine, if C++ is not involved, only a sublevel system is found supporting using a separate thread to load and initialize content in parallel with the main thread. Using this system, objects used for different physics demonstrations were placed in different sublevels, under the persistent level that only contained the table, floor, sky, lighting, and control icons in the middle. When loading and unloading different sublevels, content could appear and disappear on the table for use, and the loading and unloading actions were triggered by Blueprints from scripts on UI menu buttons. However, in Unreal Engine, because shaders on mobile platforms are always compiled when they are first used [33], the program always hitches when something is displayed on the device for the first time. This problem was not eliminated successfully.

3.8 Packaging for Distribution

After the program was completed, it was packaged as an installer for distribution on the Android platform. To pack it for Android, some development tools for Android must be correctly installed and configured, which is not an easy or firmly reliable process. According to Unreal Engine 4 documentation, Android Quick Start manual [34], most of necessary development tools can be installed, by following the instruction of an installer which is put under the directory of Unreal Engine. However, the documentation has obsolete information and is itself not complete, and only following this manual will result in a package failure, simply because configuration on Unreal Engine is not included in it. The necessary configuration is included in the second part of Google VR Quick Start manual [35].

In principle, the program should be packaged after following these instructions. However, a great deal of errors can occur and the package may fail, probably indicating low stability of Unreal Engine and Android development kits, as a great deal of error reports can be found on the Internet. Usually, a problem can be targeted and eliminated by searching with messages found in the output log indicating how or why a first error was produced, and then trying possible solutions given in the search results exhaustively, until the error messages finally change or disappear. However, it is still difficult, and sometimes some people report that their problem cannot be solved by one solution while others’ can, even though they share a same error message. They system may be highly unstable, and a developer is recommended not to change configuration or update development tools once the program can successfully package, in case it fails again. Below, five important parts that may result in package or other failure will be introduced, which were all encountered during this project.

The first is about incompleteness of installation of Android tools. The manual of Unreal Engine does not install all necessary tools for Android package and distribution. Using information from error output log, more tools may be installed by running “SDK Manager.exe” under the directory of Android SDK. Tools for Google Play service must be installed to work correctly, and SDK platforms specially for newer or older Android versions can also be installed, so that SDK level configuration in Unreal Engine can have more valid options. Although many data can be updated using this installer, some updates will result in error and should be avoided, because this SDK is originally installed as a part of Android tools by Unreal Engine, and not supposed to work to manage itself though it can. If an error happens, it can be irreversible and reinstallation may be needed.

The second problem is about version compatibility. Android SDK is separate from Unreal Engine as a third-party tool, and therefore, sometimes a high version of Android SDK is not supported by Unreal Engine, because Unreal Engine may not follow its update speed. Therefore, a lower version of Android SDK should be chosen if it is confirmed that Unreal Engine does not support the high version. However, this results in even further problems. Android development tools are many, and they rely on each other to some extent. Thus, a different version of Android SDK only could result in incompatibility with other tools, and further adjustment may be needed, which increases complexity further. A best solution can be reinstallation of all tools with an earlier version. However, sometimes a high version of a tool can be a must. For example, only a Java Development Kit (JDK) as high as version 1.8 supports Google VR, which is not included in any full reinstallation of an earlier version. Then, to solve this problem, a full earlier version was installed while only some minor parts including JDK were updated to newer versions, and fortunately a compatibility problem did not occur. The version problem may only be solved by repeated reinstallations and tests. Generally, an older version is preferred if possible, even though it may work less efficiently.

The third is about digital signature. Although this is not mentioned in manuals of Android Quick Start or Google VR Quick Start, it is a must for distribution. Without a signature, the software cannot be installed on an Android smartphone. A reliable manual can be found in Unreal Engine documentation [36]. This process hardly errors.

The fourth is about environment variables. Environment variables are needed to tell the location of installed tools to any possible program that may want to know. With these variables set, the Android tools can be found by programs such as Unreal Engine. Sometimes environment variables are not correctly set, and need manual check and resetting. Related information can be found in Unreal Engine documentation, which also provides some extra information on how Android tools may be installed [37].

The fifth is about file error, which cannot be definitely prevented. During installation, as sometimes a few programs are not supposed to work simultaneously, and sometimes a correct order of installation and reboot is needed, error can easily occur due to imperfect operation. Moreover, as the installation process itself is long, data error can be inevitable and may accumulate. Also, Unreal Engine itself faces the same situation sometimes, due to accumulated file error inside the engine after operating for long. These are solved by reinstallation, since the error reason can hardly be targeted. File error also often occurs outside a development tool. Sometimes error occurs due to inconsistency in cached temporary files, which can be solved by deleting all of them, and sometimes the files of the developed program can error, which happened for over six times during this project, and can be solved by discarding it to recreate a new one. Because of this, regular manual backups are recommended. These two errors are hard to find out because they may not be reflected in output log or anywhere.

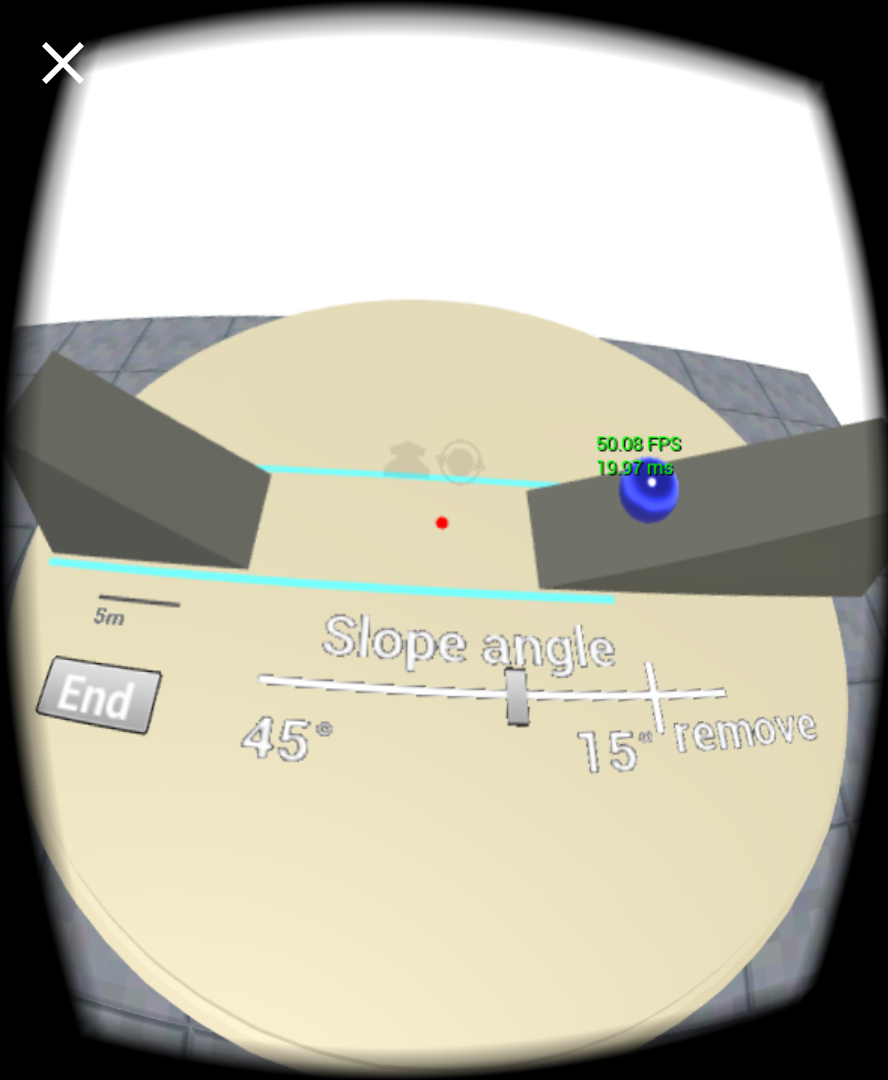
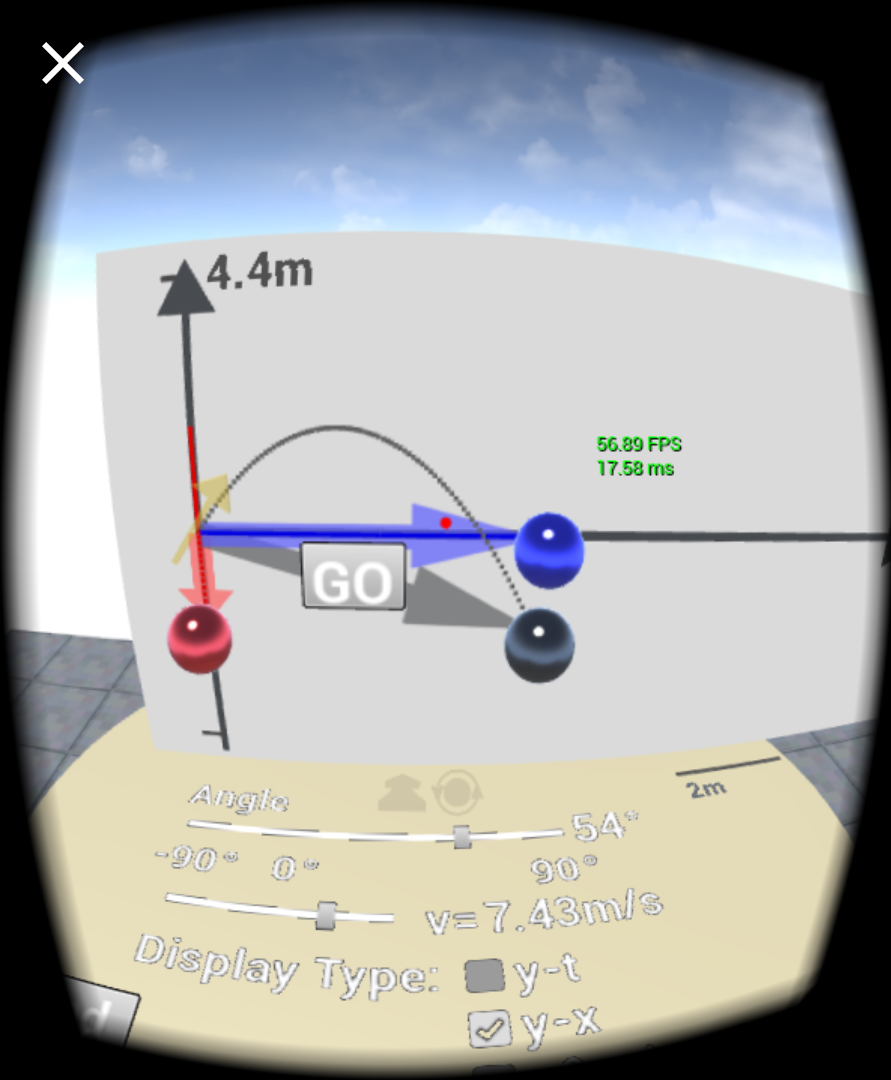
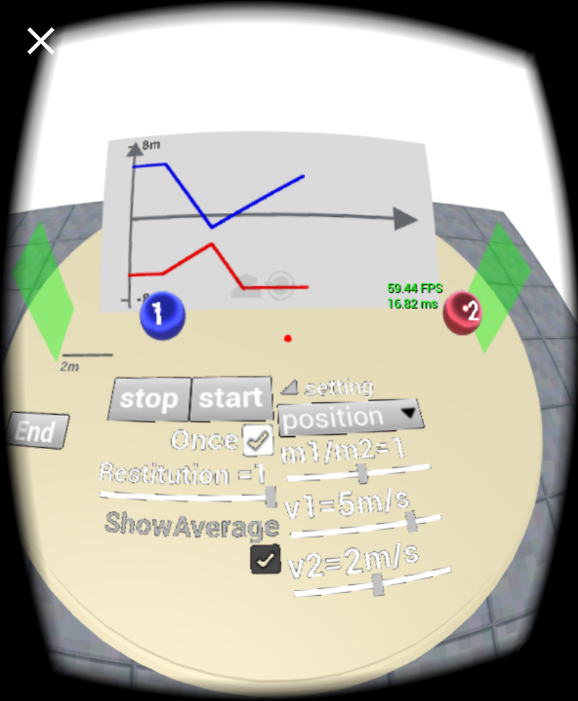
After eliminating errors, a package that can be installed on an Android device should be built and ready for distribution, which is expected to function well as described previously.

CHAPTER 4 RESULTS

In this program, there are currently three physics demonstrations, respectively demonstrating Newton’s 1st Law, projectile motion, and linear collision. The content will be described below.

The first one shows the Galileo’s experiment where a sphere rolls up a slope to the same height by which it is released from, from which Newton’s 1st Law can be reasonably deduced. The user can drag the sphere on a track and release it on the left slope to see it moving steadily to and fro between two slopes, and return to the same height even when the right slope angle is adjusted, and when the right slope is removed, it goes away endlessly. The second demonstration, projectile motion, actually includes free fall, vertical throw and horizontal throw, since the initial velocity and initial angle can be adjusted. It shows a sphere under projectile motion with specified initial properties when triggered, and it stops when hitting the table. To demonstrate the principle of projectile motion, two other spheres are used to represent decomposed horizontal displacement and vertical displacement, and they move simultaneously with the original sphere. Three arrows representing the respective displacements are drawn, forming two sides and a diagonal of a rectangle to show the combination of vectors. Besides using the planar UI to set the initial angle, the user can also drag an arrow representing throw direction on the sphere to set it, which is more intuitive. Different relations can be displayed on the diagram placed behind the sphere, including y-t, y-x relations, and both x-t and y-t simultaneously. The third, linear collision, appears less interesting, probably because the physics phenomenon is not continuous but discrete, and therefore, typical physical process that can be visually shown does not exist. In this demonstration, the user only uses a planar UI to set two initial velocities, a mass ratio, a restitution coefficient, and the user clicks buttons to start or stop the two spheres moving in the middle. The spheres collide, and stop or bounce when hitting one of two green walls put aside as borders. The user can uncheck the Once checkbox to make them bounce off the walls and collide endlessly. Different properties of the spheres can be drawn on the diagram with respect to time, including displacement, velocity, momentum and energy. Average values of momentum or energy can also be drawn. However, the momentum and energy graphs seem difficult to provide any information other than conservation and energy loss, as the relations behind are complex. To make the demonstration more interactive, the user can double click at a sphere to take the sphere’s view point with a real dimension of lengths and velocities, and move along with the sphere to watch how the collision happens. Start and stop buttons are then transplanted to the sphere so that the user can control on it, and the user can do a double-click or click for four times to exit this view mode. These are shown in Fig. 4 as below.

FIG. 4. Screenshots taken on the test smartphone of the respective three physics demonstrations.



(a)

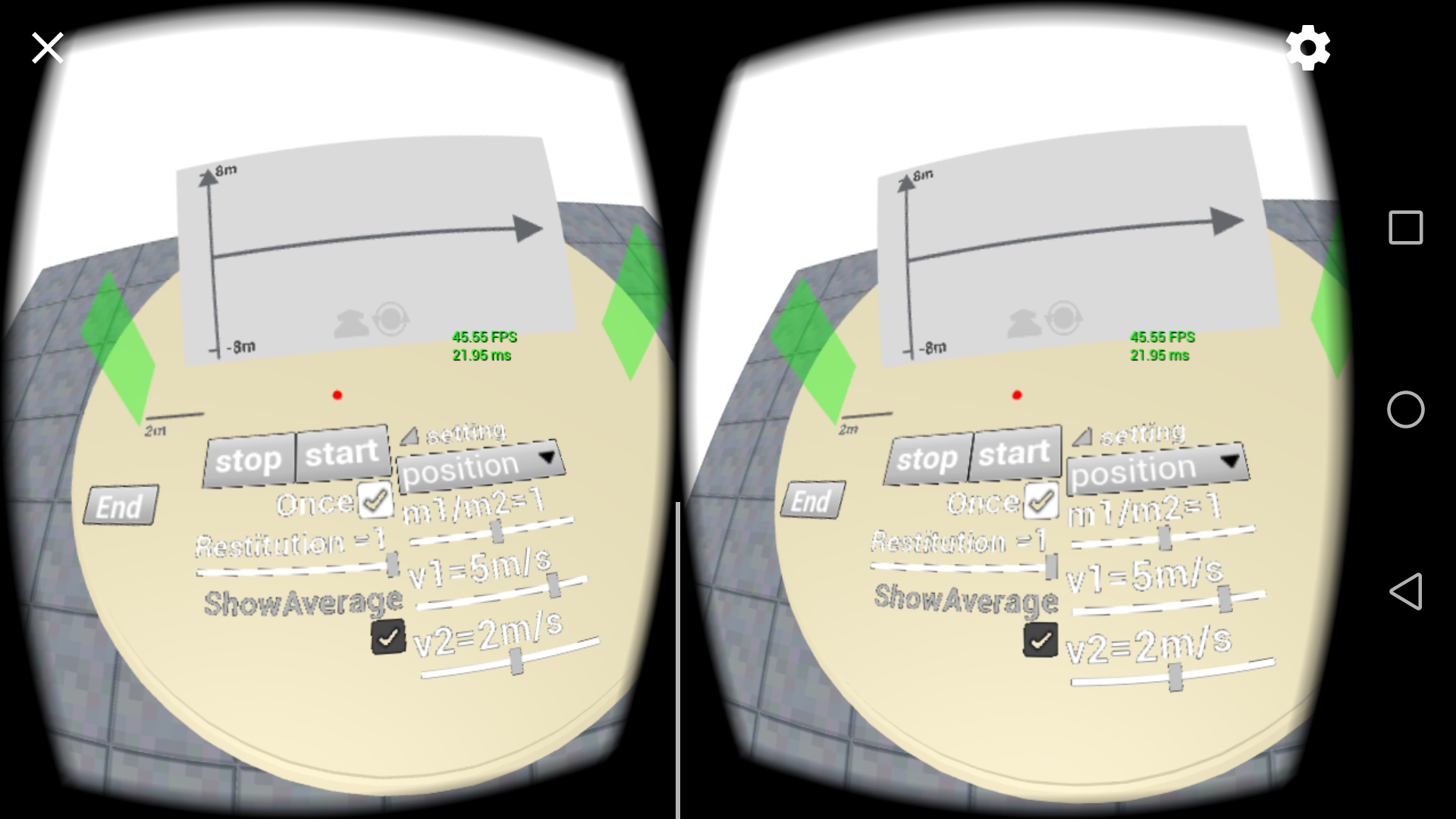
(b)

(c)

On the test device with specifications described previously, the average framerate was 60 fps running in 1080p resolution, with a minimum of 56 fps when there was much content on the screen, and a maximum of 63 fps when the screen was only filled by the sky or the floor. No occasional hitch was present and it always worked smoothly, except the time an object first displayed on the device. Colours were normal and user inputs were correctly responded without bugs. Although the screen image was always shaking due to the small shaking of hands holding the device, it was not difficult to rotate the device accurately enough to aim at a target icon to click the button. On the other hand, the user movement system worked quickly and satisfactorily, though the user could also choose to stay at the default location to operate everything. However, during the tests of the development of this program, this program crashed for two times when running, with no special reason, and it froze twice when using by testers. As no regular pattern was found and it could not be reproduced, it is considered as a problem about Google Cardboard plugin or Unreal Engine stability by itself.

The program was also tested on Sony Xperia Z5 Premium, released in 2015 and priced around 3500 HKD now [38]. The average framerate was 61 fps. On Huawei p9, released in 2016 priced around 3000 HKD, and on Blackberry STV100-1, released in 2015 around 3000 HKD, it was fluctuating mainly below 50 fps with heat produced [39, 40].

Concerning the performance of the tested VR device, Google Cardboard V2, which is roughly 50 HKD, the view angle was measured and estimated to be up to 38 degrees from the leftmost to the rightmost. During the test of this program, however, it was only comfortable to view within a cone of an approximately 20-degree angle in the centre. If the view angle changed quickly, or the user viewed at an angle not at the centre of the display device, an uncomfortable distortion of distance could be felt. Therefore, a user cannot quickly look at things that are far apart. Also, chromatic aberration was present, as Fig. 5 (a) shows, and the aliasing was still severe, making the information on UIs appear in disorder and annoying, as Fig. 5 (b) shows. The strange points are that, this aliasing was not by one pixel but by around three pixels together, and aliasing artefacts were the same on both the left and the right screen. For example, for the v1 slider in the figure, there are three pairs of aliased black lines on it, however, this is the same on both the left and the right screen, which display the thing viewed from different angles and thus should not possibly have identical artefacts if the rendering principle is the same as normal single-screen rendering.



(a)

(b)

FIG. 5. A photo of the screen taken through a VR lens (a), and a complete screenshot on the test smartphone (b).

Concerning the final program size, the package size is about 40.1 MB, and the size is about 120 MB after installation on an Android device. The amount of coding was counted by hand to be roughly 2,000 blocks in Blueprints excluding all pure data accesses and simple data operations, and the equivalent C++ coding might be over 3,000 or 4,000 lines as C++ is usually not extremely compact. Moreover, C++ in Unreal Engine needs extra coding, and it is also responsible for class configurations, which should make it even more. Including deleted and unused code during development, the number of equivalent C++ lines can possibly approach 5,000. As the recorded fastest production of code was said to be 1,000 lines per developer per week by Borland professional teams, as mentioned by Sutherland [41], the amount of coding may already be too large for a single developer, which is professional full day-time work for about one month.

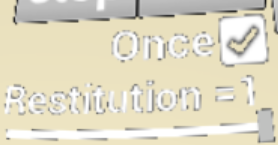
Enough user feedback was not collected, and therefore it is not reliable as an evidence, but is only comment that may be considered for reference. A tester as a HK PolyU student in year one tried and commented this program. He was satisfied with it and described it as novel, impressive, interesting and full of content, and helpful for remembering physical phenomena. Regarding using comfort, it was said that he did not feel any discomfort during use, but felt dizzy after the VR device was taken off. Also, it was described as a little breathless and deadened, which might be related with other discomfort. Another tester in year four majoring in biology commented that, it was real enough and helpful for teaching, and aliasing on UIs was kind of acceptable because the user did not need to focus on them for long.

CHAPTER 5 DISCUSSION

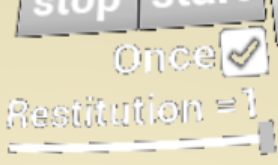
According to the framerate results, the computational cost performance is satisfactory enough, so that a user does not need an expensive smartphone to run it as expected. However, the used Google Cardboard VR device is not satisfactory, because a user often feels dizzy after using it. It may be caused by the distance distortion mentioned earlier. As the lenses always show distorted 3D images, a user may get used to this distortion and only focus at the centre while perceiving the surroundings farther than they appears on the device, and this adapted perception may explain why a user does not feel dizzy in VR but feels dizzy after the VR use since he/she needs to return back to normal distance perception. This may be the reason why it feels like a kaleidoscope and neither natural nor precise. Therefore, the quality of the device may need to be improved, while the quality of the smartphone is enough.

According to the user feedback, it was far more interesting when compared with conventional 2D demonstrations on planes, and the degree of perfection of this program was considered high. These advantages can be of vital importance because the user’s impression and engagement are related to effective learning, and the user may better remember how physics work and how the principles are by seeing attractive demonstrations for them, however, these can also be adversely affected by displeased user experience resulting from an unsatisfactory VR device.

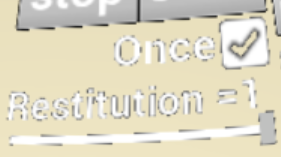
Regarding the most obvious problem, the aliased 2D UIs, there is not yet an easy solution. When testing the aliasing behaviour, it was found that probably in order to ensure artefacts seen from two screens are identical, the aliasing occurs by always more than one pixels as shown in Fig. 6 (b), and the aliased pixels are already mixed with background colour to be vague, which is different from normal rendering. Therefore, the severe aliasing effect is not only present on UI images. However, no anti-aliasing method seemed to make a large difference on UIs as shown in Fig. 6 (c)-(e) while anti-aliasing works well on other objects as Fig. 6 (a). Anti-aliasing could still be observed a little at the line in the upper left corner in the figure, making lines more continuous. The reason why temporal AA looks better in practice may be that, temporal AA provides image retention, which makes the flickering intermittent lines appear more continuous, however, no literature was found to confirm this yet. The reason for anti-aliasing failure on UIs is supposed to be that Unreal Engine renders UIs differently. Since the problem is not solved by Unreal Engine yet, it can possibly be solved by using small objects to represent the texts and buttons to simulate a customized UI that can be rendered not as default 2D UIs, and thus, anti-aliasing can work at those edges to remove aliasing, because other normal objects were indeed rendered correctly. As the programming is not complex, this solving process may mainly be construction of the assets. However, as the aliasing on button edges in this program is only as wide as three pixels, the improvement after customized UI construction may not be much, though the flickering of aliasing should be removed.



(c)

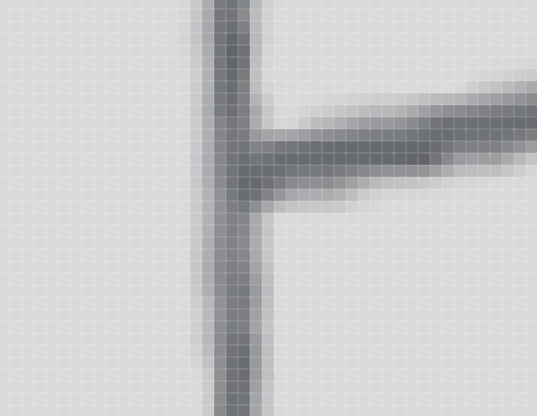


(d)

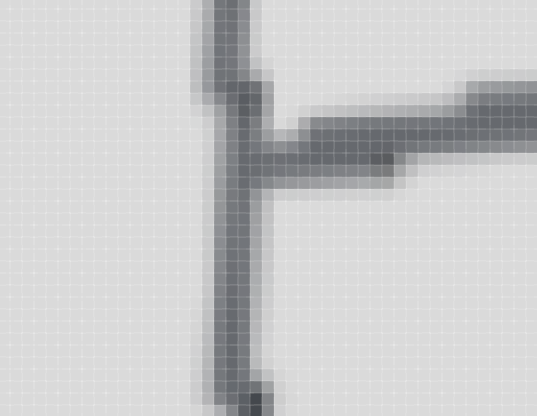


(e)

FIG. 6. Screenshots on the test smartphone of the intersecting point of two arrows without anti-aliasing applied (a) and with anti-aliasing applied (b), and screenshots of the UI texts and buttons without anti-aliasing (c), with temporal AA (d), and with MSAA (e).



(a)



(b)

Regarding the content, it is still far less than what can be done as there are only three demonstrations, and this program is not yet enough for a student to learn much about mechanics. However, concerning the workload, it is already high for a single amateur developer as mentioned earlier, as a professional programmer may need many days with standard working hours to develop such a program. Including the effort to study Unreal Engine and 3ds Max, the workload can be even larger, since the materials read during this project can roughly accumulate up to five books, and up to six or seven if books read during straying are included. It is also often commented that learning Unreal Engine is hard and needs months [42, 43], and there are over hundreds of items as options and parameters, which almost all need to be understood and confirmed in order to ensure that the program will work as expected, and this was indeed done during this project. Due to lack of study on 3ds Max, mere the table object was edited for more than six times for totally roughly 8 hours with much cancel and repeat, and also the program was restructured for many times, which wasted a great deal of time due to lack of experience. However, after having some experience, a physics demonstration excluding asset editing could be completed within only four days, which was much faster and acceptable. Actually, more time may have been used for debugging, searching for information, targeting and understanding problems, tests, and optimization.

CHAPTER 6 CONCLUSIONS AND RECOMMENDATIONS

Overall, the completion of the program developed for physics demonstration in VR is satisfactory, though some problems still exist, and it should help students to learn better than conventional 2D demonstrations. Therefore, maybe it is worth being developed more to include more content. Force decomposition, forces on a slope, friction, the law of the lever, rotation, the spring and harmonic oscillation may all be included for demonstration on mechanics. Other users recommended the doppler effect and light wave construction and destruction, which are also possible because a wavy surface can be constructed using Unreal Engine to represent the wave propagating [44], and respective results of the propagating wave can be shown in another way such as a different sound tone for the doppler effect or light brightness for light interference. These were not done or tested due to lack of time.

Concerning the target VR device, the quality should be improved for comfortable use. Although a high-end device on a computer or console is not necessary to run this program, the program can be deployed easily to those platforms owing to Unreal Engine functionalities. Other platforms may also be considered, especially the iPhone, as many students were found using iPhones when searching for testers in this project.

Regarding the ideal high-end VR device, more research and tests may also be completed, as it is the future direction and the price should be decreasing with the development of technology. With a high-end VR device, most of optimization considerations mentioned before can be ignored without any noticeable negative influence, since the computational capability is much higher; the user is supposed to view an undistorted precise scene and have a better user experience, and have much more freedom in the VR world, such as using a controller to drag objects and throw objects to initiate physics processes, without need for complicated setting on UIs, which is more comfortable and natural, and actually the programming can also be simpler. The displayed image quality can be higher due to high resolution of the device, and artefacts like aliased 2D UIs mentioned earlier can become minimal.

In any case, the development of physical phenomenon demonstrations in VR should be worthwhile. On the one hand, it matches the current technology development from 2D to 3D; on the other hand, it is intuitive, impressive and helpful for students’ learning. Probably, scientific physics simulations may also be viewed on VR devices for better demonstrations and understanding in future researches. Nevertheless, the project should be considered as interdisciplinary since most of the work done, except for investigation at the beginning, does not require much knowledge on physics.

References

1. J. Mirt, Virtual Reality HMDs 2016 Sales Numbers, <https://www.viarbox.com/single-post/2017/01/20/Virtual-Reality-HMDs-2016-Sales-Numbers>, accessed 19th Apr 2017.
2. R. Cappellina, Google has shipped 5 million Cardboard viewers so far, <https://www.wareable.com/google/5-million-google-cardboard-viewers-2234>, accessed 19th Apr 2017.
3. The definition of virtual reality, <http://www.dictionary.com/browse/virtual--reality>, accessed 19th Apr 2017.
4. 3D Organon VR Anatomy, <https://www.oculus.com/experiences/rift/872418872856459/>, accessed 19th Apr 2017.
5. T. Djukic, V. Mandic, and N. Filipovic, Comput. Biol. Med. **43**, 2046–2052 (2013).
6. C. A. Kilmon, L. Brown, S. Ghosh, and A. Mikitiuk, Nurs. Educ. Perspect. **31**(5), 314-317 (2010).
7. 50 years of VR with Tom Furness: The Super Cockpit, Virtual Retinal Display, HIT Lab, & Virtual World Society, <http://voicesofvr.com/245-50-years-of-vr-with-tom-furness-the-super-cockpit-virtual-retinal-display-hit-lab-virtual-world-society/>, accessed 19th Apr 2017.
8. PhET: Free online physics, chemistry, biology, earth science and math simulations, <https://phet.colorado.edu/>, accessed 19th Apr 2017.
9. Teaching Physics with Virtual Reality Simulations, <http://www.physics.ohio-state.edu/~physedu/vr/>, accessed 19th Apr 2017.
10. C. Savage, D. McGrath, T. McIntyre, M. Wegener, and M. Williamson, AIP Conf. Proc. **1263**, 126-129 (2009).
11. H. Kaufmann and B. Meyer, Themes in Science and Technology Education **2**(1-2), 117-130 (2009).
12. Physics in Leadwerks3D, <http://www.leadwerks.com/werkspace/topic/5102-physics-in-leadwerks3d/>, accessed 19th Apr 2017.
13. R. Davis, Unreal Engine 4 Goes Free for Academic Use, <https://www.unrealengine.com/blog/unreal-engine-4-goes-free-for-academic-use>, accessed 19th Apr 2017.
14. Unity for Education, <https://store.unity.com/education>, accessed 19th Apr 2017.
15. Free Student Software Downloads | Autodesk Education Community, <http://www.autodesk.com/education/free-software/all>, accessed 19th Apr 2017.
16. Autodesk - Educational Purposes, <http://www.autodesk.com/company/educational-purposes>, accessed 19th Apr 2017.
17. Get Cardboard – Google VR, <https://vr.google.com/cardboard/get-cardboard/>, accessed 19th Apr 2017.
18. HP Pavilion Notebook 14-al087tx Product Specifications | HP® Customer Support, <https://support.hp.com/hk-en/product/hp-pavilion-14-al000-notebook-pc-series/10862154/document/c05232406>, accessed 19th Apr 2017.
19. Blueprints Visual Scripting | Unreal Engine, <https://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html>, accessed 19th Apr 2017.
20. UE4 AnswerHub, <https://answers.unrealengine.com/>, accessed 19th Apr 2017.
21. Unreal Engine Forums, <https://forums.unrealengine.com/>, accessed 19th Apr 2017.
22. UMG UI Designer | Unreal Engine, <https://docs.unrealengine.com/latest/INT/Engine/UMG/index.html>, accessed 20th Apr 2017.
23. Widget Components | Unreal Engine, <https://docs.unrealengine.com/latest/INT/Engine/Components/Widget/>, accessed 20th Apr 2017.
24. Widget Interaction Component | Unreal Engine, <https://docs.unrealengine.com/latest/INT/Engine/UMG/UserGuide/WidgetInteraction/>, accessed 20th Apr 2017.
25. J. Davidson, Sony’s pushing VR devs to hit 90fps, solid 60fps “is a minimum”, <https://www.technobuffalo.com/2015/11/20/playstation-vr-framerate-90fps-60fps-minimum/>, accessed 20th Apr 2017.
26. T. Powter, Sony to Reject All PSVR Games That Don’t Hit 60FPS or Higher, <http://twinfinite.net/2016/03/sony-reject-all-psvr-games-dont-60fps/>, accessed 20th Apr 2017.
27. Huawei Nova CAZ-AL10 – Specifications, <http://www.devicespecifications.com/en/model/e63f3eef>, accessed 20th Apr 2017.
28. Invalidiation Box | Unreal Engine, <https://docs.unrealengine.com/latest/INT/Engine/UMG/UserGuide/WidgetTypeReference/Invalidation/>, accessed 20th Apr 2017.
29. R. Brucks, Getting the Most Out of Noise in UE4, <https://www.unrealengine.com/zh-CN/blog/getting-the-most-out-of-noise-in-ue4>, accessed 20th Apr 2017.
30. Post Process Effects on Mobile Platforms | Unreal Engine, <https://docs.unrealengine.com/latest/INT/Platforms/Mobile/PostProcessEffects/>, accessed 20th Apr 2017.
31. LightingTroubleshootingGuide - Epic Wiki, <https://wiki.unrealengine.com/LightingTroubleshootingGuide>, accessed 20th Apr 2017.
32. Level Streaming | Unreal Engine, <https://docs.unrealengine.com/latest/INT/Engine/LevelStreaming/index.html>, accessed 20th Apr 2017.
33. ios metal rendering issue in 4.13 - UE4 AnswerHub, <https://answers.unrealengine.com/questions/481655/ios-metal-rendering-issue-in-413.html>, accessed 20th Apr 2017.
34. 1. Required Android Setup | Unreal Engine, <https://docs.unrealengine.com/latest/INT/Platforms/Android/GettingStarted/1/index.html>, accessed 20th Apr 2017.
35. 2. Google VR Project Setup | Unreal Engine, <https://docs.unrealengine.com/latest/INT/Platforms/GoogleVR/QuickStart/2/index.html>, accessed 20th Apr 2017.
36. Signing Projects for Release | Unreal Engine, <https://docs.unrealengine.com/latest/INT/Platforms/Android/DistributionSigning/index.html>, accessed 20th Apr 2017.
37. Android Development Reference | Unreal Engine, <https://docs.unrealengine.com/latest/INT/Platforms/Android/Reference/>, accessed 20th Apr 2017.
38. Sony Xperia Z5 Premium - Full phone specifications, <http://www.gsmarena.com/sony_xperia_z5_premium-7536.php>, accessed 20th Apr 2017.
39. Huawei P9 - Full phone specifications, <http://www.gsmarena.com/huawei_p9-7972.php>, accessed 20th Apr 2017.
40. Specs: Blackberry Priv (AT&T) [STV100-1] | Swappa, <https://swappa.com/specs/blackberry-priv-att>, accessed 20th Apr 2017.
41. How many lines of code do professional programmers write per hour? – Quora, <https://www.quora.com/How-many-lines-of-code-do-professional-programmers-write-per-hour>, accessed 20th Apr 2017.
42. R. Fortin, 5 reasons why Unity is better for learning game development than Unreal Engine | Pixelz Games, <https://www.pixelzgames.com/single-post/2014/02/28/5-reasons-why-Unity-is-better-for-learning-game-development-than-Unreal-Engine>, accessed 20th Apr 2017.
43. How much time to take to learn C++, UE4, Blueprint?, <https://forums.unrealengine.com/showthread.php?90148-How-much-time-to-take-to-learn-C-UE4-Blueprint>, accessed 20th Apr 2017.
44. Render Target Lookup - Epic Wiki, <https://wiki.unrealengine.com/Render_Target_Lookup>, accessed 20th Apr 2017.

Appendix

The source file and packages for Android and Windows are published on

<https://github.com/wzkkzw12345/FYP13102247d_PolyU/tree/backup-upon-FYPreport-submission> under the MIT License.