Volume 7, Issue 4, April 2017 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Building Virtual Reality Applications with Bluetooth Controller Interaction for Google Cardboard using Unity 3D

Rajat Gupta, Rohan Nawani, Vishal P. Talreja, Assistant Prof. Sharmila Sengupta
Department of Computer Engineering, V.E.S. Institute of Technology, Mumbai, Maharashtra, India

Abstract— This paper discusses one way to develop a Virtual Reality game with interaction using Bluetooth Controllers for devices with support for Google Cardboard. This paper is inspired by the lack of applications on the Android Play Store that have little to no interaction with the environment. The virtual reality apps currently on the market are limited to being simulations of an environment with very little control given to the user. This paper aims to give people a way to develop bluetooth controller interaction into the applications they are developing.

Keywords— Virtual Reality, Bluetooth Controller, Unity 3D, Google Cardboard, Physics Raycast.

I. INTRODUCTION

Over the past five years, the Virtual reality scene has exploded. Devices like Oculus Rift [1], HTC Vive [2] and Playstation VR [3] have started offering virtual reality experiences to the masses but they are still very expensive and only people with disposable income can acquire such expensive devices. Google is also offering Virtual Reality experiences [4] in two flavours, first comes the Daydream, Google's latest entrant into the VR world. The only problem with Daydream is that it is currently only available to people that purchase Google's flagship phone Pixel. Second, Google had introduced the Cardboard, the VR experience for the masses. It is a cheap and affordable VR experience for anyone with a modern smartphone.

Developers have been developing all kinds of experiences for Virtual Reality devices. The aim of this paper is to provide an easy tutorial to developers to incorporate bluetooth controller interaction into the apps they are developing. This tutorial will show you how to add support for movement and interacting with objects in the environment using a bluetooth controller. It will include everything from setting up a sample scene, the requirements and the steps required to create your first VR application with support for Bluetooth controllers.

II. TOOLS USED

This section will enlighten us about the various tools that have been used throughout the development of this tutorial.

A. Game Engine: Unity 3D 5.6

In this project, the free version of the game engine Unity3D [5] is being used for the production of this tutorial. Unity is best suited for small and middle-sized development studios who cannot afford to invest in expensive high-end rendering engines. The primary reason for choosing Unity was the fact that it is very easy to use and to learn. Developing games with Unity is mainly based on a drag and drop functionality with the occasional adapting of scripts rather than writing code. Also, Unity provides multiple built-in shaders and effects as well as a physics engine and collision detection. Apart from shaders and effects, Unity provides various scripts mostly written in C# or Unity's JavaScript like script called UnityScript which can be added onto the 3D models. These scripts provide the models with the basic functionality and can be used for example for purposes like character controlling, defining actions of a rigid body object, etc. One of the drawbacks of using Unity is that it lacks forward compatibility, i.e., it cannot accept inputs from a version other than which project is being developed on.

B. Google VR SDK for Unity

Unity's native integration with Google VR makes it easy to build Android applications for Daydream and Cardboard. The Google VR SDK for Unity provides additional features like spatialized audio, Daydream controller support, utilities and samples.

Unity's native support for Google VR makes it easy to begin a new VR Unity project from scratch; adapt an existing Unity 3D application to VR and make an app that can easily switch in and out of VR mode.

The integration with Google VR provides user head tracking, side-by-side stereo rendering, detecting user interaction with the system (via trigger or controller), automatic stereo configuration for a specific VR viewer, distortion correction for a VR viewer's lenses, an alignment marker to help center the screen under the lenses when you insert your phone into a viewer, automatic gyro drift correction.

III. DEVELOPING THE GAME

In this section, we will start developing a simple game. The game will have you walk around on a simple plane on which you can move around using the joystick on the controller. There are multiple objects like cubes, spheres and cylinders surrounding you. These objects change color when you look at them and can be disabled when you press a button on the controller.

A. Setting the Scene

We start a new 3D project on Unity 3D version 5.6. We import the Google VR SDK for Unity package into our project. Add a Plane to the scene and scale i think so size so that we have place to walk around. We also add the GvrViewerMain [6], GvrEventSystem objects to the scene. The former will help us visualise the scene as if we were seeing it on our phone and the latter will enable us to add functionality to the pointer. Figure 1 shows the scene in Unity.

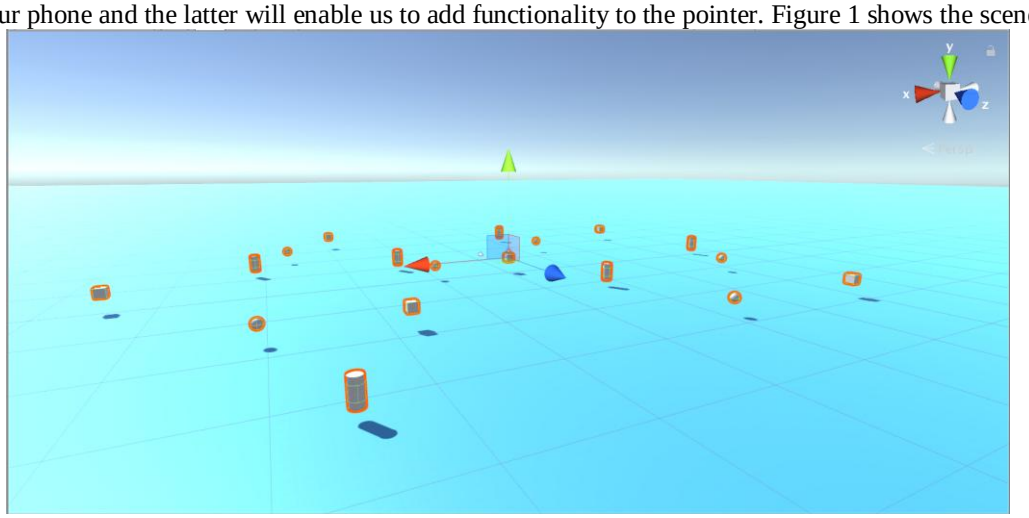

Figure 1. Scene in Unity 3D.

Before we start the development, we should change the Build Settings for our project. We change the development environment to Android and in Player Settings, we select Virtual Reality Supported and add the environments we want to build for. Figure 2 shows that we have added support for Virtual Reality development.

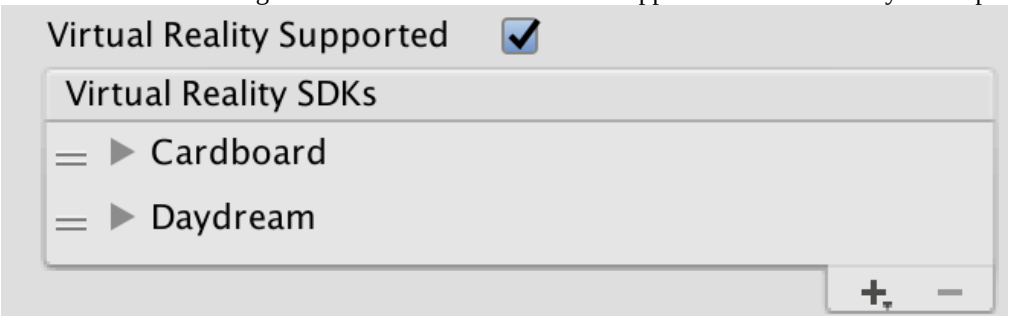

Figure 2. Virtual Reality Support added for Android devices

B. Movement in Virtual Reality

Player movement in the game was to be controlled by the head tracking on the phone and with the use of a Bluetooth Controller to move in all directions. To take input from the keyboard and various other peripherals we make use of the "Input" class in Unity [7]. By default, Input.GetAxis("Horizontal") and Input.GetAxis("Vertical") allow the player to use the WASD and arrow keys, a controller pad or other device to move the player. Movement in VR is one of the primary causes of VR sickness in users, and as such it requires careful consideration before implementing a solution in your project. [8] To set the player, we create an empty object in the scene and name it Player. We add a Character Controller [9] to this object and child this empty object with a Main Camera such that the camera is roughly at eye level for the player. The code for the Movement of the player in Virtual Reality space with the use of a controller is as shown below. This script is attached to the Player object.

```
using UnityEngine;
using System.Collections;
[RequireComponent(typeof(CharacterController))]
public class VRBluetoothController : MonoBehaviour {
   // VR Main Camera
   private Transform vrCamera;
   // Speed to move the player
   public float speed = 3f;
   // CharacterController script
```



```
    CharacterController myCC;
    // Use this for initialization
    void Start () {
        // Find the CharacterController
        myCC = gameObject.GetComponent<CharacterController>();
        // Find the Main Camera
        vrCamera = Camera.main.transform;
    }
            // Update is called once per frame
            void Update () {
        // Move with SimpleMove based on Horizontal adn Vertical input
        myCC.SimpleMove(speed * vrCamera.TransformDirection(Vector3.forward * Input.GetAxis("Vertical") +
Vector3.right * Input.GetAxis("Horizontal")));
    }
}
```

C. Interaction in Virtual Reality

In our project the way we have enabled interaction with objects is by using a Bluetooth controller. To set up the interaction we add a script on the object. Every Update() the script casts a ray forwards using Physics Raycaster [10] object on the Main Camera to see if the ray hits any colliders that might be present on the object we are looking at. The player can also get their cues to interact from the GvrReticle [11] object that is placed on the main camera. We also need to map the Joystick Buttons in the Input Manager before we can use them as references in the code. In our case, the string Fire1 corresponds to the A button on the controller. Figure 3 shows how the input is mapped in the Input Manager.

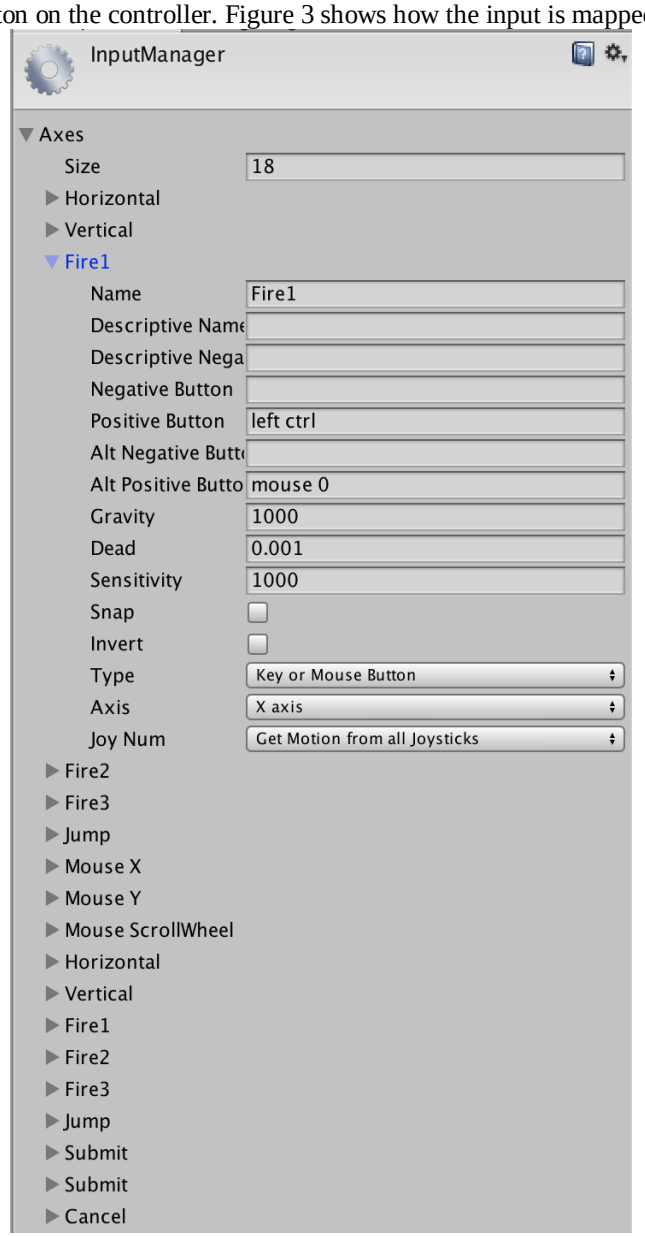

Figure 3. Mapping of Settings in the Input Manager

The Raycaster casts a ray against all colliders in the scene and returns detailed information on what was hit. This allows messages to be sent to 3D physics objects that implement event interfaces. The objects that require interaction to take place on them are fitted with Event Triggers to map the events that take place on raycast. Figure 4 shows the settings on the inspector panel of the object that need to be interacted with.

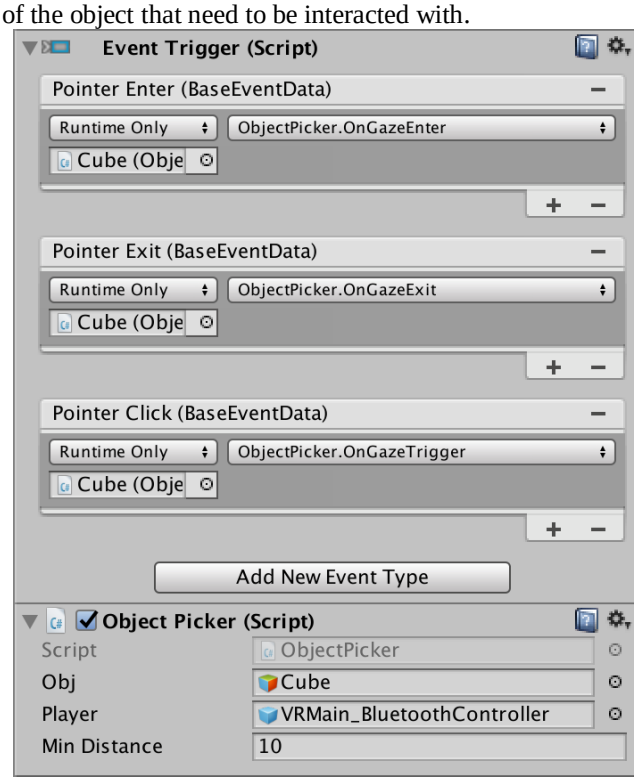

Figure 4. Settings on the Inspector Panel of the Object.

The following script demonstrates how the interaction will take place in the objects.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
[RequireComponent(typeof(Collider))]
public class ObjectPicker : MonoBehaviour {
	public GameObject obj, player;
	private bool looking = false;
	public float minDistance = 10.0f;
	private float distance;
	// Use this for initialization
	void Start () {
		looking = false;
	}
	// Update is called once per frame
	void Update () {
		distance = Vector3.Distance (player.transform.position, obj.transform.position);
		if (looking) {
			if (distance <= minDistance) {
				if (Input.GetButtonDown ("Fire1")) {
					obj.SetActive (false);
				}
			}
		}	GetComponent<Renderer>().material.color = (looking && distance <= minDistance)?
Color.green : Color.red;
	}
	#region IGvrGazeResponder implementation
	/// Called when the user is looking on a GameObject with this script,
	/// as long as it is set to an appropriate layer (see GvrGaze).
	public void OnGazeEnter() {
		looking = true;
	}
	/// Called when the user stops looking on the GameObject, after OnGazeEnter
```



```
        /// was already called.
        public void OnGazeExit() {
                looking = false;
        }
        /// Called when the viewer's trigger is used, between OnGazeEnter and OnGazeExit.
        public void OnGazeTrigger() {
                if (distance <= minDistance) {
                        obj.SetActive (false);
                }
        }
        #endregion
}
```

IV. CONCLUSIONS

Through the medium of this paper, we have aimed to create a conclusive solution for developers using which, they can add support for bluetooth controllers in virtual reality applications that they develop. The following figure show the game in action.

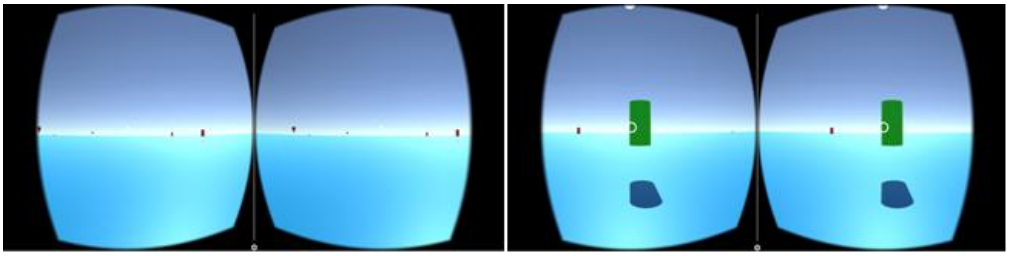

Figure 5. Starting the game and interacting with objects in the scene.

REFERENCES

[1] The Oculus Rift Website. [Online]. https://www.oculus.com/rift/
[2] The HTC Vive Website. [Online]. https://www.vive.com/us/
[3] The Playstation VR Website. [Online]. https://www.playstation.com/en-in/explore/playstation-vr/
[4] The Google VR Website. [Online]. https://developers.google.com/vr/
[5] The Unity 3D Game Engine Website. [Online]. https://unity3d.com/
[6] Documentation for GvrViewerMain [Online]. https://developers.google.com/vr/unity/plugin#gvrviewermain
[7] Documentation for Unity's Input Class [Online]. https://docs.unity3d.com/ScriptReference/Input.html
[8] Tutorial for Building Virtual Reality apps with Unity. https://unity3d.com/learn/tutorials/topics/virtual-reality
[9] Documentation for Character Controller. https://docs.unity3d.com/ScriptReference/CharacterController.html
[10] Documentation for Physics Raycaster. https://developers.google.com/vr/unity/reference/class/gvr-pointer-physics-raycaster#class_gvr_pointer_physics_raycaster
[11] Documentation for GvrReticle. https://developers.google.com/vr/unity/reference/class/gvr-reticle-pointer

