CIS568 Game Design Practicum

ASSIGNMENT #3: INTRO TO AUGMENTED REALITY

Due: Thurs., Sept. 29, 2016
(please submit to Canvas by midnight)

Assignment Overview:

The goal of this assignment is to enable you to develop a mobile augmented reality app in Unity. This assignment also covers cross-platform networking and using data from mobile sensors.

In this assignment, your goal will be to create an augmented reality paint ball game using Unity. You will be using a phone as a paint ball launcher and the computer display as a canvas.

The assignment consists of two parts. In the first part, you'll be using Photon, a cross-platform networking plugin, to connect your phone and computer; you'll be able to rotate a cube on the computer display using the gyro input from your phone. For the second part, you'll be able to create a paint ball mini game where you can use your phone as the paint ball launcher and your computer as the canvas. The paint ball mini game uses Photon for network connection between the phone and PC, Vuforia for the augmented reality part, and the data from gyroscope to keep the phone orientation in sync when the target is not in the camera view. The players will lower their phones to aim, and swipe on the phone screen to launch paint balls, trying to hit the canvas on the PC screen.

Assignment Details:

Part 1 – Rotating a Cube (40pts)

In this part, you will use Photon to connect the phone and PC, and use the gyroscope to control a cube on the PC screen. You will need to do the following:

a) Extract PaintBall-Framework.zip and open it as a Unity project. We have already imported the Photon and Vuforia plugins for you into the project framework, but you are welcome to try importing them yourself, or update them to the newest version. You'll need to get a Photon app id (please refer to https://doc.photonengine.com/en/realtime/current/getting-started/obtain-your-app-id), and enter it into **PhotonServerSettings** at *Assets> Photon Unity Networking > Resources*.

b) You'll need to work on **Cube_PC** and **Cube_Mobile** scenes for this part. Complete the cross-platform connection between phone and computer:

   In *Scripts* folder, refer to **PCNetwork_Cube.cs** and the "Networking with Photon" section of the document, finish **MobileNetwork_Cube.cs**. Build and run **Cube_Mobile** scene on your phone after running **Cube_PC** scene on your computer (using appropriate build settings), and make sure there is a "Joined" shown at the upper left corner of your phone.

c) Remove the cube object from **Cube_PC** scene and use **PhotonNetwork.Instantiate()** on **MobileNetwork_Cube.cs** to create the same cube at mobile client once joined the room.

Remember to add **PhotonView** to the prefab and add **Transform** to the observed components.

d) Rotate the cube using the gyroscope data from the phone obtained in **GyroController.cs**. You can refer to "Using Gyroscope in Unity" section below. If your phone is of similar orientation to the cube when launching the app, the cube on the PC screen should basically match the phone orientation when you rotate your phone.

Part 2 – AR Paint Ball Game (40pts)

Create your Augmented Reality paint splatter game using Unity, with the gyroscope input, Photon and Vuforia (see "Using Vuforia AR in Unity" section):

The computer screen as canvas, showing only an AR marker for the phone to locate the screen;

Once the marker has been located, the user can put down their phone to aim at the canvas and swipe the phone screen to launch a paint ball;

The paint balls are subject to gravity and physics. When the player holds their phone vertically again they should be able to see the flying balls they previously launched;

When the paint balls hit the canvas on PC screen, colored paint splatters should be created on the PC screen, at the point of impact (collision).

There are some provided resources you can use in the homework base project. You can create a basic paint splatter game by following the steps below:

a) You will need to operate on **PaintBall_Mobile** and **PaintBall_PC** maps. The scenes have been set up. But you need to copy some code from **MobileNetwork_Cube.cs** to **MobileNetwork.cs** and make sure the connection is working. You'll also need to obtain an app key for Vuforia and enter it at the **"App License Key"** field of **ARCamera** object.

When you use your phone camera to look at the target image on PC, a target mark should show up right on it by now.

b) Implement **TargetBehavior.cs**, which tracks the tracking state of Vuforia and switch the control of ARCamera between gyroscope and Vuforia. You can try adding a button to disable Vuforia input and see if your gyroscope works correctly.

c) At **MobileShooter.cs**, implement **ShootBall()** so you can shoot a paint ball upon swiping. You will need to use Photon's RPC function call to set the velocity, color and ownership of the paint ball.

The paint splatter ball game prototype should be working by now! But you always go one step further, don't you?

Part 3 – Improvements (20-40pts, up to 20pts extra)

Build upon the paint ball prototype you have, add **at least one** of the following features:

- This “game” is not exactly a game at this state, try adding some more rules and a goal to make this “game” a game.
    - You can even totally change the core mechanic, as long as there are AR, networking and mobile sensors involved - but be sure to make a backup of your part 2 implemention if you decide to do that.
- Don’t you want to share your canvas with others? Try enabling multiple phone to connect together, assign each phone a color or something special. You can even make this game a snowball fight if you implement both this and the idea above.
- Gyroscope is not very accurate sometimes, and it can only track rotation. Try improve the accuracy by introducing an extra sensor to make the control better. You can use compass to correct gyroscope and reduce drifting around y axis, and/or you can use accelerometer to update the position of the camera when Vuforia cannot find the target.
- Currently the paint gun launches paint balls with a constant speed, try improve this by changing the speed and direction of the ball launched based on player’s swipe. You can also make the control like a slingshot.
- The prototype doesn’t look nice, and there is z-fighting problem with the splatter texture. Try make the prototype look (and sound) better somehow!

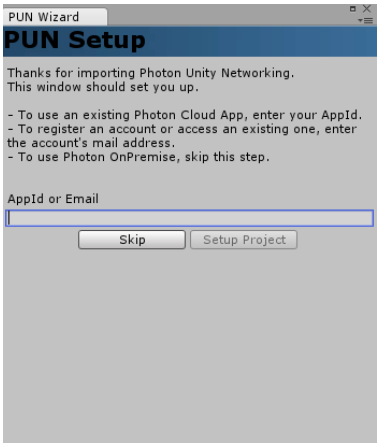
Part 4 – Submission

Have your project files, executables (for both PC and phone if applicable), a readme file and a short video (no .avi) in a zip file, upload it to Canvas by the deadline.

1. Networking with Photon

We will be using Photon to establish connection between phone and computer. Photon is a cross platform multiplayer plugin for Unity, and it is surprising easy to use. You can download Photon from https://www.photonengine.com/en-US/PUN or from Unity asset store by searching Unity asset store (by searching “Photon Unity Networking Free”).

The next step is to import the download package to your Unity project. When importing Photon to your project, you can uncheck “demos” folder to save space. A dialogue as below will pop up asking for an AppId for Photon. If you don’t already have one, you can get one for free using an email address. Then click “Setup Project” and close the dialogue.



The simplest way to make a connection between phone and PC using Photon is to use the "lobby" feather. To do this, we will be using the PC as a "Master Client", or host, to create a room in the lobby which is unique to the AppId, then our phone can simply join the room.

In Assets/Photon Unity Networking/Resources/PhotonServerSettings, enable "Auto-Join Lobby" so the application will join lobby upon launch.

We will use one scene for PC and a different scene for phones. Like you did before, you'll need a global GameObject in these scenes. To establish connection, you can attach a script like this to the global object of the PC scene:

```
public class PCNetwork : Photon.PunBehaviour
{
    void Start()
    {
        PhotonNetwork.ConnectUsingSettings("0.1");
    }
    void OnGUI()
    {
        GUILayout.Label(PhotonNetwork.connectionStateDetailed.ToString());
    }
    public override void OnJoinedLobby()
    {
        // Create a room for networking gameplay
        PhotonNetwork.CreateRoom(null);
    }
    public override void OnPhotonJoinRoomFailed(object[] codeAndMsg)
    {
        base.OnPhotonJoinRoomFailed(codeAndMsg);
    }
```

```
    public override void OnCreatedRoom()
    {
        base.OnCreatedRoom();
    }
}
```

And in the mobile scene, the code is similar, except that you need PhotonNetwork.JoinRandomRoom() to join the unique room at the lobby under your app id. And you will need to do something at OnJoinedRoom() for this assignment.

This is all about enabling Photon connection between phone and PC. When you are building the project, you should apply different settings for different scenes according to the type of your device (for example **PC, Mac & Linus Standalone** for your computer and **Android** for your phone).

When you run your PC client first and then run your mobile client, if everything was done correctly, there should be a "Joined" shown on the top-left corner of your phone and your computer. This indicates the connection between the two clients has been successfully established.

Once joined the room, you can use **PhotonNetwork.Instantiate()** to spawn objects over the network. In order to decide which data to sync between clients, you need a PhotonView component on the prefab to instantiate. You can drag components (such **Transform**) to the **Observed Components** tab of the PhotonView so the component data will be atomically synced to the network from the owning client.

For further information, you can see to the official tutorial for Photon by http://doc.photonengine.com/en-us/pun/current/tutorials/tutorial-marco-polo .

2. Using Gyroscope in Unity

With Unity, we can access data from mobile sensors easily. In this assignment we will be using the gyroscope.

We can use **Input.gyro** to access gyroscope data. It is an instance of the class **Gyroscope**.

Before we use the data from the gyro to rotate our cube or camera, we need the following code to enable gyroscope input and set the update interval:

```
Gyroscope gyro;
…
    void InitGyro()
    {
            gyro = Input.gyro;
            gyro.enabled = true;
            gyro.updateInterval = 0.01f;
    }
```

Once setup, it will automatically update its data from the sensor input. Now, you can access the following members:

gyro.attitude: the attitude (ie, orientation in space) of the device.

gyro.gravity: the gravity acceleration vector expressed in the device's reference frame.

gyro.rotationRate: rotation rate as measured by the device's gyroscope.

gyro.rotationRateUnbiased: unbiased rotation rate as measured by the device's gyroscope.

gyro.updateInterval: sets or retrieves gyroscope interval in seconds.

gyro.userAcceleration: the acceleration that the user is giving to the device.

gyro.attitude returns a quaternion which represent the current orientation in the device's reference frame. However, although Unity allow you use the same code on difference devices, **gyro.attitude** of difference devices may have different coordinate systems and you need to manually convert it to Unity coordinate.

When testing on an Android phone, **gyro.attitude** was of a right-handed coordinate system while Unity is using a left-handed coordinate. We can use the following code to convert it:

```
Quaternion ConvertRotation(Quaternion q)
{
    return new Quaternion(q.x, q.y, -q.z, -q.w);
}
```

To make the rotation in virtual world in sync, we can save a reference rotation R in virtual world (eg. The initial rotation of the rotating cube for the first part, or the rotation of the ARCamera when Vuforia find the target for the second part) and the corresponding gyro data R_g from **gyro.attitude**. When we have the new gyro rotation R'_g, we can get the local rotation difference ΔR and new virtual object rotation R' by:

$$\Delta R = \left(R_g\right)^{-1} R'_g$$

$$R' = R\Delta R$$

3. Using Vuforia AR in Unity

Vuforia is an AR platform that is very easy to use in Unity. We can download the Unity version of Vuforia SDK at https://developer.vuforia.com/downloads/sdk.

Once downloaded, import the package into your Unity project. You will need to register an account and get a free license key. Then you can create your custom target database and image target. Then you can download them as a Unity package and import into your project.

Once you imported the packages, you can put an ARCamera and a ImageTarget from Vuforia folder into your scene. After you set the **Database** and **ImageTarget** in the property of the ImageTarget and the license key in ARCamera, you can put any 3D geometry above the ImageTarget object. Once you run the application, it will automatically use your device camera to sense the marker, and you should be able to see your geometry above the marker once

successfully tracked the target. You can see to https://www.sitepoint.com/how-to-build-an-ar-android-app-with-vuforia-and-unity/ for a more detailed tutorial.

Vuforia automatically adjust the location, orientation and projection of the ARCamera by the calculated position and orientation based on the view of the marker in the device camera. In the final part of the assignment, you will need to be able to keep the orientation in sync while putting down the device and swipe on the phone screen to launch the paintball. You'll need to switch the controller of the virtual camera between Vuforia and gyroscope.

To do this, you can monitor the tracking status of Vuforia in your ImageTarget object by adding a script with **ITrackableEventHandler** interface and listen to the **OnTrackableStateChanged()**. You can create your code based on **DefaultTrackableEventHandler** class in **Scripts** folder of Vuforia, which is loaded as a component of ImageTarget prefab by default. Once tracked the marker, switch the control of ARCamera to Vuforia by the following code:

```
TrackerManager.Instance.GetTracker<ObjectTracker>().Start()
```

And

```
TrackerManager.Instance.GetTracker<ObjectTracker>() Stop()
```

when the marker is not tracked and use gyroscope data to manually control the rotation of the ARCamera.

Now you are good to go with your paint ball AR game. Happy coding!