ECE1778 Project Report

Kathy Au Billy Yi Fan Zhou
Department of Electrical and Computer Engineering
University of Toronto
{ kathy.au , billy.zhou }@utoronto.ca

Executive Summary

The goal of this project is to implement a multiplayer shooting game. Players are placed in a world where game contents are overlaid upon the image captured by the phone's camera. This altered version of the reality will be displayed on the screen, allowing the player to be immersed in a virtual reality world while fully aware of their surroundings. Our game pits multiple players against each other in a target shooting match. After both players join the game, targets will be randomly generated in the shared virtual space, and each player will rotate their phone to aim at targets and destroy them.

1 Introduction

This section introduces the motivation behind our game application and gives a brief overview of its major functionalities.

1.1 Motivation

Smartphones are equipped with multiple sensors and network connectivity. These capabilities allow the device to interact with the user, the environment, and the Internet simultaneously. It is up to application developers to realize the full potential of these phones by making good use of the available hardware such as the camera, accelerometer, gyroscope and GPS. A recent study of Apple's App Store showed that the game category has the second highest number of applications. Our team of two programmers developed a multiplayer shooting game application that uses the myriad of sensors on smartphones to offer a multimedia gaming experience to Android users.

1.2 Overview

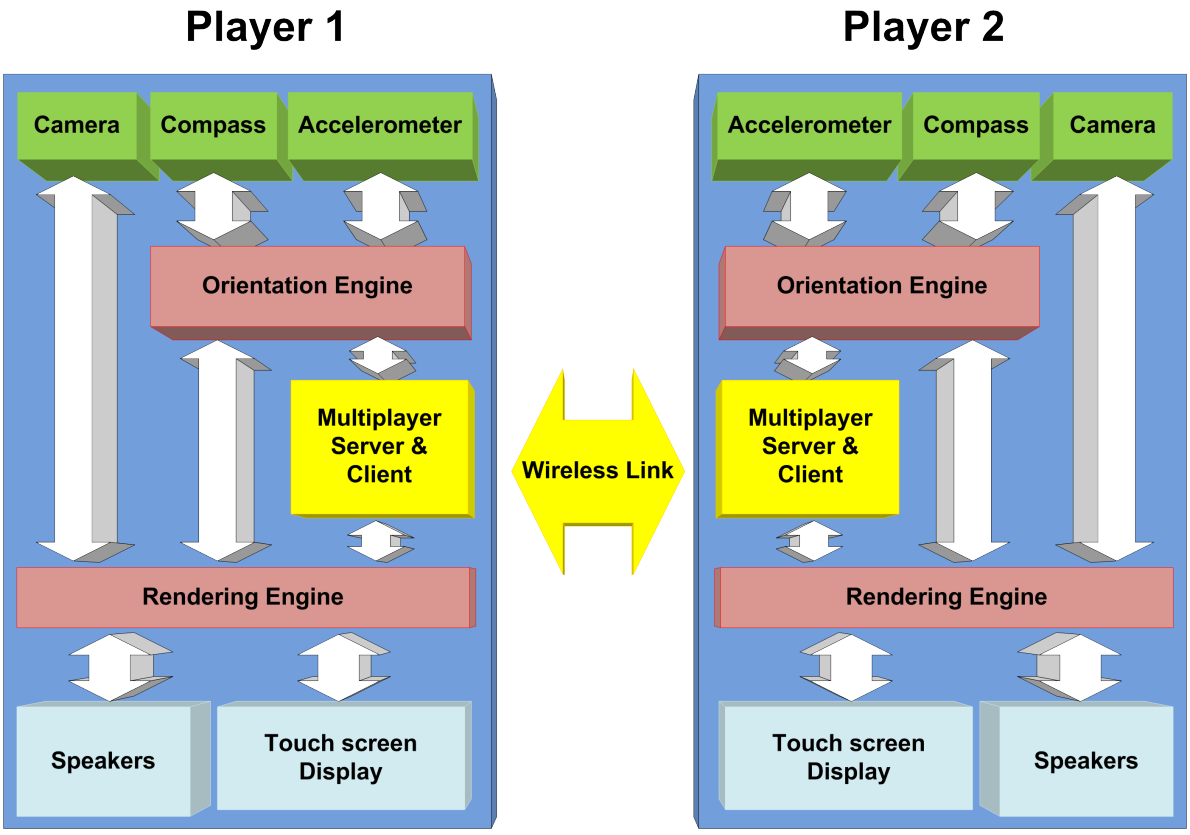
We propose a multiplayer shooting game in virtual reality. A 3-dimensional virtual space around the user will be used as the stage for the game. Both players are immersed in the same virtual space where they are located at the center. The rear-facing camera will be used to obtain the scenery behind the phone to create an illusion of seeing through the phone for the user. Once a game is started, targets will be randomly generated for the game. These targets are overlaid on the screen as if the phone is pointing at it in virtual space. As a multiplayer game, the players are to race to eliminate the most targets. If a target is eliminated by player 1, it will disappear from player 2s screen as well. The game is finished when all targets are eliminated or the time runs out. Statistics of the current game session will be presented to both players at the end.

The touch screen is used to navigate the user menu. The camera, the screen, the accelerometer and the compass will be used to support the VR aspect of the game. The phone's orientation obtained from the sensors will be used to aim within the virtual space. And as discussed earlier, the camera deals with overlaying virtual targets on the display. Audio output will be used to provide in game feedback to the players.

Our games achieves multiplayer capability by having the two smartphones communicate over Wi-Fi. The phones must be connected to the same Wi-Fi hotspot for this happen. Wi-Fi enables a stable low latency communication link that is not otherwise possible via the 3G network. Wi-Fi also has the side advantage of not incurring any costs to the players.

2 Detailed Description

This section outlines the major components of our game and the data paths between the components. Then each subsection will focus on the individual components and describes its functionalities in detail. The following block diagram gives an overview of all the components in our application.



2.1 Orientation Engine

The orientation engine takes the magnetic readings from the compass and the force readings from the accelerometer to calculate the current orientation of the phone. This orientation is then used to determine location and direction the player is facing within the virtual space. The orientation engine is divided into two parts: the first portion calculates the inclination (deviation from the horizontal plane) of phone given the accelerometer readings, while the second portion calculates the azimuth based on the compass readings and the current location of the phone.

2.1.1 Inclination Sensing

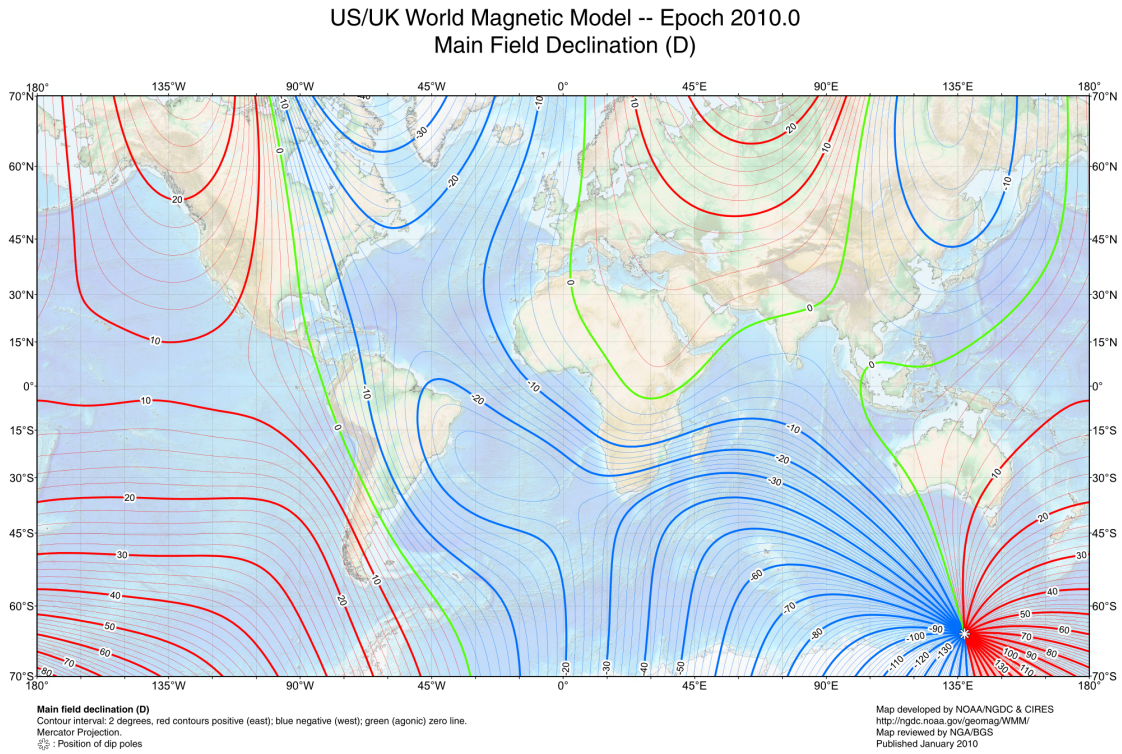
This half of the orientation engine takes the accelerometer readings and calculate the inclination of the phone, which correspond the θ axis in the spherical coordinate system. We observe that if the sensor is perfectly vertical it will experience a gravitational pull of exactly $1g$. When the sensor deviates from the z axis by an angle θ, it will experience an acceleration of g', which is less than $1g$. In fact, we can precisely calculate the angle of deviation θ using equation (1).

$$\theta = cos^{-1}(\frac{g'}{g}) \tag{1}$$

Thus we register this class as a `SensorEventListener` to incoming accelerometer readings and apply equation (1) to each new reading as they arrive. The results are then sent to the rendering engine for further processing.

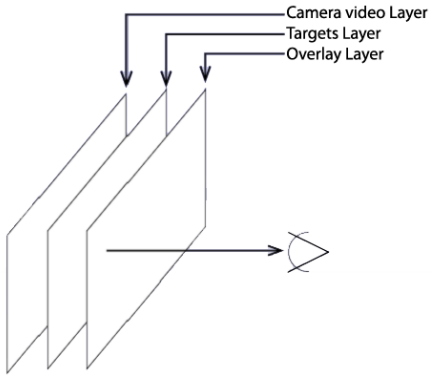
2.1.2 Azimuth Sensing

This part takes the magnetic readings from the compass, and applies a correction factor derived from the World Magnetic Model (specifically WMM-2010) to calculate the orientation of phone with respect to true north. This corresponds to the φ angle in the spherical coordinate system.



2.2 Rendering Engine

The rendering engine consist of several `Renderer` class that each render a specific layer of the game. This modular design allows us to rapidly prototype new functionalities without breaking existing rendering layers. All the layers are rendered sequentially so that the later layers will overwrite the the results of any previous layers.



The following subsections will describe each `Renderer` class that is responsible for rendering a specific layer.

2.2.1 Camera video layer

The first layer is the camera video layer. This layer simply display the current image that is being captured by the smart phone's camera, so that the players will think they are "looking through" the smartphone. This layer serves to inject a sense of reality into a game that consists of purely virtual targets. This enhances the virtual immersion experience and also helps the player to see the actions of their opponents.

2.2.2 Targets layer

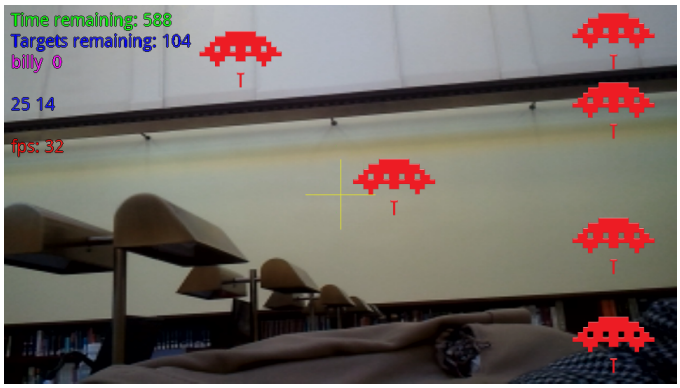
This rendering layer handles the display of targets inside the virtual space. Each target is individually rendered by superimposing the the target `Bitmap` object over the canvas. In single player gaming mode the targets are static sprite in the shape of alien ships. However in multiplayer the targets are the avatars of the opposing players to make the gaming more personal and competitive.

Since the smartphone screen can only view a small window of the entire spherical coordinate system at once, we perform the optimization of transforming the coordinates of the visible targets, as opposed to all targets. the number of visible targets can be determined by the current viewing vector of the phone $\vec{V}$ and the viewing angles of the phone. The two viewing angles $\Delta\theta$ and $\Delta\varphi$ determine the how "wide" the views in the θ and φ axis respectively. Thus the "visible" portion of the spherical coordinate system is simply the range of $(\vec{V}_\theta \pm \Delta\theta, \vec{V}_\varphi \pm \Delta\varphi)$. It follows that only the targets that fall within this narrow range needs to undergo the remaining transformations and be displayed on screen.

2.2.3 Overlay layer

This layer act as a Head Up Display to provide the player with vital information such as the cross-hair, targets count, and the name and score of their opponents. This layer is rendered last because these information are critical enough to overwrite the two lower layers. Using modular programming techniques, we separated each sub-layer into its own class, all of which extends the `Renderer` class. This way we can switch each individual sub-layer on and off, depending on the setup of the game. For example there is no need to render opponent's laser beam in single player, therefore its associated `Renderer` class, the `RemoteLaserRenderer` class does not need to be initialized during single player loading.

2.2.4 Rendered Screen



The screenshot above shows the combined view of the three layers produced by the rendering engine in single player gaming mode.

2.3 Multiplayer Communication Engine

This section describes the `Communication` class that is responsible for connecting two devices over a network for a multiplayer game session. The two major components inside this class are a server thread and a client thread. These communication threads run in the background. They interact with static game state data structures and send messages to the main UI thread.

In multiplayer mode, there is a host player and a guest player. The host player is the player that received the connection request. Host is responsible for game initialization and communicate target locations in the virtual space to the guest player. Both players will have a server thread and a client thread running during the gameplay.

2.3.1 Multiplayer Server

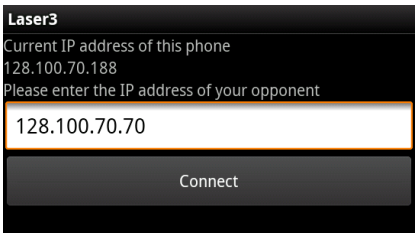
The `Server` class is a background thread that opens and listens to a port. This thread is started as soon as users select multiplayer from the main menu. When a connection request from a client is received, a TCP connection is established for ongoing communication throughout the game. This class is designed to handle special `Laser3Packet` objects. The server descides on the action items based on the type of packet received. Some available actions involves updating remote player's profile information, launching the main game screen and removing targets destroyed by the opponent.

2.3.2 Multiplayer Client

The `Client` class is also a background thread that constantly polls a send queue for packets to be sent to the server. During a multiplayer game, any actions that needs to be communicated to the other player are represented in the form of a `Laser3Packet` object that is stored in the client's send

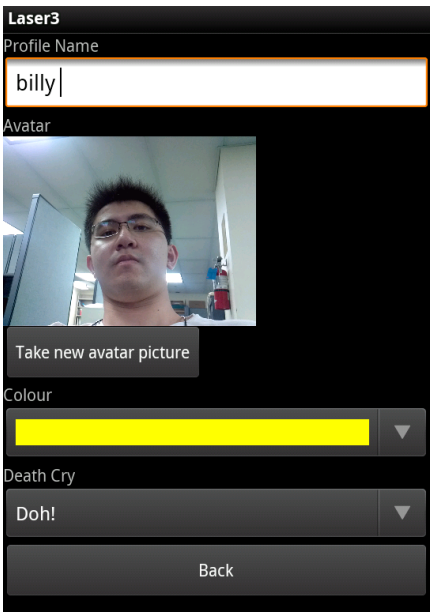
queue. Example of gameplay messages include target hits and pointing vector updates. It is important to note that pointing vector updates are sent periodically to provide the opponent with a real time view of the player's aiming action.

2.3.3 Game Initialization Flow



This screenshot shows the connection screen for multiplayer gaming. Connection is established by entering the opponent's IP address. Since only IP address is required for a game session, connection can still be made if two devices are connected to the same router without actual Internet access.

Once a connection is established between host and guest player, their profile information containing their name, avatar, colour preference and death cry is exchanged. The following screen is accessible from the main menu where players can customize their profile in advance.



As mentioned earlier, the host player will randomly generate targets in the virtual space based on his difficulty and timeout settings. These game initialization data is sent to guest player who will setup the game. When the guest player finished setting up the game, an acknowledgement will be sent to the host player that the game can be started at this point.

3 Contributions

This section describes the contributions of each team member.

3.1 Kathy's Contributions

I developed the multi-threaded communication engine in this application. I wrote all the code associated with connection establishment, game initialization and game state synchronization. I developed the profile and options pages which are used to change in game settings. I was also in charge of implementing the backbone of the `PlayerProfile` class used to hold both local and remote player's profile.

3.2 Billy's Contributions

I developed the orientation engine as well as the rendering engine. I wrote all the individual `Renderer` classes with the exception of the `RemoteLaserRenderer` class which was developed by Kathy since it is associated with the multiplayer mode. I developed the player profile picture system which enables players to shoot at avatars of their opponents instead of static sprites. I was also in charge of implement the sound effects of the game.

4 Future Work

Due to the limited time frame available, there are a number of additional features that we would like to highlight as some possible future work.

The success of our game depends heavily on the ability of the orientation engine to pick up an accurate orientation of the device. However, after some experiments we found out that the compass's readings is very unreliable especially when around many electronics. This is a limitation of the hardware and we would like to experiment with built-in gyroscope available on the high-end phones.

Also, currently multiplayer mode connection depends on knowing the opponent's IP address. Since IP address is cumbersome to remember and it is also hard to find players, a future improvement of our application would involve building a web server that can handle game creation and joining game. This should allow users to play against anonymous players connected to the Internet.

Since significant effort was focused on the backend work during the project time frame, this result in a rather primitive user interface. Some extra time should be devoted to polishing the UI to make the game more attractive as a commercial product to end users. In the future, we would like to submit our game application to the Android Market, since we have a game application we would like to expose it to as many audience as possible.

5 Reflection

We learned a lot about programming in Java on the Android platform. In particular, we learned about socket programming in order to implement the communication engine. Also, we learned about different rendering techniques using the canvas class. In terms of hardware, we discovered that we shouldn't rely too much on the compass sensor, as its readings often fluctuates a lot. We were able to set goals that follows the spiral model where we first worked on rendering the screen and obtaining orientation of the phone and then we incrementally build the single player and subsequently the multiplayer mode. Since our game involves physically moving the phone, it makes live demo of the application a challenge with a static camera. What we would like to do differently in the future is to investigate ways of displaying a phone's screen directly on a project screen. In conclusion, we found that it was a fun project to work on aside from our own research projects and it really gives us the background knowledge we need to start creating our own mobile applications.

Total word count: 2313