AN APPLICATION OF AUGMENTED REALITY ON EVERYDAY GAME SYSTEM

David Hu, Eric Hsin, Wei Fang

Department of Electrical Engineering
National Taiwan University

b01901133@ntu.edu.tw, b01901014@ntu.edu.tw, b02901054@ntu.edu.tw

Abstract

Slither.io has been a trending game among students, a simple but addictive piece of web gaming. Also trending, virtual/augmented reality has taken the world of technology, including gaming, by storm. In the 2016 Electronics Entertainment Expo, or E3, VR and AR has been the new yet the eminent competition of the year. The exponential growth of interest has validated the promising years to come. While the market tiers up applications and hardware, we look to ride the wave of popular technology. Using common and well-known development tools such as Unity 3D and Vuforia, we transform a game as old as "Snake" into an application set on course of future gaming, and simply much more fun.

1 Introduction

Slither.io[1] is a web page based, multi-player combating on-line game. The game has been widely played and gained a lot of popularity recently, and it is not surprising that such a renowned game would have a its mobile device version on several platforms including the mostly used iOS and Android[2] [3]. While having its own APP, the control system of Slither.io on mobile device hasn't been well designed. The basic moving operation of the character is done by touching certain positions on the screen while such action will cause severe disturbance to players' sight. Modification on its control mechanism is required.

Augmented reality is an ideal tool to achieve the goal. Taking advantage of manipulating the interaction between reality and virtual objects, augmented reality allows player's to control the character without touching the screen. Also, a game based on reality scene is more attractive to the players since we are living in an environment

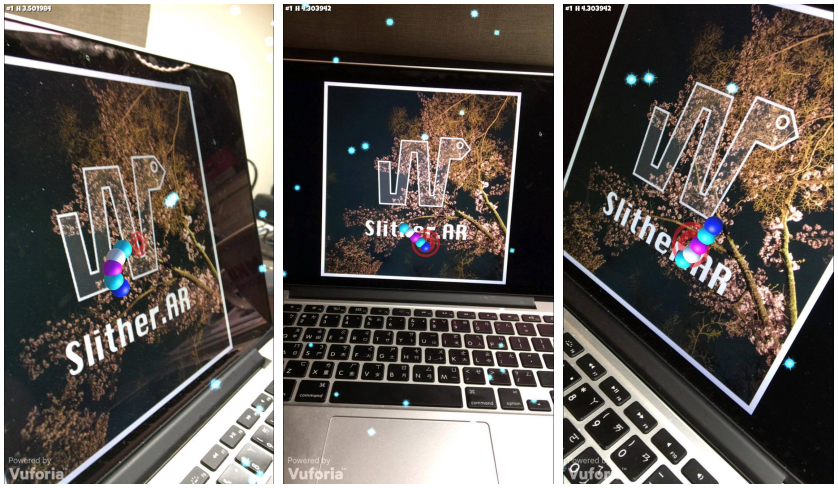

Figure 1: *Slither.AR* gameplay example. The game character stays on the game plane regardless of change of camera view angles.

with real objects. We call the game *Slither.AR*. The implementation is available at `https://github.com/unieric uni/slitherAR`.

2 Game Mechanism Design

In order to preserve the joy brought by the original game, we tried to imitate most of the basic game functions of Slither.io. Consequently, the game Slither.AR is composed of two basic items on the battle field, food and snakes. Snakes are the characters controlled by players, whose ultimate goals are to eat food and do their best to kill the other snake. Each snake can perform two basic movements: moving and dashing. The character controlled by a player will move in a certain direction specified by the player's cursor. Once a character has collected enough food, the player is able to make their snake dash with higher speed for a period of time at the cost of score accumulated by eating food. In addition to gaining more score, food can also help snakes grow larger.

A snake will not die until its head collides with the body part of another snake. Self-collision will not cause any damage. The game ends at the moment the player's snake is dead. The corpse of a killed snake will soon become food containing enormously high score. The larger the dead snake is, the more food will be generated.

The main difference of our design lies in the control system. In the original web page-based game, a player specifies the direction of a snake by moving his/her cursor. As mentioned in the previous chapter, using finger touching instead of cursor will result in severe disturbance to the gaming experience. Thus, in replacement we use AR (augmented reality) to control the character. An AR mark, such as a binary graph or an image with enough details to be recognizable, is viewed as the game plane in the camera on a mobile

Figure 2: Overall game architecture

device. Both snakes and food will be generated on the plane after successful recognition. Using AR technology, the game items will stayed on the AR mark plane even though the camera change its view angle or distance.

In the new design scheme, we try to control the destination of a snake instead of its moving direction. The destination is specified by a virtual ray-cast shooting form the camera along the normal vector of the camera plane. The snake controlled by the player will move toward the intersection of the ray and the game plane once it goes certain distance from the intersection. The snake will remain on the same speed even if it meets the intersection, and will turn its head once exceeding the distance again. Such moving rules will ensure the player to have a pleasant gaming experience while the snake constantly moves. For simplicity, finger touching on the screen will make the snake dash. The following sections describe the components of our design, and the overall game architecture is illustrated in fig. 2.

3 Basic Gameplay

The game was implemented with the game engine Unity[4] and then delivered to the iOS platform. This section describes the basic gameplay design.

3.1 Movement

The head of the snake moves forward with a constant speed. It can only change its direction by rotating its forward vector. The direction of rotation, as mentioned in the previous section, is the direction towards the virtual cursor, but it is clipped if it exceeds

a maximum rotation angle. The sensitivity of the rotation decreases as the snake grows, preventing a single snake from being too dominant such that it possesses great turning speed while being able to circle others.

3.2 Growth

As in the original game, the snake grows longer as it consumes more food. Each time the snake's head collides with a food, it creates a new body part at the end of the snake. Each body part has a internal list of previous body parts and the snake head. It follows the path of its previous body part, smoothed with a spring-damper like function that does not overshoot. As the snake grows longer, it also scales up when it reaches certain pre-determined lengths.

3.3 Food Generation

Food is spawned randomly in the playing field every 0.5 seconds. Three kinds of food with different scores are spawned by sampling a distribution. The score of each food object decays exponentially while it is not consumed by a snake.

3.4 Dashing

The snake can also perform *dashing* when the player taps the phone screen, increasing its speed at the expense of its own length to try to defeat other snakes or get to food with extremely high scores. Once the snake starts dashing, it will lose a body part every 0.5 seconds, starting from its last body part. The lost body parts would turn into food objects with a low score, and will be placed at the position of the last body part. The scale of the snake will also decrease as it becomes shorter.

3.5 Collision

To defeat other snakes, the player has to either position its body in front of other snakes' heads, or circle the opponents completely so that they will collide into the player's snake eventually. When a snake's head detects a collision with another snake's body, the entire snake turns into food with extremely high scores, and the player exits the room and returns to the lobby.

4 Network Connection

4.1 Host Model

We used the Photon Unity Network (PUN)[5] for our connection to enable multi-player utilities. PUN is server-client based, and the game is hosted on a regional, dedicated cloud server. As for the hierarchy, a cloud servers hosts rooms, a room hosts a game by a master client.

4.2 Synchronization

To make a game object network aware, means of gates and protocols for communications are necessary. In PUN's case, "Photon View" is attached to an object, so that it is visible and is enabled to streaming.

A couple of synchronization methods are implemented upon Photon Views throughout the implementation of our on-line game:

Object Synchronization (serialization): A simple data streaming queue. A Photon View and its remote duplicates synchronize on a self-defined rate around 20 times per second; it either en-queues some data updates or de-queues buffers that are sent to the local client.

Remote Procedure Call (RPC): A Photon-View-owned function. A client broadcast the RPC function via Photon View to its targeted remote duplicates.

There are a lot of data and states to be synchronized in our game. We look at the update frequency to decide how we synchronize. For frequent, visual updates such as position, rotation and appearance, serializing with specific data stream is fast and simple. For game logic and states, which are more tolerant of latency than actual game-play, we use RPC to suffice the need of calculation and action of specific client.

4.3 Game Maintenance

Game objects are usually instantiated on the network rather than locally. Most objects aren't owned by a certain player's client, so the master client is responsible for these objects. This is where RPC comes in handy, we can send the updates to the master client only, therefore reducing network load. Take our game for example, the master client is in charge of spawning and consuming food, updating scores, and ranking scoreboard. The master also handles player entrances and exits including itself. Note that the master client assigns its successor when it exits the game.

5 Augmented Reality Control

We implemented the AR control system described in the second chapter with a Unity package called ***Vuforia*** [6] [7], developed by *Qualcomm*. The package enables us to have basic AR manipulation with an object called AR camera and an object called target image, which is the AR mark mentioned in previous chapters. The game objects attached to the image target object will hold their relative position to it while changing camera view angles. The food and snakes are both spawned on the AR game plane held by the target image. The destination and alarming boundary are defined by two rays shot from two invisible game objects attached to the user screen, which is the $y = 0$ plane in terms of the global coordinate system. One intersection with the plane resembles the snakes' destination, while the distance from one intersection to another represents the radius of the alarming region. Once a snake exceeds its alarming region, the game system will add on a certain amount of speed, whose scale is determined viewing the relative position of the destination and the snake, normal to the moving direction. Using such manipulation, the snake will gradually turn its head toward the specified destination instead of performing an unnatural, sudden movement.

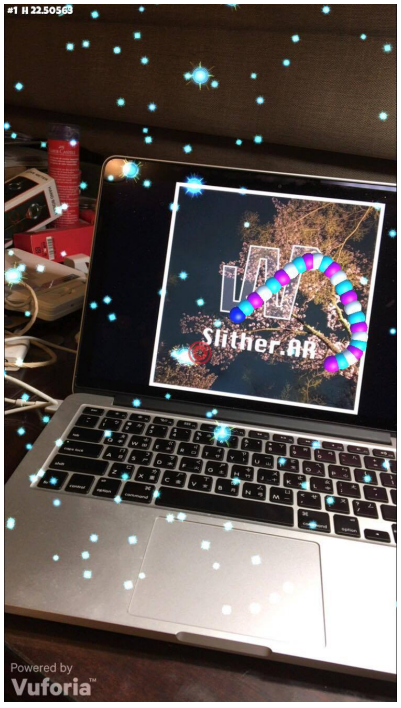

Figure 3: A screenshot of the implemented game. In this picture, a smooth-angled turning is performed by the snake, whose track can be recognized by its following body parts, toward the circled destination.

6 Results

A screenshot of our implemented game is shown in fig. 3. The image shown on the laptop screen in the background is the AR tag, which has to be in the camera view at all times. The red doubly-circled mark in the center of the screen is the intersection of the virtual ray emitted from the camera and the gameplay plane, which controls the destination of the snake. Snakes are modeled with colored spheres that are connected like a chain. The blue sphere, representing the head of the snake, moves toward the intersection mark, and the body parts follow the path of the head. As mentioned in the previous section, once the snake overshoots the destination and exits the alarming region, it performs a smooth u-turn, as shown in the figure. The glowing objects in the screen are food objects, with the larger ones representing those with higher scores. Finally, a scoreboard is shown in the upper-left corner, showing the top 3 players with the longest length.

7 Conclusion and Future Work

In conclusion, the game is implemented perfectly and, plus, the problem of the control system in original design is optimized using AR technique. The optimization allows players to move their snakes instinctively to the destination specified by the front cameras on mobile devices. In addition, the gaming experience is enhanced by real scenes instead of animated background.

However, there is still one remaining problem. Since the destination of a snake is specified by the camera on a mobile device, players cannot move their hand casually when holding the mobile device. A casual movement of the hand will lead to unexpectedly moving of the snake or lost of AR mark, while holding the mobile device steadily for a long time is tiring and inconvenience for a player. An elegant solution might be the detection of large variation in the relative position of AR mark and front camera in unit time. We will try to implement this idea in the following updated system, and we believe that the control system of Slither.AR will be more instinctive and user friendly.

References

[1] slither.io: `http://slither.io/`

[2] slither.io on App Store:
`https://itunes.apple.com/us/app/slither.io/id1091944550?mt=8`

[3] slither.io - Google Play Android:

https://play.google.com/store/apps/details?id=air.com.hypah.io.slither&hl=zh_TW

[4] Unity: http://unity3d.com/

[5] Photon Unity 3D Networking Framework SDKs and Game Backend: https://www.photonengine.com/en-US/PUN

[6] Vuforia: http://www.vuforia.com/

[7] Vuforia Developer Portal: https://developer.vuforia.com/