BeeBarber, a combat action VR game with real-time hair simulation and elements of Cthulhu horror

Diyuan Dai
The University of Texas at Austin
diyuan.dai@utexas.edu

Jiaxi Chen
The University of Texas at Austin
jiaxi312@utexas.edu

Abstract

BeeBarber is a VR gaming application to let users experience being a bee flying around an artificial head and doing a haircut. The application has a sophisticated hair simulation system that mimics the interaction of normal and cut hairs. Additionally, BeeBarber incorporates various gaming features, including a health bar system, saber enchanting, execution mode, and dash-to-dodge system to make the game entertaining.

1 Introduction

There are two overall goals we want achieve through BeeBarber. Firstly, hair simulation is always an interesting as well as challenging topic. Significant achievement of hair simulation can be seen from the movie. However, not much existing work has been done about lifelike hair simulation in AR/VR. Therefore, We attempt to develop a hair simulation system in VR device. Though the simulation process is extremely hard, we can learn a lot about computer graphics and physical simulation through this project. The main challenge of is that we have to learn a lot of concepts about hair simulation. More importantly, existing work of hair simulation cannot be completely applied to our project due the limiting computing power of Quest devices.

The second goal of our project is to make a VR game. Games are always our favorite. However, we never experienced gaming in virtual reality. Making a VR game would require to use the Unity platform and other gaming and VR libraries which is a great experience for gaming development. The hard part of making a VR game is to familiarize ourselves with the developmental pipeline of using Unity and VR libraries. Also, 3D game is different than simple 2D game. So, more works must be done to make the game run smoothly. While creating this game, we have to consider the limitation of the Quest device as well.

2 Related Works

When the idea of hair cutting thing came into Diyuan's mind, we did not expect to have so many existing VR Barber projects, but we are glad to find out that most VR project is low-poly and most physics are done volumetrically. There has been no real-time simulation of hair in VR yet, so we decide to make our own. Besides hair, we add some combat, some pose detection, some horror elements, and some Cthulhu.

There are some games where we find some inspiration, including the first VR game most people played, Beat Saber. Beat Saber shows us the potential of user interaction for entertaining purposes. Most video games previously are based on pressing buttons; it is a precise way to interact but not immersive at all; then we have gesture controlling or pose detection controlling games. BeeBarber relies heavily on pose detection, and it requires the player to act like playing a saber, flying with their "wings" and dodging with their heads. For other sources of inspiration for level design and battle design, please refer to Chapter 3.6.

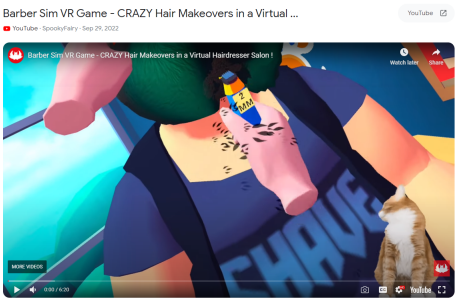

Figure 1: VR Barber Shop

Figure 2: Beat Saber

One public package we use is Ezy-Slice, it is an open-source slicer framework for the Unity3D Game Engine, which provide us with the ability to slice any convex mesh using a plane. We use them in slicing objects with collision, but not for hair. For hair, we wrote our own line collider with unity build-in functions.

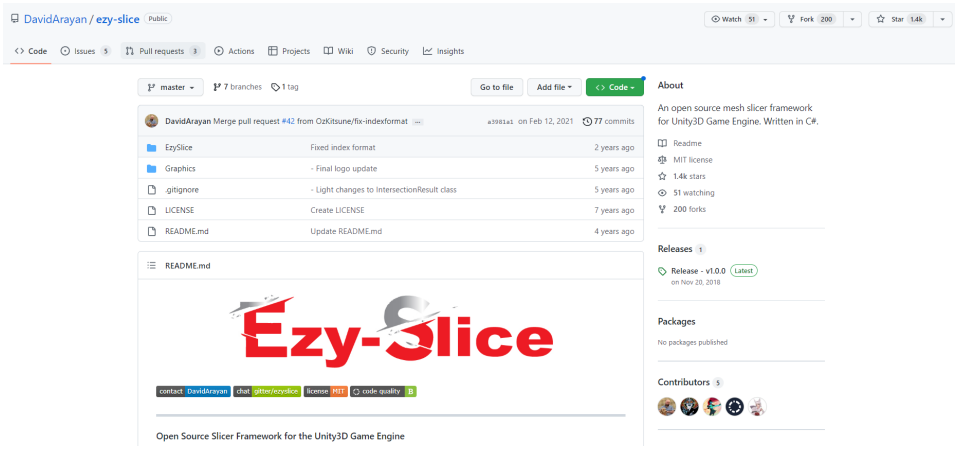

Figure 3: EzySlice

3 Technical Details

3.1 Project Repo and overview on game flow

Demo video here:
https://drive.google.com/file/d/1dkHfUWbQn9vZoyk5kHHiuLTyypodwJ7E/view?usp=sharing

Project Github Repo:
https://github.com/DD-1111/BeeBarber

Please watch this demo video to see the experience of a gamer playing the game smoothly. The story and mechanism will be explained in the video, and also there is a demo for hair simulation.

3.2 Hair

3.2.1 Basic data structure

In this project, hair is comprised of particles. To avoid the massive usage of computing resources, we decided to avoid building collisions for hair and mesh renderers of the cylinder. Instead, we use a line renderer, which is a built-in render that takes in positions of points and renders a line between positions. Because we need to implement the cutting feature, we have to make our minimum unit particles and their connected particles' pointer. It means we need to create a line renderer, springs,

and line collider for each particle so that we can disable one of the particle connections to cut things down. We only have one code file for the "hair" scope, which is our configuration file, which is in charge of instantiating the hair segments and setting the parameters like spring force, decaying factor, and damper of each segment within this piece of hair. For the "particle" scope, it looks like a LinkedList; each particle knows which segment to connect with, and there is a piece of code that takes care of the updating line render's position. There are two strings attached to one particle, a hand-written line collider that takes care of trigger separation (see next paragraph), and a script to deactivate the particle once it gets cut and falls onto the ground for a while to save computation. We finally decided to use the built-in joints and rigid body system because the built-in APIs are easy to interact with each other and are well organized in structure. Most importantly, we eventually need the built-in physics engine to take care of collision and visual effects.

3.2.2 Forces

There are two strings attached to one particle, one is for the connection to the next particle, and another one takes care of the bending force. For bending force, I used a configurable joint that can disable the twisting angular motion to stabilize the hair. In configurable joints, we can set a string with a customizable anchor. If we set the anchor on its main axis, so all the particles will have a bending force toward the center. Gravity and damper are the built-in trivial forces. To make the hair a perfect customizable curve, I also implemented a decay factor so the forces would have a different impact according to the position of the particle in the hair. I implemented noise by adding random mass to particles, and for some random particles, we also enabled the sphere collider to act as a hair-spray force.

3.2.3 Cutting and collision

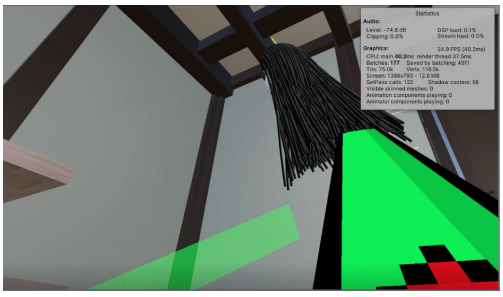

Figure 4: HP bar as the saber

Figure 5: Hair being cut

Since this is a VR project, I have to be very careful of the computation amount. Though we know there is still a huge space for optimization improvement, we show that doing heavy computing work on VR headsets is a somewhat bad idea. Collision detection has always been a big problem because of the complexity between every object and the high cost. To solve the collision detection burden, I first try to use the layer mask so that certain layers of the object will only count in specific layers for physical reactions. I tried to disable the interaction between hair so that it can still interact with the other objects like desks and sabers, but it seems that unity will calculate the collision first and then apply the filter. Then we came up with an idea, which is to give particles collision components once they get cut. The logic is that:

1. We first detached the connected particle's springs and destroyed the original line renderer.
2. Then, for this particle and each particle after this one, we apply sphere collision to it so it falls onto the ground.
3. Finally, add a dummy copy of the cut particles and attach it to the originally connected strings as a substitution.

However, this makes the cutting feature a problem since we do not have a collision of particles and segments any longer. We found out there is a built-in line cast function that takes in two positions and checks whether there are objects in-between. We add some logic and modifications to it, and it

works perfectly for our segment collision with the saber. Also, this is where the message triggers the controller's vibration from.

3.3 Motion and Pose Detection

The motion/pose detection interaction is the most innovative part. Using only a thumbstick to move and two buttons to take control of jump/fly is boring, and his traditional controlling method is inherited from a joystick, not utilizing any features of VR. So Why are we using VR? Because of the unique inversive experience! How can we have everything controlled in virtual reality by just pressing buttons? In this project, we implemented two pose detections, flying and dodging:

Flying is triggered if the player flaps his "bee wings." The way how Oculus controller is designed makes it practicable since they provide us with the direction vector and position vector. Suppose we check the rotation and position of every frame. In that case, we are able to detect any motion within the controller theoretically since the calculation of angular velocity and linear velocity is not complicated; it just requires a lot of time to optimize for precision. The case also applies to the headset itself; the fine-tuned gyroscope inside provides all information we need to detect head movement. In our project, if you twist the neck in a certain direction, you will make a dash movement to that direction to dodge the projectiles shot at you by giving your body a force toward that direction. To increase the difficulty of the game, we add a cooldown for this ability to avoid it being over-powerful. The cooldown is shown on the player's left hand.

I have to admit that it may be somewhat hard for people not playing games to get used to the control system since it consists of thousands of codes. But I can't even imagine a more intuitive way than something like moving your head to dodge and flapping wings to fly. It is pretty common to see a lot of game live streamer moving their body subconsciously when they get too involved in the game.

3.4 Boss Fight

3.4.1 Naive Chasing AI

In the boss battle fight, the boss will become "mad" and chase the player. To make the boss always move towards the player, we first used the "MoveToward" function provided in Unity which can make an object move towards another. Then, to rotate the boss, making it always face the player while moving, we first calculated the position difference between the boss and the player. The result is a vector that represents the direction of the movement. We can then apply the built functions to convert the vector to Quaternion to make the boss always face the player.

3.4.2 Shooting Projectiles

To make a bullet shoot toward the player, we first created the bullet object, a capsule, in the same position as the boss. Then, we computed the vector difference between the boss and the player. Using that vector, we added the force to the bullet in that direction. After that, the bullet will move toward the player. However, when the player moves, the bullet will only hit the original position.

3.4.3 Charge, Enchant, and Ultimate

Figure 6: Enchanting

Figure 7: Consuming

To make the battle less tedious, like cutting hair all the time, we implemented a charging system. The player can get charges by slicing a projectile shot toward the player, up to 3 stacks. After three stacks, the player can consume charges to enchant the weapon, making it effective against Medusa's invincible snakes.

The enchanted saber became very long, and we can use it to attack Medusa far from a long distance; this is a homage to Elder Ring. In Elder Ring's fight against Yamata no Orochi, the player needs to retrieve Serpent Hunter Spear, which is a spear that could be enchanted and grant players attack range.

3.5 Singleton mode for game system

Our game consists of 25 code files, hundreds of configurable variables, and 1000+ lines of code (excluding outsourced packages and built-in unity code). There has to be some system design to organize the code. Singleton mode is a design pattern that is often used in game development to ensure that only one instance of a particular class or object exists in the game at any given time. This can be useful for objects that represent the global game state, such as the player character or event system because it allows them to be accessed and modified by different parts of the game's code without creating conflicts or inconsistencies. Additionally, singletons can improve the performance of a game by reducing the number of objects that need to be created and managed. In our project, there is a game state manager takes part in every feature, like hp interaction and instance initialization.

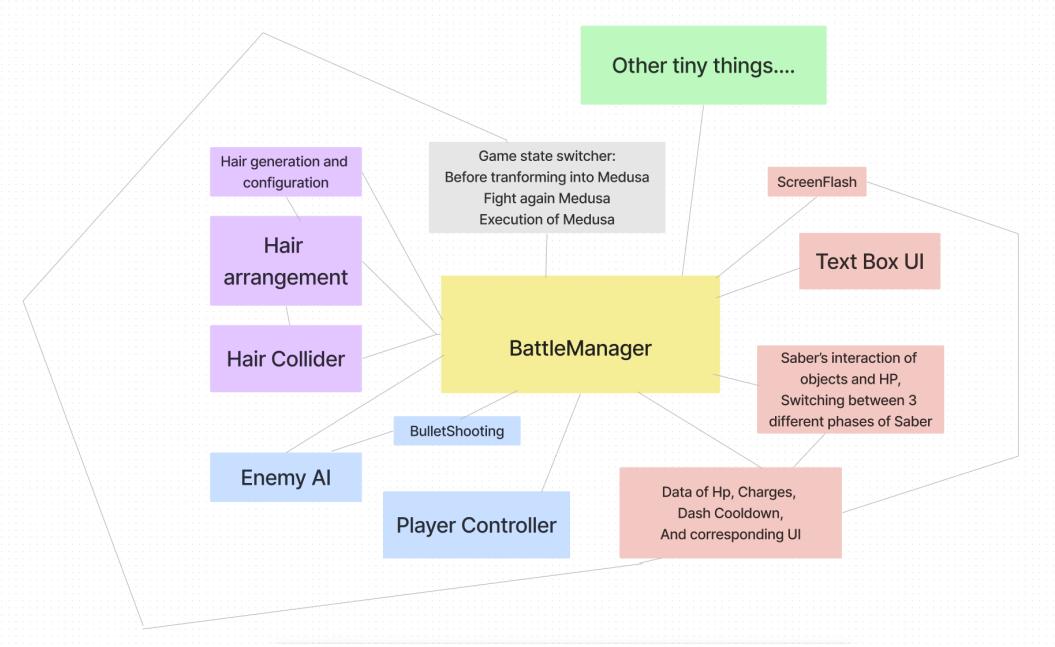

Figure 8: Singleton mode

3.6 Other Features

The traditional UI does not work that well in VR because it feels like something is following close to you. We have made our blade as the HP bar and other UI elements attached on the left hand, so the player is able to check their status through their left arm.

One other experience-purpose polishing we made is a script that detects whether the player is hit. When the player is hit, the screen will flash red for 0.5 seconds. This is done by creating a giant canvas in front of the VR camera. In normal situations, the canvas is not active. However, when the player is hit, the script will activate the canvas and create a coroutine to make it disappear after 0.5 seconds. The same effect is implemented for the state transition when players enter the boss fight.

We also have another feature paying homage to another epic game, which is the "Finish Him" from Mortal Combat. It is an execution mode after boss' HP drop below 0, allowing players to fly to him and slice the boss into pieces for fun. In Mortal Kombat, the Finish Him prompt appears when a player has reduced their opponent's health to a certain level, allowing the player to perform a Fatal Blow, a special move that can potentially defeat their opponent. This Finish Him combo varies depending on the champion being played and shows each champion's unique battle style. This move can only be performed once per match, so it should be used strategically to try and secure a victory.

Figure 9: Screenshot of Mortal Combat 11's Finish-Him, from official trailer

4 Results

The expected result of our application is the game can run without any issues, and players can have a great experience. Particularly, the game takes place in a room with a handful of hairs hanging around the room's ceiling. Then, the player will be regarded as a bee that can fly around by flapping its wrists as wings and hovering over by pushing down triggers. The player is able to move freely with joysticks and watch the whole scene by moving the head around. More importantly, the right controller can be used as a saber. When moving horizontally fast enough, it can cut the object encountered into pieces and cut our real-time rendered hair. There is a health bar associated with the hairs that can be seen by the player. Whenever they cut their hair, the head's health will drop. After dropping to some extent, the game state will switch to a boss battle where the head is now shooting a bullet at the player and cannot be attacked by normal saber. The player needs to either dodge or cut the bullet into pieces. If the player cuts the bullet, he or she gets a supercharge. When there are three stacks of supercharge, the player can shoot a laser saber that can hurt the boss, which is invincible to the normal saber after its health drop below 50%.

5 Conclusion

Overall, we almost finished the features in our proposal–the hair simulation, dashing system, and battle system. etc. All of those features work as expected. More importantly, the hair simulation, though not so sophisticated, is extremely hard to implement considering all problems we have in unity. It behaves well and can be rendered on Quest devices.

However, we hope the hair simulation system can be perfect. For now, the hairs don't have a collision first, so before being cut, hairs can go into some objects. Also, due to the drawback of line render, which updates only through points, hairs sometimes can be stretched to an extremely long distance because some of the particles will get frozen while the connected part has not been frozen yet.

We were supposed to do more gaming features like pose detection for dashing forward, not only dash to left or right sides. Better UI and instruction may also be helpful since our game is a soul-like game, user may spend some time get used to the control system if they are not game played

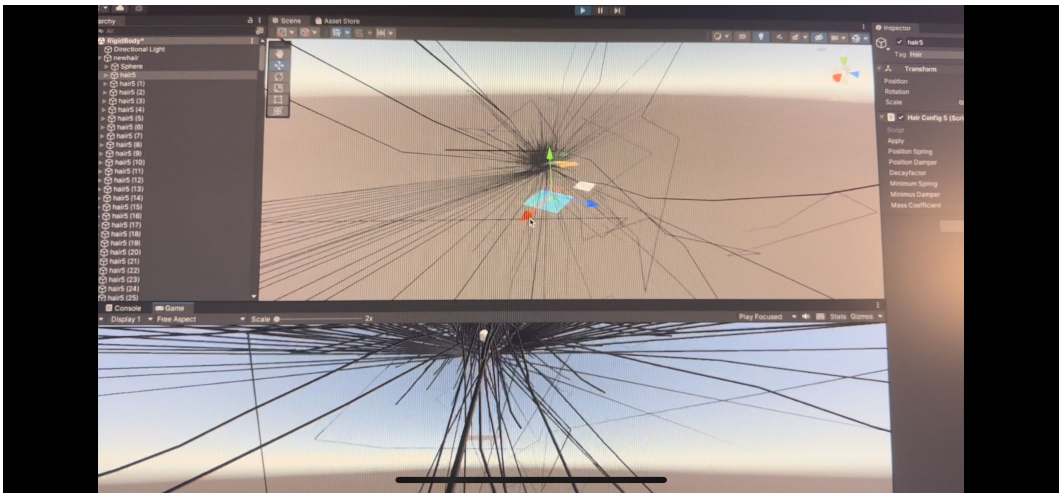

Figure 10: Exploding

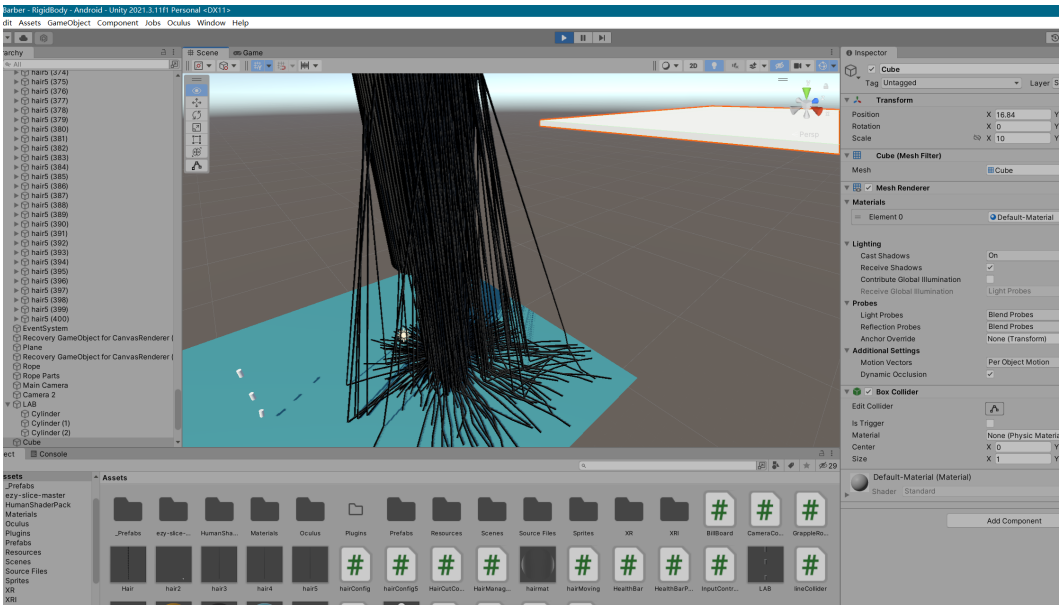

Figure 11: Line Render

before or have not played any arcade-style combat flight simulation games. Also, our game can be more polished by having an art team that takes care of the visual effect to make it more virtually engaging. Some bugs for hair simulation and cutting parts can be further optimized. In a nutshell, the project includes most of the features and works perfectly. Still, lots of improvement can be made, and I hope to explore more pose/motion detection for game control because of its immersive experience in VR.