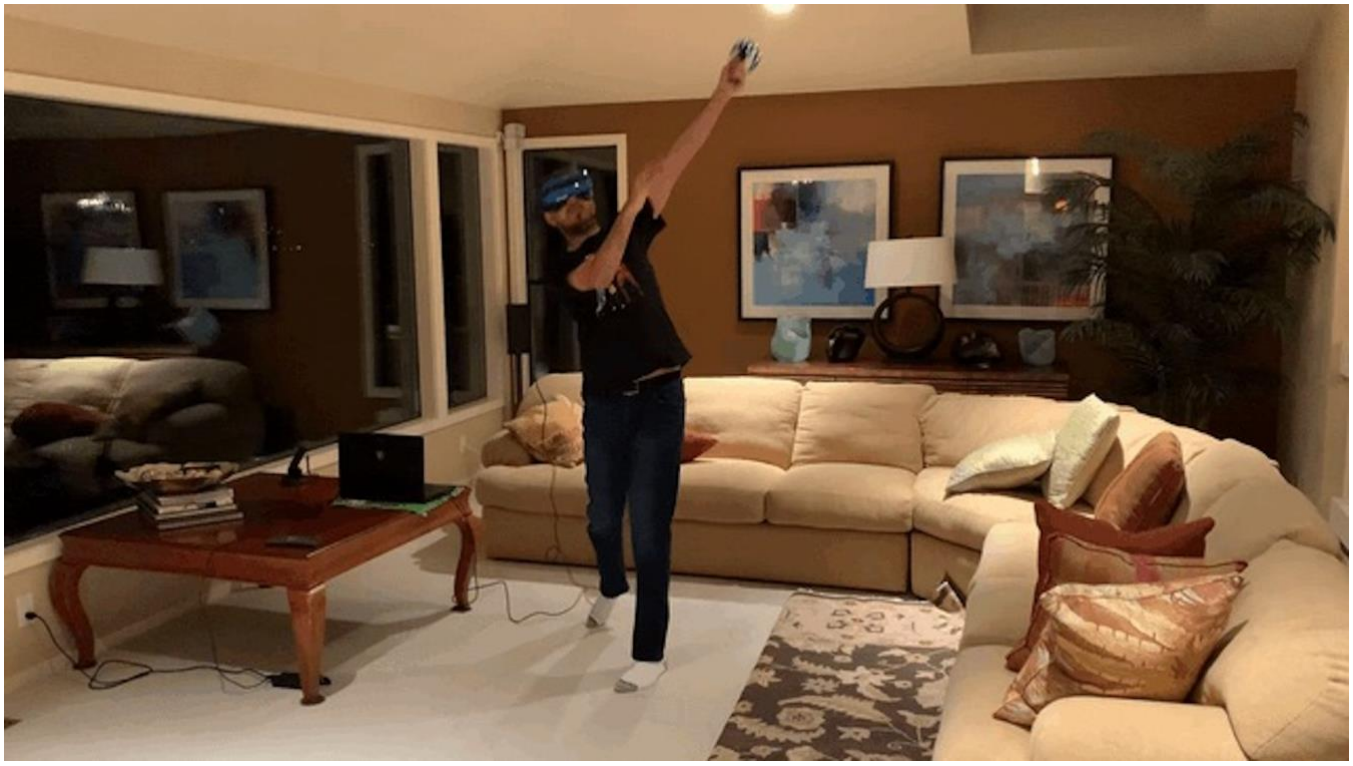
Virtual Batting Cage

Play Like a Pro

Dylan Hayre
CSE 490V
University of Washington
Seattle, WA
dylanh10@uw.edu



Abstract

This project simulates what it is like to hit a baseball moving at high speeds. The user can choose the speed and the location of each pitch. The user can also enable Professional mode, which simulates facing a real baseball pitcher. With my love for sports and intrigue for gaming, I was able to build virtual reality batting cage.

Introduction

The primary motivation for this project was to build something based on a topic I find interesting. I have always been a big fan of sports, including baseball. I've always been curious behind the physics behind baseball too. It takes great skill to be able to successfully hit a round ball with a round bat.

My approach to developing this project was to look up lots of documentation on how to use Unity. Whenever I needed to add a new feature, or play with the physics of the game, the Unity documentation provided me with clear solutions.

Overall, completing this project taught me many useful skills about game development and VR development. If I were to work on something similar, I would have much better knowledge on how to create a good application.

Related Work

There are lots of VR baseball games that currently exist. The one I used as a good example to follow is Everyday Baseball VR [1]. This game is a simple baseball simulator that allows the player to hit incoming pitches from an actual pitcher. The features from in that I wanted to incorporate included a trail that follows the ball after impact, and indicator of how far the ball was hit, and a notification when the player hit a homerun.

In-game Features

The Virtual Batting Cage includes a few interactable features. These include:

- Adjustable pitch speed
- Adjustable in-game height

- Adjustable height of the incoming pitch
- Wooden or metal bat
- Professional mode to randomize pitch speed and location

Method

There are four main steps that are involved to properly hit a baseball in VR. The first is to give the baseball a trajectory to follow when pitched. Second, you need to know how fast the bat is moving prior to impact. Third, the game needs to know when the bat and ball collide with each other. Lastly, the ball must travel along a certain trajectory with a certain speed after impact.

1. Pitch Trajectory

The first step is to know where the pitch should travel to, and how fast it should travel. First, in Unity, the baseball must be assigned a Rigidbody. This allows the ball to be affected by physics, including gravity and velocity changes. The user inputs how fast, in mph, the ball will travel, and how high above the ground the ball should be when it reaches home plate, in inches. After converting these values to meters/second and meters, I used the kinematics equations to solve for the initial velocities of the ball along the x and y axis. Specifically, I used equation 3 to calculate the time it takes to reach home plate, since the distance traveled is always constant, and the user input how fast along the x axis they want the ball to travel. After calculating t, I can use equation 3 again to calculate the initial velocity of the ball in the y direction, using a = -9.8 m/s. The change in x is just the difference between the height the ball starts at, which is always 1 meter, and the height the user entered as the height of the pitch. Using these values, I set the initial velocities along the x and y axis of the ball, and let the physics engine handle the actual travel of the ball.

1. $v = v_0 + at$
2. $\Delta x = \left(\frac{v+v_0}{2}\right)t$
3. $\Delta x = v_0 t + \frac{1}{2}at^2$
4. $v^2 = v_0^2 + 2a\Delta x$

2. Swing Speed

To know how far a ball will be hit, we need to know the speed that the bat is moving i.e. the swing speed. Conceptually, swing speed is based on how fast the bat is moving as the hitter turns their hips, so it can be thought of as an angular velocity. Unfortunately, this is not tracked automatically by MRTK, so manual calculations are necessary. Luckily, the bat does keep track of its rotation at each frame. So, to get the angular velocity, I take the difference between the rotations of the bat at the current frame and the previous frame, and divide it by the time between each frame, which is provided by Unity. This gives the angular velocity of the bat. To get the swing speed, I multiply the angular velocity by the length of the bat. Unfortunately, the swing speed is not always very accurate, and sometimes is massively overestimated. To combat this, a put a limit on all swings to be less than 75mph, since that is the average swing speed of a professional player.

3. Collision Detection

To have objects collide in Unity, both objects must have a Collider attached to them, so both the ball and bat must have Colliders. Unfortunately, just adding colliders is not good enough to detect collisions with the baseball simulator. Because the ball is small and moving at a high speed, it often teleports through the bat without noticing a collision. Also, the way MRTK keeps track of the controller is by constantly teleporting it to a new location, rather than continuous smooth movements in the game. This further adds difficulty with trying to get the ball to hit the bat.

To work around this issue, I used raycasting. Raycasts are invisible lines of a fixed length that travel in a given direction. If a ray passes through a collider, it returns this information. This essentially allowed me to expand the size of the ball. I cast three rays from the center, top, and bottom of the ball, each being about a half meter in length. The direction of each ray is the same as the direction that the ball is currently traveling, which includes it moving down because of gravity.

4. Trajectory and Exit Velocity

When a raycast detects a collision, it returns the normal of the surface that it hit. To keep things simple, I use this normal as the direction the ball will travel after impact. If multiple raycasts detect collisions, then I average the normal given by each hit.

To calculate the exit velocity, I used an equation derived by Alan Nathan. This equation estimates the exit velocity based on the pitch speed and swing speed. To keep things simple, I set e to always be 0.1 for a wooden bat, and 0.25 for a metal bat, since a metal bat produces a stronger force on the ball.

$$v_{ball} = e * v_{pitch} + (1 + e)v_{bat}$$

[2] Above is the Alan Nathan equation. The value e represents the coefficient of efficiency i.e. how well the ball was struck. On average, a well struck ball yields an e of 0.1.

Implementation Details

The headset I used for this project is the Windows Mixed Reality Headset. The tools I used to build this project include Unity and Windows Mixed Reality Toolkit. Unity provided an excellent environment to develop the game. This project was also a great way to make myself get familiar with Unity. Windows MRTK is the tool that enabled me to incorporate Virtual Reality into Unity. It attaches a camera into the scene that will follow the user's headset.

Limitations

There are a few issues with the simulation that prevent it from feeling truly realistic. First, since the raycasts are relatively long, a half meter, the trajectory of the baseball after impact is not accurate. For example, when the raycast detects a hit, it travels along the normal at that moment. However, there is still a half meter in

between the ball and the bat, and the bat is still rotating. So, the ball does not travel in the direction if there was a real collision.

Second, the swing speed that is calculated is not very accurate. As previously mentioned, sometimes the speed is a significant overestimate, and sometimes it is an underestimate. I noticed that swinging as hard as you can results in a slower swing speed than a casual swing. This is one issue that I'm not sure how to fix easily.

Lastly, the impact location on the bat currently does not affect the exit velocity of the ball. For example, a ball that hits the further end of the bat will travel further than a ball that hits closer to the hitters hands. To correct this, the bat would need to contain multiple colliders that each dictate what e value to use in the Alan Nathan equation.

Conclusion

Through completion of this project, I was able to teach myself the basics of working with Unity, and working on a virtual reality application. I learned a lot of skills by working on this project. While it may be simple, developing a game or application is a great entry into the world of VR.

Acknowledgements

- Thanks Kirit for all of the Unity help!
- Distinctive Developments Ltd for the baseball stadiums
- CGunwale for the baseball bat textures

References

[1] vrgamecritic. "Everyday Baseball VR - Trailer [VR, HTC Vive, Oculus Rift, WMR]." *YouTube*, 13 Sept. 2018, www.youtube.com/watch?v=-CqrGvFxRiw.

[2] Cole, Bryan. "What Does a 100mph Swing Speed Mean?" *Beyond the Box Score*, Beyond the Box Score, 11 Feb. 2015, www.beyondtheboxscore.com/2015/2/11/8010803/zepp-swing-sensor-perfect-game-100-mph-swing.