Animated Puppet with Wiimotes

Ryan Loebs

Abstract

Throughout the course of this quarter I have been developing a toy in which you control a puppet with the use of two Nintendo Wii Remotes. Although the project was not carried out according to plan the results are still entertaining.

1. Introduction

One of the key selling points of the Nintendo Wii was the use of motion controllers dubbed the Wii Remote, or more colloquially the "Wiimote." These remotes are a dream of a peripheral and contain numerous sensors for a game or toy to take advantage of. It is equipped with a three-axis accelerometer, a 100Hz infrared camera capable of detecting up to four individual light sources, an array of buttons across the device and the ability to expand this list with a port on the bottom. The entire device is available to any device that supports the Bluetooth Human Interface Device profile from the intended Nintendo Wii, to a personal computer, to even Android phones [1]. Nintendo's choice to use Bluetooth has made the device an incredibly useful and versatile device for more uses than a simple game controller. Projects such as those by Johnny Lee explore new uses for the device such as creating a digital whiteboard [2]. Personally I have used it as a remote control for my computer media players.

Although the Wii hasn't had tremendous success with "hard-core gamers" it has delivered with this remote and numerous games have been created to utilize the features. Games have ranged from shooting games which utilize the infrared camera to simulate pointing at a spot on the screen to party games which have you shaking the remote to trigger the

accelerometers.

The project that this paper describes aims to discover the process in creating an application with this remote in the form of a digital marionette. The remotes will serve as a tangible version of the handles one would use to control a marionette with the goal being to represent these handles as closely as possible.

2. Technical Details

Originally I had set out to complete this project using OpenGL and an open source library called CWiid in the C language. Given the time constraints and the complexity of creating a rudimentary physics system in 3D I could not reach this goal. Instead I switched to using the XNA framework in the C# language with the WiimoteLib library to access the remote data and FarseerPhysics to manage the ragdoll physics of the puppet.

XNA is a framework by Microsoft that provides a very easy interface to a DirectX rendering context and a fully managed game loop. It was first released in 2006 and has evolved into a very stable and powerful platform that can be used to develop for three Microsoft platforms simultaneously with very little platform-independent code.

WiimoteLib is a C# library that evolved out of a tutorial that was written describing how to read raw Wii remote data in Windows. Over time it too has stabilized and become an extremely simple mechanism for reading in Wii remote data.

Example WiimoteLib Usage:

```
WiimoteCollection wiimotes = new WiimoteCollection();
wiimotes.FindAllWiimotes();

foreach(Wiimote w in wiimotes) {
    w.Connect();
    w.SetReportType(InputReport.ButtonsAccel, true);
    w.SetLEDs(1);
}
```

The above code illustrates the ease of use in connecting to available remotes known to the system. The only prerequisite to using this code is to have the system connect to the remotes via Bluetooth first. This can be accomplished using almost any known Bluetooth adapter with any available Bluetooth stack. The code above has zero error checking and would produce a fatal error if there weren't any remotes attached to the system prior to running. This can be avoided by wrapping the entire block in a try/catch so the program can either die gracefully or try again if the user wanted to.

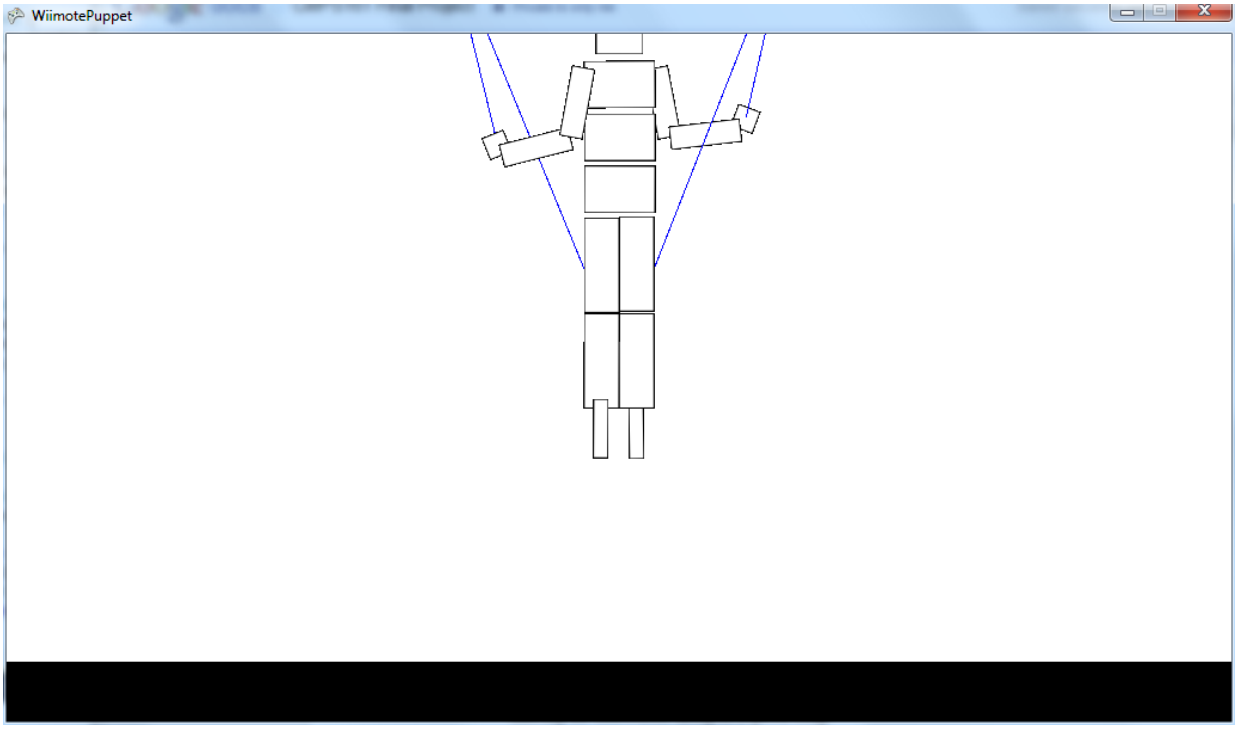
The Farseer physics engine is another open source library available for C# that was created to port the Box2D physics engine to the C# language. Over time Farseer started to develop its own systems straying from Box2D but overall attempts to mimic it as closely as possible. For the sake of this project the world gravity and joint systems were used heavily in order to create the marionette.

3. Method

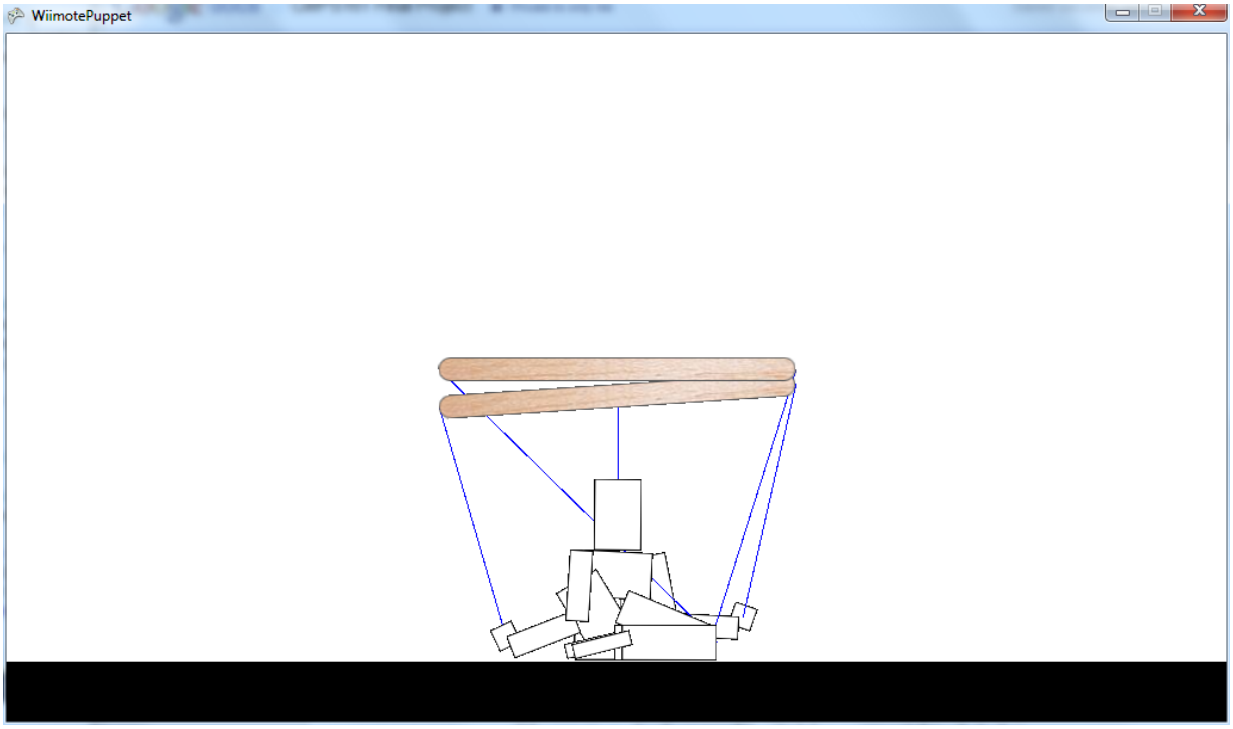
My first inclination in creating the puppet was to use inverse kinematics (IK) to be able to pull on the various parts of the body. The complexity of an IK system in contrast with the scope of this project made that option unreasonable. Instead I went with a rag doll approach in that each individual body piece is connected to its neighbor with revolute joints which gives the illusion of forward and inverse kinematics.

The seams of this approach can be seen from time to time when the parts twitch occasionally or when the body parts start to drift apart due to their weight straining the joints. Overall this solution gives the visual representation desired and was significantly easier to accomplish than true IK.

The Farseer physics engine provides a very well detailed object for each body in the world including its position and rotation which was used in rendering the textures. By itself the engine does not render anything but just sits in the background crunching numbers for the programmer to do with as they see fit.



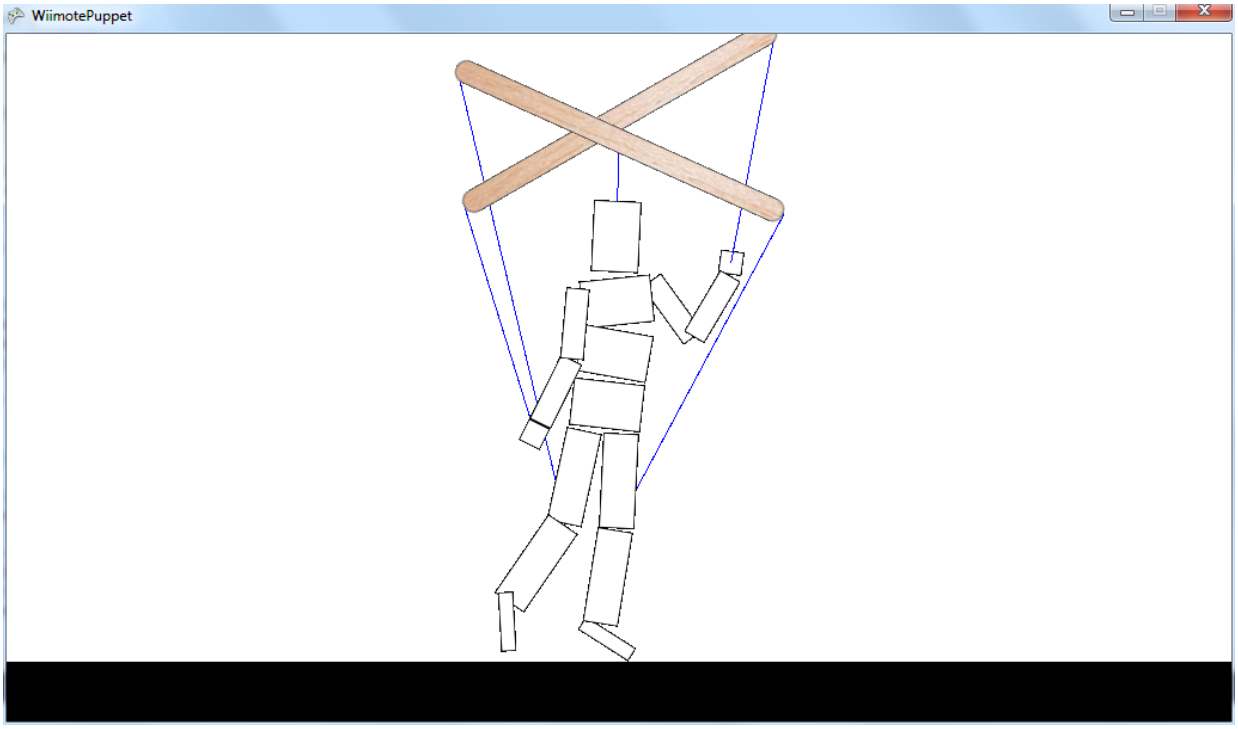
The height of the control handles is determined by the rotation of the remote along the X-axis with the midpoint being when the remote's buttons are facing the user. If the user places the remote face down the handle will drop down as low as it can go and if the user places it face up the handle flies up above the screen. This particular solution was decided upon to have direct control over the height without having to worry about calibration errors or lost data packets in the event that the motion of raising or lowering the remote was used. Very few commercial Wii games attempt to detect the motion of the remote in space and tend to have issues when they try. Wii Sports contains a mini-game for Golf and one common problem players have is the club not understanding where exactly the player's hand is in space.



Besides these particular design choices the handles react to the position of the remote as closely as possible which has a very satisfying feel when dancing the marionette along to a good song.

3. Results

Although the project needed to be scaled back to 2D the goal of creating a marionette controlled by two Wii remotes was a success. The WiimoteLib provided extraordinarily easy access to the data from the remote accelerometers which was easily translated onto the handles which controlled the hands and knees.



One issue that came up with the accelerometer data is that the granularity of the data is very low. In 180 degrees of motion only about 24 unique positions are reported which caused the handles to twitch and jerk around very rapidly as the data changed. This was dealt with by buffering the positions coming in from the accelerometer and tweening the handles to the current orientation which also provided a nice smoothing effect that was not initially intended. The downfall to this solution is that quick movements of the remotes do not translate quickly enough to the puppet and incur a small delay. Quick movements can still be accomplished but rotating the remote 180 degrees from left to right illustrates the problem clearly.

The rag doll physics was all handled transparently by the Farseer engine and was not an issue whatsoever as it was an extremely easy library to learn despite its complete lack of proper documentation. The only negative was the time it took to rig up the puppet with all of the rectangles and joints joining everything as I wanted, but this would not have been avoided no matter what I used.

4. Conclusion

Development using the Wii remotes has become an extremely simple affair due to all of the work that others have done to get the data from the remotes to the code. Although I did not program the physics that powers the rag doll it was still necessary to have an understanding of the physics concepts to take control of the engine.

Had there been more time I feel that I would have been able to accomplish the same thing in 3D with OpenGL but there simply was not enough time. The Wii remote libraries available in C are just as simple to use as WiimoteLib but were created in the C style and allow the programmer much more control than the managed C# equivalent.

5. References

[1] Wii remote connectivity for Android devices. http://goo.gl/80Hm2

[2] Wii projects. Johnny Chung Lee. http://johnnylee.net/projects/wii/

[3] XNA Game Studio 4.0. Microsoft game platform for C#.
Available at http://create.msdn.com/

[4] FarseerPhsyics Engine. Box2D port to C# and XNA.
Available at http://farseerphysics.codeplex.com/

[5] WiimoteLib. Managed Wiimote library in C#.
Available at: http://wiimotelib.codeplex.com/