KinectFlix: A Hand Gesture Recognition Application for Kinect to Watch NetFlix

Kuo-pao Yang*, Sebastian van Delden, Elyse Bond

Computer Science and Industrial Technology, Southeastern Louisiana University, Hammond, LA 70402

*Corresponding Author: kyang@selu.edu

Copyright © 2014 Horizon Research Publishing. All rights reserved.

Abstract KinectFlix is a hand gesture recognition application which uses a Xbox 360 Kinect to interact with NetFlix while watching online movies and TV shows. This project is designed and developed with the intention of providing an easily adaptable interface for the Xbox 360 Kinect to be used with Windows Presentation Foundation (WPF) applications to provide a controller-less interactive user interface.

Keywords Xbox 360 Kinect, NetFlix, Controller-less Interaction

1. Introduction

For decades, researchers have been exploring new approaches to physical and digital space integration [1]. Recent years have been particularly exciting due to the relatively inexpensive, gesture-based control interfaces now commonplace with gaming consoles. Although the Microsoft Wii pioneered sensing and gesture recognition technology, the Microsoft Kinect is arguably becoming the platform of choice by gamers and researchers alike [2]. Microsoft released the Xbox 360 Kinect to North America and the UK in November 2010. The Kinect is a controller-free entertainment console that has been a monumental success for Microsoft. With only hand gestures and voice commands, this hardware allows users to interact with software without needing assistance from any other input device. Input devices are a significant part of every computer system but with this sort of implementation you essentially are the input device [3]. The idea for KinectFlix came after the Kinect was released.

Since Microsoft, at the time, had not released Kinect's API and documentation to the public, we first started developing open source drivers which allowed Kinect-to-PC communication. During this development, it became clear that an interface for the Kinect would be a great way to allow different applications to be developed for use with the Kinect.

KinectFlix was created using C# with Model View View

Model (MVVM) Architecture, Prime Sense ManagedNite Library, Open NI, NetFlix Odata, and the Microsoft Xbox 360 Kinect. C# was used as the application's base programming language for the back end structure and user interface. The MVVM architectural style allows for the user interface and back end functionality to work independently and not rely on each other. This architecture was an adequate choice because it allows the application to have an easily interchangeable user interface. The Prime Sense ManagedNite library is a wrapper for the .NET programming languages. This library is an open source library that allows us to programmatically access data from the Xbox 360 Kinect. Open NI is software that was developed for hand and motion tracking. This software is used with the ManagedNite library for processing data being interpreted by the Kinect. NetFlix is a Windows Presentation Foundation (WPF) based application, which is a next generation presentation system for building Windows client applications with visually stunning user experiences. NetFlix Odata, which streams and updates information from the NetFlix database, was used in creating the user interface for the application.

KinectFlix functions by processing information being read by two cameras on the Xbox 360 Kinect. That data is then manipulated to correspond to the users hand to mouse coordinates of the user's computer. Depth perception of the users hand is used to create functionality for mouse clicks and hand gestures. This functionality is then used to navigate and search a user interface that contains a repository of movies and TV shows that are available on NetFlix.

2. The Problem

Although there are many applications that have been created that use the Kinect to provide some functionalities, there has not been an application that allows one to plug in a user interface and already have that basic functionality of the Kinect for use in NetFlix at their disposal. As tangible interfaces are increasingly becoming commercially available products in the current generation [4], this will

eliminate a developer from having to repetitively “Reinvent” functionality for the Kinect.

On the other end, the user interface provided allows one user to search through the Netflix database for movies without having to use the remote. This is convenient in the sense that it saves time in the situation where the remote cannot be found or if the user of the product does not feel like getting up to get the remote control.

It is the intention of this application to provide software developers with a convenient means of starting an application which implements technologies of the Microsoft Kinect as well as a convenient means for users to browse the Netflix repository in a controller-less environment. While this application is not yet complete, the following sections of this report will be a guide through the implementation and development process of the application that makes these functionalities possible.

3. Software Implementation and Development

3.1. Combinations of ManagedNite and OpenNI with Xbox Kinect

A combination of the ManagedNite Library with the OpenNI device drivers are used with the Microsoft Xbox 360 Kinect to enable a PC to use the Kinect and provide a .NET library for accessing and controlling motor functions and data being processed by the Kinect.

OpenNI is an open source framework that provides necessary API's for writing applications that will utilize natural interactions. This framework is being used for tracking the hands of the user with the Kinect. Once the user's hand is registered with this API, a point is created that then corresponds to the movements and gestures of the user's hand until that point is lost. The state of the API tracking a user's hand is called a session.

The ManagedNite library is a C# wrapper to the OpenNI API's that allows the application to integrate the Kinect with a series of object classes to read and manipulate data. This library allows the developer to create session states and events that will correspond with hand movements. The combination of session states and events are what makes hand gestures and functionality through an application possible in a controller-less environment.

The Kinect for Xbox360 is equipped with two cameras, 4 microphones, and a motor that is used for pivoting the cameras vertically. Of the two cameras, one of them is used with an infrared emitter which creates a depth map of what that camera is seeing. This depth map reports distance to the nearest surface at each pixel [5]. The other camera captures the visual spectrum at a 640 X 480 resolution. The combination of these two cameras are used to create a three dimensional model of everything that it sees. In essence, this sensor model observes the user's behavior and

generates data [6]. This enables the identification of hand gestures regardless of the orientation of the user to the camera [7].

3.2 KinectFlix Back End Structure

KinectFlix has an elaborate back end that is used to handle the interactions between the Kinect and the user as well as the interaction between the Kinect and the user interface. This section will go into detail about the implementation of the earlier mentioned libraries and the streaming of content using the Netflix Odata.

When the application starts, the Kinect is initialized and sits in an idle state until a gesture is recognized that will tell the Kinect to start tracking and filtering data. Upon initialization, the cameras and the motors are started, and the monitor area is determined and scaled accordingly. Once the initializing gesture is detected, various event handlers and listeners are created such as Swipe and Click shown in Figure 1. These listeners are guidelines for what the Kinect is looking for.

```
controller = new XnmSessionManager(
    ni, "Wave", "Click");
controller.SessionStarted += new EventHandler(
    <EventArgs>(controller_SessionStarted);
controller.SessionEnded += new EventHandler(
    controller_SessionEnded);
filter.PointCreate += new EventHandler(
    <EventArgs>(filter_PointCreate);
filter.PointUpdate += new EventHandler(
    <EventArgs>(filter_PointUpdate);
clickDetect.Push += new EventHandler(
    <EventArgs>(clickDetect_Push);
swipeDetect.SwipeLeft += new EventHandler(
    <EventArgs>(swipeDetect_swipeLeft);
swipeDetect.SwipeRight += new EventHandler(
    <EventArgs>(swipeDetect_swipeRight);
controller.AddListener(clickDetect);
controller.AddListener(swipeDetect);
controller.AddListener(filter);
```

Figure 1. Initialized event handlers and listeners created when initializing gesture is detected

In Figure 2, when the Kinect sees that guidelines are met, an event handler for a gesture is fired. When a gesture is triggered, a predefined programmed instruction set will be executed [8]. This method of interaction can easily be altered so that the gesture will interact differently based on the needs of the application. No single method for an

automatic hand gesture is suitable for every application. Each gesture recognition algorithm depends on user cultural background, application domain, and environment [9].

When a specific hand gesture is not being executed, the application is tracking the hand coordinates and using them to control the mouse. This is done by constantly relaying the coordinates of the hand to the coordinates of the mouse shown in Figure 3. An algorithm is used to transition from hand coordinates to mouse coordinates.

```
if (Session.clickDetected)
{
    this.Debug = "Click Detected";
    this.posX = this.superSquare.Width / 2 +
        Session.Xpos * this.Speed;
    this.posY = this.superSquare.Height / 2 -
        this.PropChange("xSensor");
    this.PropChange("ySensor");
    SetCursorPos((int)this.posX, (int)this.posY);
    button1_Click(sender, e);
    Session.clickDetected = false;
}
```

Figure 2. Instruction set that is executed when the gesture for the click event is performed by the user

```
this.posX = this.superSquare.Width / 2 +
    Session.Xpos * this.Speed;
this.posY = this.superSquare.Height / 2 -
    Session.Ypos * this.Speed;
```

Figure 3. This code segment is mapping out the transition of coordinates from the user's hand to the mouse coordinates

The above mentioned interactions work together to perform complete human computer interaction without the use of a physical input device. The innovative aspects introduced by hands free gaming systems are an indication that technology is progressively reaching more natural ways of human interaction with machines [10]. These indications are also true with the future of software applications.

4. Graphical User Interface

The user interface was developed with the intentions of allowing the user to browse the Netflix repository by genre of movie. The following section provides a brief walkthrough of the user interface and what it provides.

This application is started at the Welcome Screen shown in Figure 4. The user is prompted to begin waving. As long as the Kinect has detected the user's hand, this application automatically navigates to an alternate screen to notify the

user that his/her hand is now being tracked by the Kinect. The user is then prompted to click the "Browse" button, which then brings the user to the Netflix Repository.

Upon reaching the actual Netflix repository within the application, the user is presented with a multitude of options. By default, a category is already selected and displays movies for that particular genre as shown in Figure 5.

In order to switch the category, the user should move his hand from right to left or from left to right. This registers within the application as a "Swipe". When this swipe is detected the storyboard will activate and automatically change the genre.

Figure 4. Welcome screen prompts user to initiate the application by waving

Figure 5. Browse through genres

Figure 6. Individual movie information screen

Once the user has found a genre they would like to explore. They may either scroll up and down through the list of movies or select the movie and view more information about it. To scroll up and down, click the up and down arrows located on the right side of the list of available movie titles. To perform a click, push the hand

that is guiding the mouse forward. This will then perform a mouse click within the application and scroll up or down depending on which button you are pressing. Similarly, to view more details about a title, click on the title you wish to view. A window will then pop up that will display details about the title that the user has chosen as shown in Figure 6.

While the interface that has been provided is not too intricate with details, it does a good job of displaying the functionality of the Kinect. This interface also provides a simple and well developed starter application using Netflix Odata. The basic functionalities for hand gestures include:

- Wave: Activate the program or bring it out of an idle state
- Push: Click
- Swipe-Up: Scroll up through movies in current genre
- Swipe-Down: Scroll down through movies in current genre
- Swipe-Left: Scroll genres to the left
- Swipe-Right: Scroll genres to the right

5. Conclusion

This project successfully integrates the Microsoft Kinect into a modern business application, implements a Kinect controlled mouse pointer, and uses hand-gestures as "short-cuts." Multiple hand recognition and mouse controls will be the future work.

REFERENCES

- [1] A. Mazalek, S. Chandrasekharan, M. Nitsche, T. Welsh, P. Clifton, A. Quitmeyer, F. Peer, F. Kirschner, and D. Athreya, I'm in the Game: Embodied Puppet Interface Improves Avatar Control, Proceedings of 5th International Conference on Tangible, Embedded, and Embodied Interaction (TEI '11), Funchal, Portugal, 129-136, January, 2011.
- [2] A. Williams, L. Hughes, and B. Simon, Proximity: Exploring Embodied Gameplay, Proceedings of the 12th ACM International Conference on Ubiquitous Computing - Adjunct (Ubicomp '10), Copenhagen, Denmark, 387-388, September, 2010.
- [3] C. Schonauer, T. Pintaric, and H. Kaufmann, Full Body Interaction for Serious Games in Motor Rehabilitation, Proceedings of the 2nd Augmented Human International Conference (AH'11), Tokyo, Japan, Article No. 4, March, 2011.
- [4] A. Williams, and E. Kabisch, Bizarro Game Controllers Studio Proposal, Proceedings of 5th International Conference on Tangible, Embedded, and Embodied Interaction (TEI '11), Funchal, Portugal, 337-340, January, 2011.
- [5] A. Wilson, Using a Depth Camera as a Touch Sensor, Proceedings of ACM International Conference on Interactive Tabletops and Surfaces (ITS '10), 69-72, Saarbrücken, Germany, November, 2010.
- [6] J. Lee, J. Kiser, A. Bobick, and A. Thomaz, Vision-Based Contingency Detection, Proceedings of the 6th International Conference on Human-Robot Interaction (HRI '11), Lausanne, Switzerland, 297-304, March, 2011.
- [7] A. Ramey, V. González-Pacheco, and M. Salichs, Integration of a Low-Cost RGB-D Sensor in a Social Robot for Gesture Recognition, Proceedings of the 6th International Conference on Human-Robot Interaction (HRI '11), Lausanne, Switzerland, 229-230, March, 2011.
- [8] A. Bleiweiss, D. Eshar, G. Kutliroff, A. Lerner, Y. Oshrat, Y. Yanai, and O. Interactive, Enhanced Interactive Gaming by Blending Full-Body Tracking and Gesture Animation, Proceedings of ACM SIGGRAPH ASIA 2010 (SA'10), Seoul, South Korea, Article No. 34, December, 2010.
- [9] J. Wachs, M. Kolsch, H. Stern, and Y. Edan, Vision-Based Hand-Gesture Applications, Communications of the ACM, Vol. 54, No. 2, 60-71, February, 2011.
- [10] M. Roccetti, G. Marfia, and M. Zanichelli, The Art and Craft of Making the Tortellino: Playing with a Digital Gesture Recognizer for Preparing Pasta Culinary Recipes, Computers in Entertainment (CIE) - Theoretical and Practical Computer Applications in Entertainment, New York, Vol. 8, Issue 4, Article No. 28, December, 2010.