CS 4999 Independent Research

Midterm Report

Cornell University Program for Computer Graphics

Lim Mingjie, Kenneth (kl545), Leart Albert Ulaj (lau8)

April 11, 2014

Lorem Ipsum

Contents

1. Foreword

This paper accompanies software code and documentation hosted on a private version control server, to which access is available on request.

2. Overview

Microsoft's new Kinect for Windows, released under the Kinect for Windows Development (K4WDev) Program[1] incorporates a time-of-flight (TOF) laser for depth-sensing at an effective range of 1–15 feet (0.3–4.5 meters). The higher level of precision and increased range of the TOF laser (compared to the older Kinect, which uses a pseudo-random infrared dot pattern) makes the new Kinect a suitable candidate for gesture-recognition applications in: (1) environments where it is not always possible for the user to be situated in front of the sensor such that depth detection is optimized; (2) environments where the user may be one of multiple bodies in the sensor's field-of-view (FOV), of which it is difficult to disambiguate the actions and gestures of the other non-participatory bodies and the number of bodies far exceeds the number that can be tracked by the Kinect; and (3) environments which are not well-lit.

Although the Kinect's target field of application is video gaming, the robustness of its 3-D capture data has encouraged both research at the academic level, and independent projects of varying scale at the amateur/enthusiast level. Most approaches to gesture-based interfaces, however, rely on joint-tracking rather than recognition. A set of parameters, e.g. velocities and accelerations, or user-defined point clouds in the sensor's FOV are specified for the purpose of localizing and tracking movement. For example, a number of applications have been developed using USC's Flexible Action and Articulated Skeleton Toolkit (FAAST) and SigmaRD's SigmaNIL Hand Recognition Framework, among others. Although many approaches have been shown to work well in scripted scenarios, one would expect difficulties deploying these approaches in real-world scenarios due to differences in users' body parameters and movement styles.

In this paper, we describe the use of a supervised constraint-based algorithm for robust gesture detection using the new Kinect. The algorithm is simple, extensible, and computationally efficient. We also describe the use of temporal averaging methods in the design of a user interface that consumes gestural data from the Kinect. Lastly, we contextualize this software ecosystem in the use of classroom (or instructional facility) design as an example of a real-world deployment scenario.

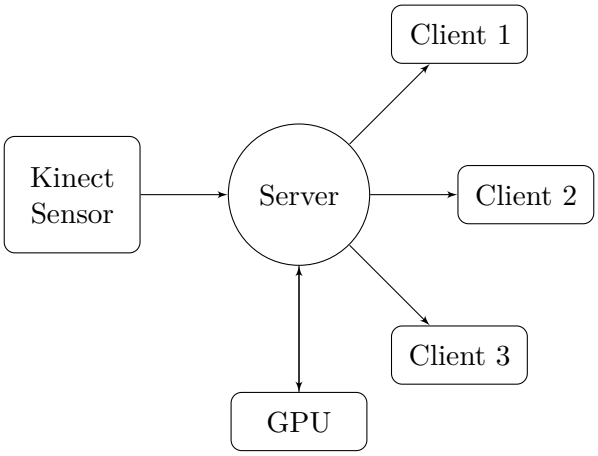

Figure 1: Software infrastructure event flowchart depicting the Kinect sensor interfacing with three clients via the server.

3. Project Specifications

The software is conceptualized as a hybrid client capable of supporting multiple connections. The back-end is written in a low-level language that interfaces directly with the Kinect (via the Microsoft Kinect API) and uses the workstation's graphical processing unit (GPU) to convert raw sensor data into well-formed intermediate output. The intention is to offload processor-intensive computations to the workstation and stream the output to the front-end, which is a modular, lightweight application with little to no dependencies. The front-end consumes the output and renders graphical content to the user's display. Figure 1 depicts the flow of data from the Kinect sensor to the server, where it is interpreted before being piped to connected user clients.

Establishing the software as a dichotomy of a front-end and back-end allows for the front-end to be adapted for different applications, treating the output from the server as a black-boxed "bag-of-features". In addition, since the Microsoft Kinect API is still under development, new changes can be slipstreamed into the back-end without further modification to the rest of the software infrastructure.

3.1. Hardware

We utilize the new Kinect sensor, provided to the Cornell Program of Computer Graphics (PCG) by Microsoft under the K4WDev program. Interaction with the application was done on an 82-inch Perceptive Pixels Inc. (PPI) touchscreen display at HD resolution (1920×1080). Testing was performed on a secondary, 52-inch PPI touchscreen display at 4K resolution (3840×2160).

[1] http://www.microsoft.com/en-us/kinectforwindowsdev/newdevkit.aspx

3.2. Back-End

3.3. Front-End

Content schema and human interaction feedback were rendered in real-time in a modern web browser. The application leverages the HTML5 WebSockets API to connect to the back-end, then renders incoming data on a GPU-accelerated HTML canvas layer overlaid on the application user interface. Serving the application via a web interface eliminates the need for users to install any software on their devices (since all assets can be hosted on a remote server) and allows for multiple (remote) connection instances without additional infrastructure overhead.

4. Road-map

I. Core Functionality

The Kinect SDK tracks 21 3-D joint coordinates of the human skeleton model in real-time (30 frames/sec), corresponding to the spatial location of a user in the Kinect sensor's FOV. This polling rate generates $21 \times 30 \times 3 = 1890$ real-valued skeleton model coordinates per second. In addition, the skeleton model is robust to variation in the shape and size of the human body, color and texture of clothing, and objects or surfaces in the background, thus we omit consideration for these variables (otherwise considered anomalies) in the design of our action model representation.

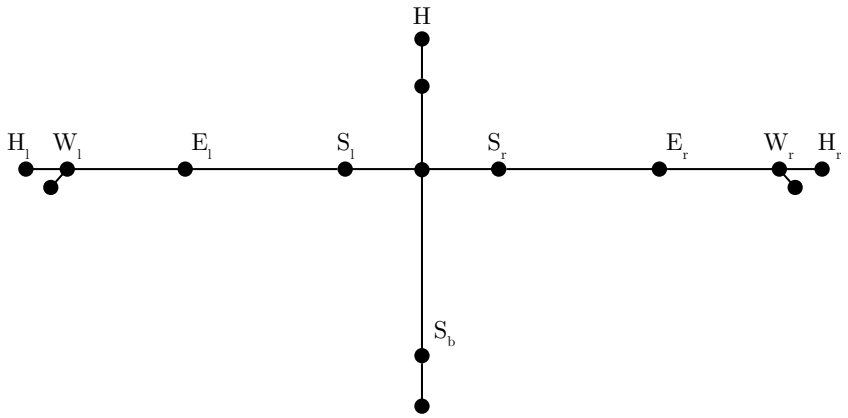

Figure 2: Partial joint nomenclature for the Kinect Skeleton Model, showing the top half of the body. Joints of interest are labeled, where H corresponds to the head, $S_{\{l,r\}}$ corresponds to the left or right shoulder, $E_{\{l,r\}}$ corresponds to the left or right elbow, $W_{\{l,r\}}$ corresponds to the left or right wrist, $H_{\{l,r\}}$ corresponds to the left or right hand, and S_b corresponds to the base of the spine.

Since our implementation is gesture-based, we begin by simplifying the skeleton model and focusing only on coordinates above the base of the spine. Specifically, the most pertinent joints are the left and right hand, left and right wrist, left and right elbow, and left and right shoulder. Figure 2 assigns labels to these joints for ease of reference in subsequent discussions.

II. Establishing Coherence

II.1. Fleck WebSockets Server

II.2. HTML5 WebSockets API

III. Viewport Nomenclature

IV. Gestural Interaction

As a simple nomenclature which we will build upon in this section, we classify gestures as either *arm-level* gestures or *hand-level* gestures. Eponymously, arm-level gestures involve the movement of the entire arm, such as pushing, pulling, or swiping actions. Hand-level gestures involve the movement of only the hand, such as expanding all fingers to show an open palm, clenching all fingers to form a fist, or extending a single finger to point in a direction.

V. Trigger Mechanism

In practice, gesture recognition must be applied to a finite data-set (e.g. a video segment), in which the algorithm returns all statistically significant occurrences of a gesture within the data-set. In such cases, it is trivial to repeatedly expand a window of analysis to include frames before and after a suspected gesture component. However, it is not possible to deterministically identify a gesture when incoming data arrives as a stream, because the subsequent frames of data are unknown to the algorithm. We overcome this difficulty by constraining the initial set of permissible actions to the raising of the left or right hand above shoulder height. Specifically we define the constraints:

$$x_{S_r} \leq x_{W_r} \leq x_{E_r} \tag{4.1}$$

$$y_{S_r} \geq y_{W_r} \geq y_{E_r} \tag{4.2}$$

for detecting a raised right hand, and similarly:

$$x_{S_l} \leq x_{W_l} \leq x_{E_l} \tag{4.3}$$

$$y_{S_l} \geq y_{W_l} \geq y_{E_l} \tag{4.4}$$

for detecting a raised left hand.

V.1. Cursor Manipulation

V.2. Hand States

V.3. Pulling and Pushing

V.4. Swiping

VI. Content Development

VII. Testing and Robustness Evaluation

This section provides a speculative overview of work that may be conducted this area, and will be updated as more information becomes available.

VIII. User Recognition

This section provides a speculative overview of work that may be conducted this area, and will be updated as more information becomes available.

5. Deliverables

I. 27 January

II. 24 February

Phase I is expected to be completed by this time.

III. 31 March

Phase II is expected to be completed by this time.

IV. 28 April

Phases IV and VI is expected to be completed by this time.

V. End of Semester

Phase VII is expected to be completed by this time.

6. Extensions

7. Acknowledgments

The authors are indebted to Donald Greenberg, Joe Kider, John Decorato, and Jeremy Newlin for advice and assistance rendered throughout the course of the project. This work was supported by the Cornell Program of Computer Graphics.