Generalized Server for using Motion Capture Data in Virtual Reality Applications

Amey Vikas Edwankar
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14586
axe5592@cs.rit.edu

Abstract—This paper focuses on the development of a generalized server for using motion capture data in virtual reality applications. With the rise of affordable VR and AR devices, they have become more approachable to the masses other than the niche market of high-end gaming consumers. Along with the use of motion capture devices like the Kinect, this can lead to the creation of immersive experiences which earlier would not have been possible. One such experience is Farewell to Dawn which is a mixed reality dance performance which leverages the Kinect for performance capture and display their performances on a virtual stage on VR and AR devices. The server developed will be used in Farewell to Dawn to facilitate the motion capture at one site and transport the capture performance to the server site where it will be rendered by attaching the movements to a 3D character.

I. INTRODUCTION

A. Problem and Motivation

The goal of this project is to design and implement a generalized, realtime server for sending and receiving full body motion capture data regardless of the motion capture system that is utilized.

B. Related Work

Although the Kinect has been released since a long time, it has not been used in a widespread manner. The uses of the Kinect are in very specialized circumstances which only relate to performance capture and rendering the captures body data. There haven’t been any applications where the captured body data has been streamed to another device since most applications perform the rendering and the performance capture at the same site. Therefore although there are various formats used to store 3D motion capture data there aren’t any standards which we can conform to.

There are multiple resources online especially in blogs which contain tutorials for Unity and the Kinect plugin for Unity. I was able to leverage these to understand the basics and build on top of them. In addition to the performance capture tutorials I was also able to find comprehensive tutorials for Unity’s networking suite. On studying both the higher level and the lower lever networking API which is provided by Unity I decided to use the lower layer API since it uses the UDP protocol hence allowing us to stream the motion capture data.

II. BACKGROUND

A. Microsoft Kinect

The Microsoft Kinect is a motion sensing input device developed by Microsoft which allows us to track the position and movement of the human body. This is done through reducing the person’s body to a simple skeleton where points on the body are depicted in the form of co-ordinates in a 3D environment. The Kinect can distinguish between a human and other objects as can be seen in figure 1. The kinect tracks human bodies by tracking important points on the body such as the head, arms, hands legs, knees, hips etc. These points on the body are tracked in 3D space and are connected together to form a basic skeleton representing the human body. Figure 1 shows the kinect identifying a human body among other objects, coloring it and joining together the various important points on the body to create a skeleton.

B. Unity

Unity is a game engine which allows the development of video games and simulations. We will be leveraging unity to record the motion captured through the Kinect and also to
stream the performance capture data over the network. The Kinect SDK for Windows contains a Kinect plugin for Unity which will allow us to record the performance capture data. Figure 2 shows an example of the various feeds that we can access from the Kinect using the Unity plugin. The feeds shown on the image are a video feed which is taken from the Kinect’s RGB camera, a grayscale feed which is obtained from the Kinect’s infrared camera, a skeletal feed which contains the tracked body parts connected by a mesh and a raw depth field which contains a representation of how far away objects are in relation to the human body which is recognized.

Unity also has a robust networking suite and contains a higher and Lower level networking API. By leveraging the lower level networking API we will be able to send the performance capture data through the transport layer using UDP hence allowing us to stream the performance capture data from the location where the performance is captured to the location of the server which will also contain the Unity application which will visualize the performance capture data.

### III. DESIGN

The networking system will consist of two components one on the side of the performance capture site and the other on the site of the server.

#### A. Performance Capture

The performance will be captured through Unity using the Kinect plugin for Unity. The Kinect through the use of its sensors allows us to read 7 different feeds which we can use. For our purposes we will be reading the coordinates of all the skeletal points of the performer. The coordinates for every skeletal point for every frame will be written to an object which will contain all the coordinates of the various points of the performers skeletal frame.

#### B. Streaming

For streaming the performance capture data to the application the Unity transport layer API will be leveraged, this will allow us to stream data through UDP. The object containing the coordinates of all the skeletal points will be streamed for every frame that is recorded.

#### C. Character Rendering

On receiving the streamed object data at the server site the coordinates of the skeletal frame will be attached to the corresponding points on the 3D character for each frame that is received allowing us to apply the performers movements on to the 3D character.

### IV. IMPLEMENTATION

The system consists of two components the capture site where the motion capture takes place and the server where the motion capture data is received and rendered onto the 3D character.

#### A. Capture Site

The capture site first tries to connect to the server to do this we need to provide servers IP address. Once the capture site gets the IP address it begins the motion capture. For each frame that the capture site receives 3D coordinates the capture site does the following in it’s update function:

- Extract 3D coordinates for each body part received from the Kinect
- Add each body parts 3D coordinates to a dictionary
- Serialize dictionary
- Send serialized dictionary to server.

#### B. Server

The server site does the following in it’s update method to render the character for each packet it receives from the capture site.

- Deserialize dictionary
- Extract 3D coordinates for each body part from deserialized dictionary
- Apply 3D coordinates to the character.

### V. RESULTS

The server can successfully render a 3D character which mirrors the motion that a performer performs. The motion is captured successfully at the capture site and then streamed to the server where it is applied to the 3D character. The results can be seen in the figure 4 which shows a rendered character.

### VI. CONCLUSION

#### A. Current Status

Currently the system only takes one Kinect’s data which only follows one performer which means that only one character can be rendered on the screen at one time.
B. Lessons Learned

The project was a huge learning experience and allowed me to marry two of my interests distributed systems and computer graphics. The following are the lessons that I have learnt through the development of the project.

- Learned how to work with a Kinect and how it captures motion.
- Learned how to use the Kinect SDK and the Kinect plugin for unity.
- Learned how to use Unity’s networking suite in particular the Transport Layer API which uses the UDP protocol to stream data between communicating instances of Unity.
- Learned how to apply captured motion from a Kinect to a 3D character.

VII. Future Work

Currently the system only has the capability to capture and render one performer's motion to a 3D character. In the future the system should be extended so it can receive the motion capture data of two different performers and display them in the same scene with relative distances between them so that performers can perform in front of separate Kinects in separate sets and their performances can be displayed in the same scene.

VIII. Acknowledgement

I would like to thank my advisor, Dr. Joe Geigel for providing me the opportunity to work on this. I am grateful to him for his guidance and support throughout the design and implementation of the project and also for providing me with the resources needed to complete this project.

References