Leia Chang

Zoe Wood

May 31, 2017

Senior Project Report Visualizing Dance

0. Abstract

This project aims to explore different outlets for artistic rendering of dance movement. Here, motion captured dancer movement is visualized in 3d via an OpenGL graphics pipeline, portraying the dancer as a skeleton with various ribbons tracing the dancers' movements through space.

1. Introduction

Dance is an art form that exists in all cultures across the globe. However, it can be difficult to share dances across cultural boundaries or levels of experience; various dances embody different forms of bodily movement, and what is inherent to certain cultures can be wholly absent in another.

The audience of a dance show can see the movements of the dancers, and can understand the larger architecture of choreography. However, many audience members may find themselves at a loss when attempting to gather a deeper understanding of the dance; Body mechanics and feelings can be unfathomable to those who have never tried it. Even those who indeed already have backgrounds in dance may be at a loss for what it might feel like to pursue another form of the art. This project intends to explore a new representation of dance movement that allows a viewer to gain an intuition about the form the dance takes, as well as give dancers an extra dimension of expression to explore.

This project is a series of pipelines to visualize motion-captured dance movement as a way to provide extra context into the types of movement inherent in various styles of dance. Working from pre-existing motion capture data, movement is rendered using an OpenGL pipeline, with particular focus on spatial representation and sustaining presence of motion.
Several areas of interest here are; the pre-processing of motion-captured data, the generation of ribbon and brush-stroke geometry, and integration of this project into virtual reality.

2. Previous/Related Work

In the past, artists have used light projection and pre-choreographed routines to create visualizations similar to the ones created in this paper. While these create interesting visuals for dancers and choreographers to use, they are time consuming to create and most often are still 2-d projections. Dancers can use them to illustrate motion and add dimension to choreography, but they still do not provide audience members with a greater understanding of 3d space. Notable examples of this type of exploration into the collaboration of light projection and dance choreography include the works of Enra, the Japanese dance company behind "*pleides"* and *"Torque starter"*.

Another dance-centered work relating to this project is the use of long-exposure photography to capture motion. A most notable modern series is the work by Bill Wadman, whose work *Dancers in Motion* focused on Ballet dancers and portrayed their art as seemingly tangible clouds of motion. While this provides a more solidified representation of movement, it does not allow for the playback of the motion, and presents it as a static object rather than as evolving.

Some other related work includes artists tools created for VR; The TiltBrush, created by Google, for example. These programs allow artists to draw in 3d space using a series of colored ribbons.

However, these are simply tools, and require manual input and VR hardware to use. These are not suitable as avenues of expression directly ported from dance motion[1].
Additionally, there have been many motion-capture rendering pipelines created to help visualize the data captured from human bodies. However, these are created as tools for analysis, not for use in expression or artistic pursuits.

3. Solution

Program Overview

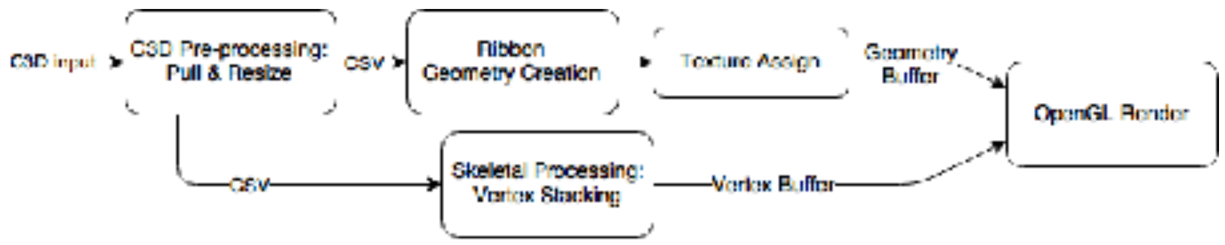


FIGURE 1: PROGRAM OVERVIEW

The pipeline for this program starts with C3D motion captured files, a filetype described in detail below. These C3D files require extra processing, and so here two sets of python scripts are used to separate and reorganize the data provided. Once pre-processed, this data is ported into the main rendering program. In this program, the data is loaded into buffers to be sent to the GPU, and are partially rendered based on time. The render style is based on pre-processed features, and is outlined below.

Motion-Capture Data: Retrieval and Reformatting

The C3D file format is a public domain, binary file type that stores both the labels and the data of a motion-capture system in a single file. The data and its corresponding labels is packed into binary, and thus requires an external program to be viewed and edited. In order to be usable by third-party apps, the C3D files may be converted to ASCII and CSV files, holding the labels and the data respectively. This file format packs data into bits as closely as possible, making its size much smaller, but making the format harder to work with. However, this data type is a proprietary type, and all software created to read and use this file type is limited to usage on the Windows OS.

[1] Some artists have experimented with using Tiltbrush and similar tools as expressions in dance (see Scribbler, by Danny Bittman). However, these require the usage of hands to draw, and are inapplicable to visualizing pre-existing styles of partnered dance.

All data used in this pipeline is in the C3D format. This format was chosen simply because of its common availability for the type of motion desired; all motion capture data used was sourced from the CMU Graphics Lab Motion Capture Database.
To make data easier to work with, files of the C3D format were converted to a generic CSV file format. This unfortunately also strips the file of its metadata, leaving only raw point values. To access the data and retrieve all associated data points, it was necessary to convert the file into an additional text file, where all metadata was listed (Including the order in which points of the body were stored).

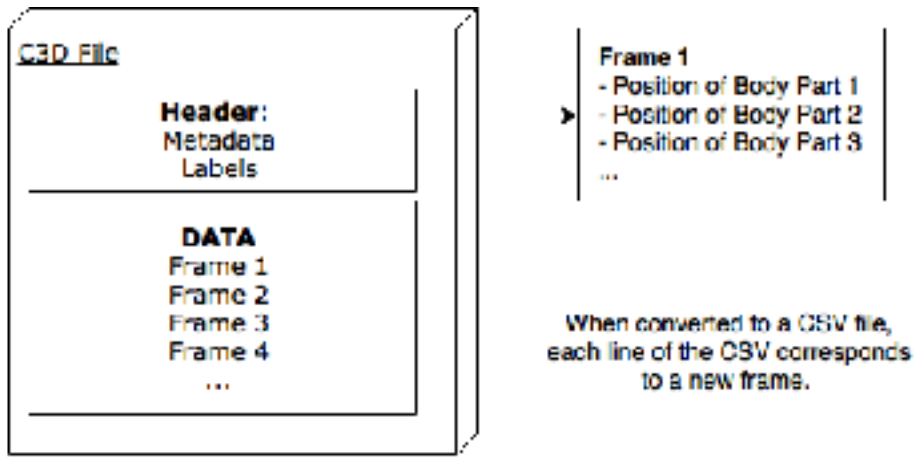
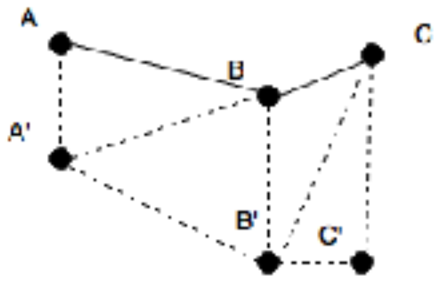

C3D File
Header:
Metadata
Labels
DATA
Frame 1
Frame 2
Frame 3
Frame 4
...
Frame 1
- Position of Body Part 1
- Position of Body Part 2
- Position of Body Part 3
...
When converted to a CSV file, each line of the CSV corresponds to a new frame.

FIGURE 2: C3D FILE ARCHITECTURE

Two python scripts are then used on this generic CSV file, retrieving the user-defined body part and subsequently resizing the data for on-screen viewing. This processed data is then saved as subsequent CSV files, which can be imported into other programs or (as seen in this project) directly copied into data structures.

Artistic Rendering: OpenGL Pipeline

Ribbon Geometry: Data used for ribbon creation is processed in an additional step. In this separate pipeline, each vertex pulled from the motion-capture data is combined with its adjacent vertices to create a third new point. These pre-existing and newly created points are replicated into a new buffer, ordered such that they create the vertices of the triangles to be rendered by the GPU. The GPU accesses these buffers based on time, each frame adding another set of triangles to the number it will render. This serves as the animation for the ribbons.



The displacement of P' from P (where P is a point in the original motion-capture data) is directly affected by the distance between P and W (where W is the previous point in the list of vertices).

This series of vertices is then listed in the geometry buffer as follows:

A, A', B,
B, A', B',
B, B', C,
C, B', C'

FIGURE 3: GEOMETRY CREATION FROM PRE-EXISTING POINTS

In order to provide a more artistic rendering, the distance between adjacent points is taken into account during triangle creation. As the vertices from the original motion-capture data are listed chronologically, each vertex is compared to its predecessor to determine its distance. As each vertex is a datapoint taken at equidistant intervals of time, we can determine that further distances are instances where the measured point is moving at a faster rate. In order to achieve a more artistic brush-like stroke, faster curves are painted with a thinner stroke, and areas where the body part lingered in space are painted with thicker or heavier strokes. This creates a more dynamic effect, as seen below in Figure 4.

In this particular instance, we calculate the displacement (in the Y direction) of P' as follows:

$$D = ((1 - (d/d_{max})) \cdot W)^2$$

Where D is the final displacement, d and d_{max} are the distance from point P to its predecessor, and the maximum distance between adjacent points, respectively. W is a constant greater than 1, as determined by the programmer.

This displacement D is subtracted from P's original Y coordinate, and the resulting coordinate is the position of P'.

Ribbon Texturing: These geometric ribbons are then partitioned into texture areas, and rendered with a stroke-like texture. This partitioning is done by 'walking' through the buffer of triangles, assigning texture coordinates (XY in pairs, each ranging from 0-1) along the vertices in a repeated loop. A simplified version of this is illustrated in the diagram below.

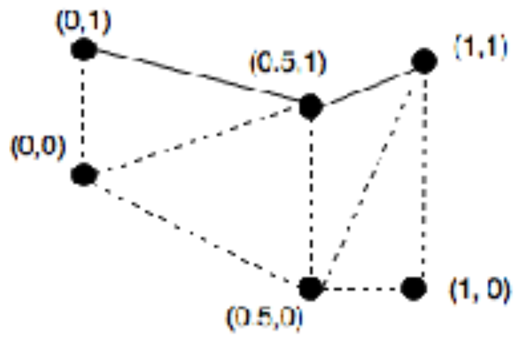


FIGURE 4: RIBBON TEXTURE ASSIGNMENT

In order to appear more like painted strokes, the shaders discard any texture pixels that are white, leaving behind only the black ink-like strokes. The results are shown below, in Figure 5. The combined effect of the shader discard and ribbon texturing creates ‘strokes’ of various sizes.

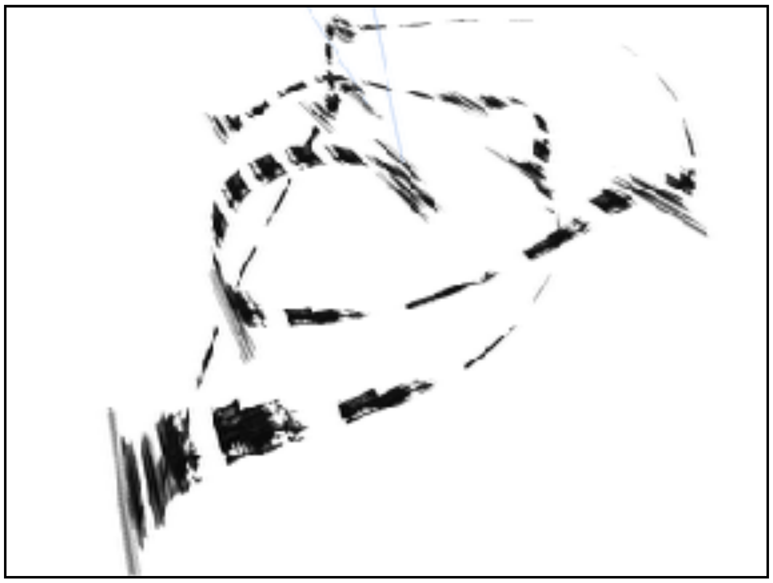


FIGURE 5: RIBBON RENDERS

Movement and Size Context: Skeletal Rendering

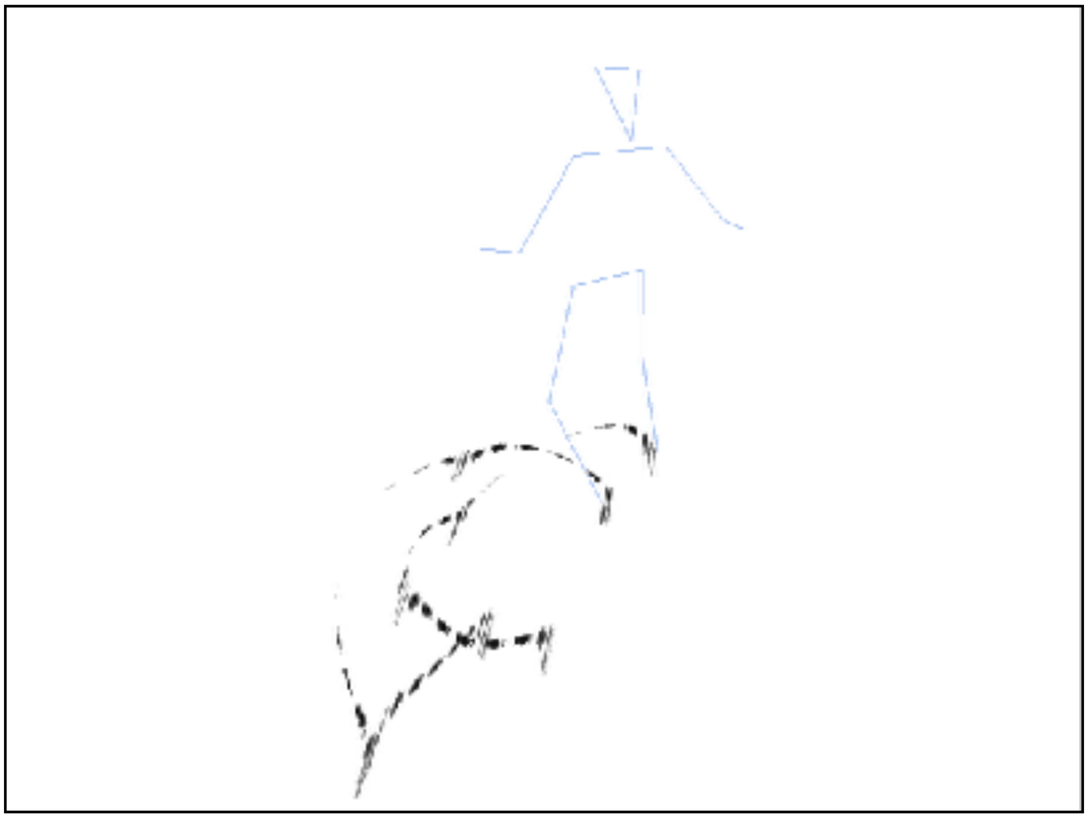


FIGURE 6: SKELETAL RENDERING

To give context to the previously rendered ribbons, it was necessary to provide a skeleton of the dancer as well. This created both a sense of scale and orientation, as well as made the movements more intuitive.

The skeletal data is processed separately from ankle-ribbon data. In order to render the skeleton as a series of lines as desired, the vertex representations must be linked together as 'bones'. Because the original motion capture data does not have any representation of linked body parts, these parts are manually chosen and then 'stacked' into line segments. This stacking is done by intermixing vertices from the 'bone' endpoints, essentially creating a new buffer of vertices in sets of two, each representing the bone in space at the time of data capture.

Two modes allow for different viewing of the skeletal movement - the first simply renders the skeleton moving over time, while the second shows all of the positions that the skeleton has taken from the beginning of the data. This second mode provides a more solidified understanding of the movement.

Mobile VR Port

Porting this same motion-capture rendering into VR was desirable in order to provide a more intuitive sense of space not available on a 2D screen. A secondary desire was for the VR experience to be available for viewing without desktop setup and specialized equipment; Thus, the project was aimed for mobile VR experiences. This portion of the project was done in collaboration with fellow student Tyler Dahl. This collaboration additionally allowed the project to be tested in iOS and Android environments.

Through Unity, the motion-capture data is loaded via a custom CSV reader. These vertices are then used to determine the translation to be applied to a custom object in the scene (in this case, a simple sphere and cube are used to represent the ankles of the dancer). These objects are assigned target locations retrieved from the position buffers based on current position. We perform a LERP interpolation onto the object's path, with the start point being its current position, and the end point being the next position retrieved from the list. As the object reaches the next target position, we increment the end point to be the subsequently listed position. This translates the object smoothly through space and along the path of the dancers as captured in our data.

To visualize this path similarly to the ribbons in OpenGL, the objects have an added Trail Renderer component. This component traces the object's path in space with ribbon-like geometry. While this does not take into account the speed of the movement, we are able to taper the ribbon at its start and end, creating a more dynamic ribbon, and providing the illusion of a growing and shrinking trail.

4. Results

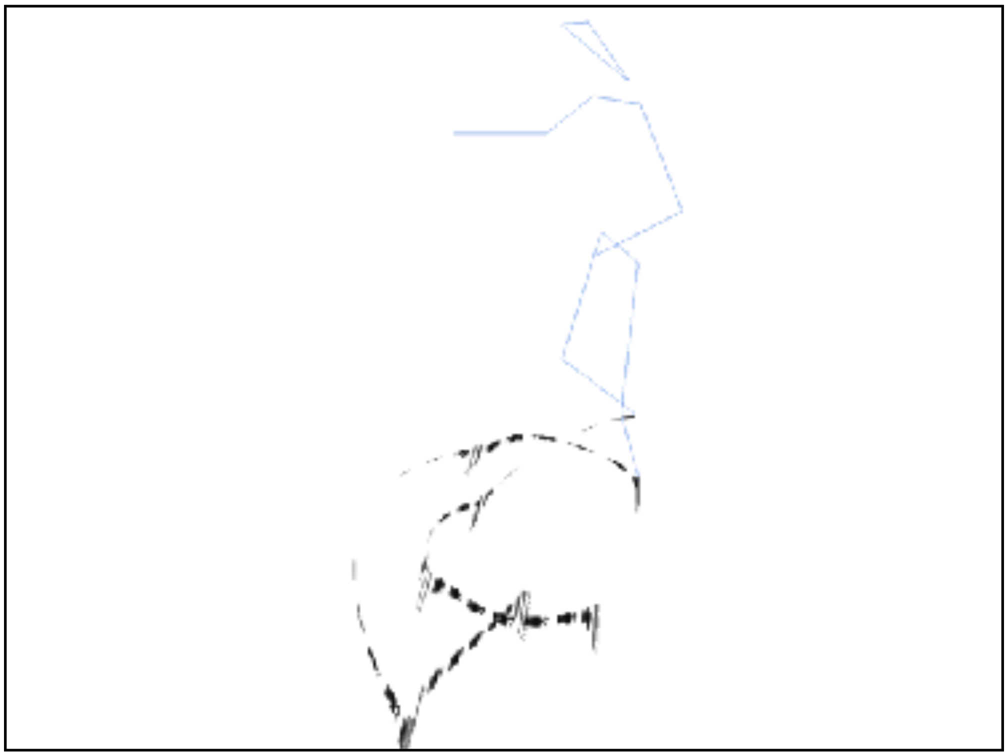


FIGURE 7: MODE 1 - DEFAULT VIEW

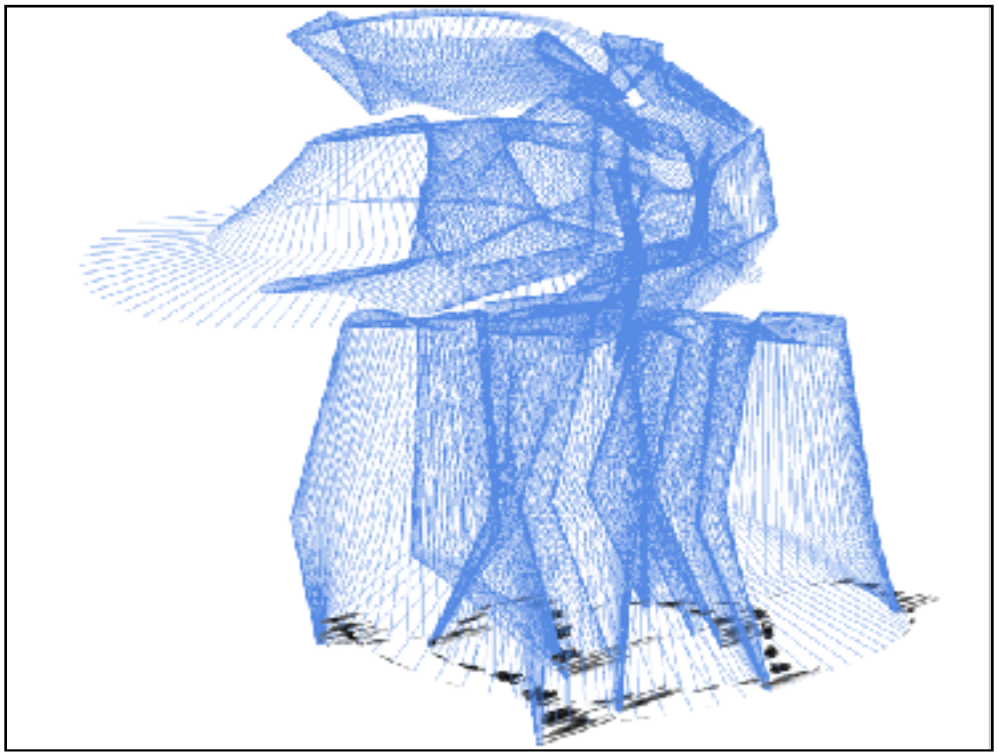


FIGURE 8: MODE 2 - ALL SKELETAL POSITIONS

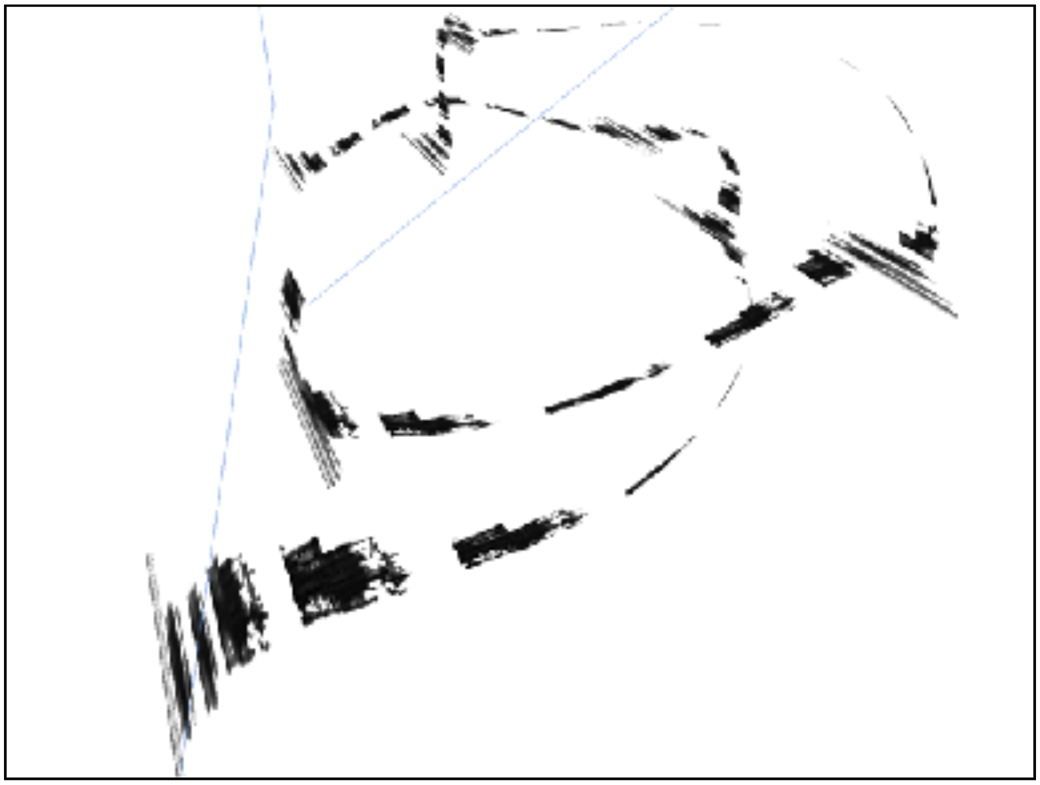


FIGURE 9: CLOSEUP OF RIBBON TRAILS

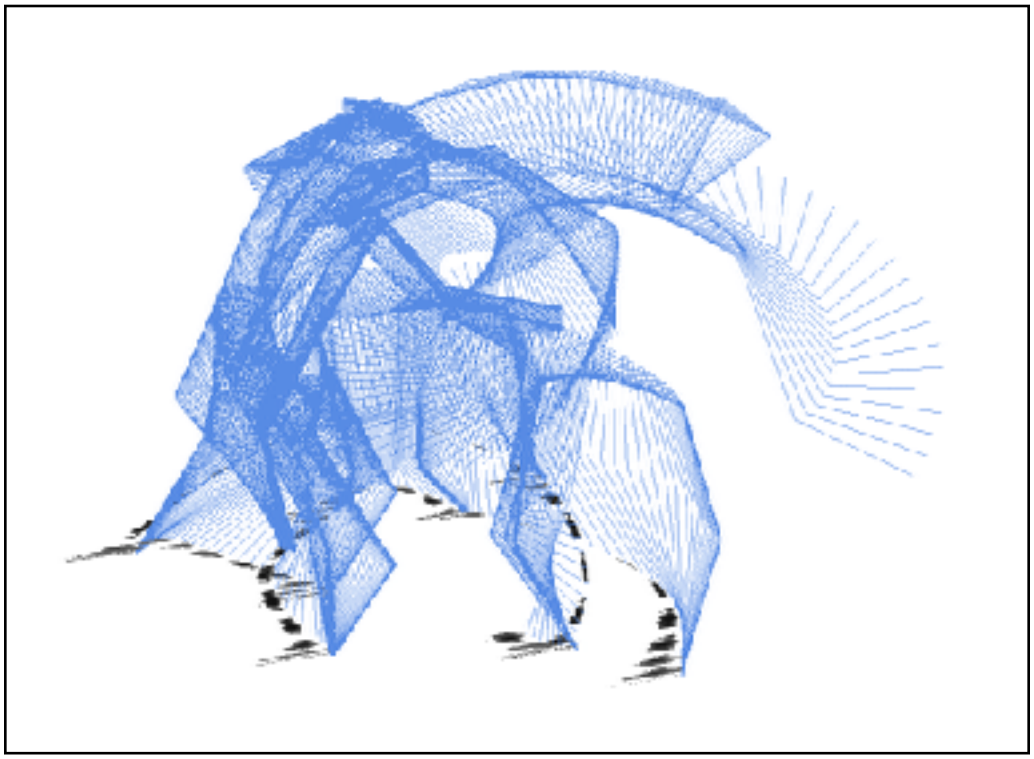


FIGURE 10: ALTERNATE VIEW OF MODE 2

As seen above, this program successfully portrays the movement of the dancer. The in-program camera allows for the user to view the dance from any angle, via the WASD keys and mouse-tracking. The motion and rendering can be paused and restarted at any point in time, using the space bar and the R key, respectively. The L key toggles between the two modes.

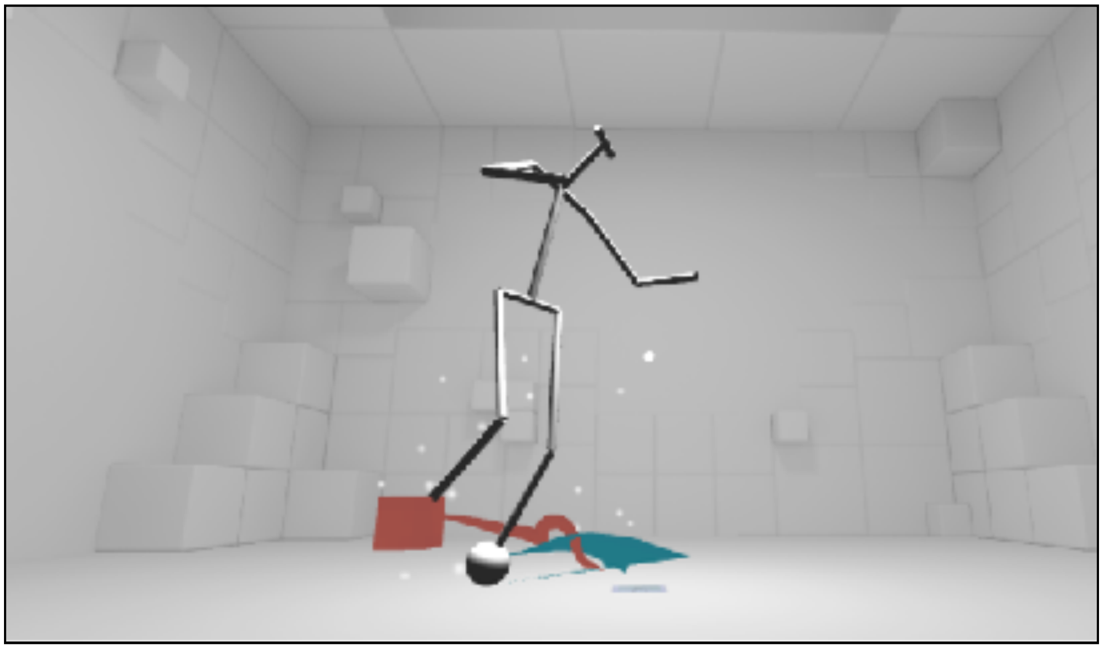


FIGURE 11: SCENE VIEW OF VR PORT

In the VR port, movement of the camera is controlled by the tilt of the mobile phone. Movement is only triggered when the user taps the screen. Additionally, the camera creates a trail as well, allowing the user to create their own movement trails in conjunction to those created by the motion-captured data. This addition of user interaction really pulls the user into the space and into a mode of creativity.

5. Reflections

While motion-capture data is plentiful, the actuality is that the data was messier than anticipated. This dataset was particularly difficult to work with. Not only was the data format created for a Windows-only environment[2], but the data labels were mis-ordered and required extra steps for retrieval. Additionally, the data appeared to be turned sideways; During final rendering, all of the objects required rotation by negative ninety degrees about the X axis. Should the project be attempted again, it would be advisable to research file formats further, and use one with more flexibility across environments.

Additionally, OpenGL, while powerful, proved to have some unexpected limitations. On the particular computer that this was developed on, variable line width had been deprecated, and therefore only lines of one pixel in width were available for use and rendering. Beyond this, however, OpenGL was sufficient for the project.

While mobile VR is the most portable and has low barrier to entry, the resulting experience is novel but not meaningful. The inability to physically move around the scene removes some of the physical connection to the movement being rendered. However, the mobile VR experience does adequately provide a representation of the movement and sense of space as desired, and the ability to interact with the scene and create extra ribbons in the space seemed to provoke curiosity from users. In retrospect, switching focus to more interaction and adding modes to engage creativity may have been desirable.

6. Future Work

Though the pipeline for using this motion-captured data is in place, it still requires much user input and intervention. Because many motion-capture datasets are hand-labeled, body parts and vertices must be hand-picked from the labeling scheme in order to be rendered. In the future, it would be advisable to make this first step user-specified, and the automate the remaining parts (retrieval of data, and conversion into spatially logical area). While the linking of bones would still be necessary, being able to do this as a part of the program setup would be ideal. Additionally, updating the pipeline to accommodate motion capture file types would be an area for improvement in the future, as the current pipeline uses a mixture of C3D and CSV files.

An original but unfeasible goal for this project was to accumulate the data in real-time, or allow data to be tracked by users themselves for rendering. At this time, positional tracking through mobile devices is not possible. (Most positional tracking via mobile devices is done through

[2] There is one program available for C3D viewing/editing on MacOS; However, this failed to work in the author's environment and for these purposes is ignored.

satellite tracking, which is sufficient for projects requiring less fine-grain tracking such as Pokemon Go[3].)

Finally, an area of future exploration (as an alternative to mobile VR) is to move this project into full VR/AR integration with existing systems designed specifically for this purpose. Some work has already been done in using controllers to motion track movement beyond gestural movement, particularly as 'windows' between dimensions (ie. between the real world and a virtual world)[4]. This work could be continued into rendering live motion in VR, and overlaying this artistic rendering into the real world via AR technologies.

[3] Similarly, Ingress by Google used the same technology and cell phone tracking to engrain real-world movement into the game.

[4] Vision 2017 - Making Magic Across Both the Virtual and Real World at PlayStation

7. References

Bill Wadman (n.d) *Dancers in Motion*. Retrieved from https://www.billwadman.com/motion/

Carnegie Mellon University (n.d) *CMU Graphics Lab Motion Capture Database.* LindyHop2 retrieved from http://mocap.cs.cmu.edu/search.php

Dr. Andrew Danis (2016, November 6) *An Overview of the C3D File Design*. Retrieved from *https://www.c3d.org/overview.html*

Enra, Japanese Performance Arts Company (n.d) Retrieved from *http://enra.jp/*

Dock (2012, January 10) CSVReader. Retrieved from http://wiki.unity3d.com/index.php?title=CSVReader

Danny Bittman (2017, January 11) Scrubbler | Tiltbrush in Real Life. Retrieved from https://vimeo.com/198976363

Getting Started in Google VR in Unity (2017, April 18) Retrieved from https://developers.google.com/vr/unity/get-started

Steven Osman - Sony (2017, May 17) *Vision 2017 - Making Magic Across Both the Virtual and Real World at PlayStation*. Retrieved from https://youtu.be/d4ntbWK-jwc

Tutorial 5: A Textured Cube (n.d.) Retrieved from http://www.opengl-tutorial.org/beginners-tutorials/tutorial-5-a-textured-cube/