A Pipeline for Synthesizing Action-conditioned Human Motion from Raw Motion Capture Data

by

Ritaank Tiwari

S.B. in Computer Science and Engineering and Mathematics
Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2023

Authored by: Ritaank Tiwari
Department of Electrical Engineering and Computer Science
August 18, 2023

Certified by: Praneeth Namburi
Research Scientist
Thesis Supervisor

Certified by: Tony Eng
Senior Lecturer
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

A Pipeline for Synthesizing Action-conditioned Human Motion from Raw Motion Capture Data

by

Ritaank Tiwari

Submitted to the Department of Electrical Engineering and Computer Science
on August 18, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In many sports, less-experienced trainees will often draw inspiration from videos of experts. While this can be an effective tool for improvement, this process lacks the ability for the trainee to specifically focus on improving their skills based on the limitations of their current abilities, body type, and weaknesses.

Since sports are very competitive, there exists a need to convert expert movements to a series of standardizable forms and movements that can then be pedagogically applied to the differing needs of various trainees: specifically, their different abilities, body types, and weaknesses.

Effectively, this conversion requires a pipeline that can take an input of motion capture data, automatically label the markers used, create a skeletal representation, and then train a machine learning model to accurately synthesize human motion, conditioned on the action type.

The outputted motions can be rendered for any body type, and could be customized to the trainee. The designed pipeline is not fencing specific – it is highly adaptable to the nature of the data or sport, robust to errors and noise, as well as tightly integrated in an easy-to-use library.

Thesis Supervisor: Praneeth Namburi
Title: Research Scientist

Thesis Supervisor: Tony Eng
Title: Senior Lecturer

Acknowledgments

I would like to thank the amazing team at the MIT.nano Immersion Lab for their incredible teamwork, passion, and knowledge. Specifically, I thank Praneeth Namburi for his mentorship, care, and for providing me the opportunity to join the lab. I also must extend my gratitude towards Brian Anthony and Talis Reks, as well as my fellow colleagues Roger, Jess, and Duarte, for creating such a great community. Enya and Niles – your efforts made my research possible.

To Tony Eng, I deeply appreciate you being an advisor through this last year, and for bringing me on to the 6.UAT staff. I learned so much teaching alongside you.

I thank my fencing coaches Jarek Koniusz and Robert Hupp: thank you for creating wonderful culture in the MIT Fencing Team. It kept me grounded throughout many difficult times at the institute. I would surely not have found this incredible research opportunity without them.

Last, but not least, I thank my family: Mama (Anuradha), Papa (Sandeep), and brother Vedaank, for their unconditional love and support throughout my life. To my friends – my close friends – you know who you are. I'm so lucky to have you all with me. I'm better every day because of it.

I recognize that I stand on the shoulders of giants, both as a man and as a researcher, and for that I am eternally grateful.

Contents

List of Figures

Chapter 1

Introduction

Motion capture refers to a set of technologies to record the movement of people or objects in an accurate way such that the recorded data can be imported into a computer, visualized, and manipulated. The use cases are broad, from animation, gaming, and movie-making, to sports, virtual reality (VR), and augmented reality (AR) experiences. With the growth of AI and virtual reality, this market is expected to grow from $180 million dollars in 2022 to $600 million dollars by the year 2030 [15]. Our research focuses on the use of motion capture data in the context of sports, specifically fencing, with the end goal of improving the process of athlete practice.

1.1 Motivation

Less-experienced athletes often draw inspiration from videos of experts, whose movements have evolved through several stages of refinement. There are several problems with this destination-oriented approach. Copying the movements of experienced athletes before the requisite abilities and skills have developed can increase (1) the predisposition to injury, (2) the development of poor technique, and (3) an overall delay in the learning process due to learnt bad habits. Practice does not make perfect – practice makes permanent. Furthermore, videos often occlude important aspects of the movement, as their perspective is limited. Even multi-camera setups, such as those in professional sports leagues, face this limitation [5].

1.2 Preliminary Work

We are using sports fencing as the medium for tackling the above problems. Preliminary work done by the MIT.nano Immersion Lab suggests that self-paced fencing practice conducted in an immersive environment can improve athlete training and response. An immersive environment refers to the use of extended reality, or XR for short. Extended reality is a blanket term covering all interactions of the real and the virtual, including AR and VR. In this preliminary work, a fencing trainee was subjected to three experimental conditions. In the first, the trainee watches a video demonstration of a move and then replicates it from memory. In the second, the trainee mimics the action concurrently with the video demonstration. And in the third, the trainee wears a virtual reality headset and mimics a demonstrator performing the move in virtual reality. The sequence of these conditions was randomized.

An empirical analysis of the motion data conducted by fencing coaches led to the conclusion that the movements in the VR condition were more compact and consistent, more closely recreating the demonstrator's movement. Figure 1-1 summarizes the experiment and visualizes the results [5]. Figure 1-1, panel (d), shows, in purple, the trace of the sword tip through 3D space from the beginning to the end of the movement. The white figure on the left represents the subject in the ready position. Panel (e) shows the trajectories of the tip again, this time split according to the experimental condition, from the subject's horizontal point of view. Left to right, these modalities are replicating video, mimicking video, and mimicking VR. Here, the idea of 'compactness' can be seen, with the tip having much less displacement in the right-most (green) data. Panel (f) shows a birds-eye view of the same data as (e) [5].

This experiment indicates the value of immersive experiences in athlete training. In some ways, this is not surprising, as the VR experience creates a realistic, multidimensional, and unoccluded view of the demonstrator, especially since the subject is free to move around in the virtual world and view the demonstrator from all angles. Figure 1-2 depicts the immersed VR view which is shown through the headset [17].

1.3 Thesis Objective

The goal is to build a pipeline that leverages existing generative modeling techniques to train a model that can return synthetic, action-conditioned human body motions that match the subject's body type. To train this model, we collect training data of fencing movement, process it, compress it, and then train a generative modeling architecture.

It may appear that simply recording data of many different fencing actions, each performed across different skill levels, and replaying them to the trainee as requested could be a simpler solution to having custom-generated actions. However, such an approach has several limitations, many of which we have mentioned above: it fails to factor in body types, expertise levels, and trainee weaknesses. The inherent randomness of the generative model aligns with the stochasticity of fencing. Additionally, the architecture we use allows us to match the generated motion to the subject's specific body proportions, providing a much more realistic view of the action for the trainee. The actual human testing of this synthetic training data and an associated evaluation of learning is beyond the scope of this thesis; we focus on building the pipeline for such experimentation to be done in the future.

We must briefly introduce some understanding and vocabulary to better discuss our objectives. In motion capture, data collection is performed by placing small markers on the subject's body, which a multi-camera system is then able to track in real-time to capture the data. These markers are small orbs which reflect the infrared light emitted by the infrared tracking cameras. Motion capture data can be collected in primarily two ways: with active markers or passive markers. In both approaches, markers are physically attached to the body at key joints and positions so that the key aspects of the subject's body and movement are recorded. In the OptiTrack [6] system in the Immersion Lab, a calibrated, multi-camera system emits infrared light which reflects off the markers to inform the camera system of the marker's exact location. In an active marker system, each marker has its own power source, which can in turn power an LED so that each marker flashes with a unique pattern. Thus,

the recorded data on an active system inherently knows **which marker is which** (called a marker label), as opposed to a passive system. If there is a marker on a subject's hand, and the subject puts that hand in a pocket, and then removes it some time later, the active system can recognize it as the same marker from before, whereas the passive system cannot.

In the passive (unlabeled) system, collected data is essentially represented as a vector of shape $F, M, 3$ (number of frames by number of markers with an x, y, z coordinate to represent each marker through time. This unlabeled data is often called raw, or referred to as a mocap point cloud (MPC).

1. Our pipeline shall be fully end-to-end from data recording to motion synthesis.

   This objective basically states the core work of the thesis. From some raw motion capture data, we build a pipeline combining many radically different components: marker-labeling (labeling the markers with their associated body part automatically), body-solving (determining the actual shape and orientation of the human body from a cloud of markers), post-processing and data-alignment, followed by the training of generative models. To be clear, this work does not introduce a new model architecture for this objective. Instead, we leverage the state-of-the-art architectures, and build interfacing, preprocessing, and postprocessing between each often-disparate module to allow for a smooth data flow, all the way from the point of data capture to the generation of synthetic motion (also called motion synthesis). The first half of the pipeline for automatic labeling solves a huge pain point in the lab to label data captures, which are currently done by hand.

2. Our pipeline shall be intuitive and easy to use.

   The MIT.nano Immersion Lab attracts scientists and engineers of all disciplines, with unique strengths and skills. Thus, the key functions of our designed pipeline must be intuitive to use and well-documented. Tools such as DeepLabCut [12] are widely popular in pose estimation in motion research for their low barrier of entry and their intuitive interface. We aim for a similar ease of use.

3. Our pipeline shall be robust to all types of human motion data, whether the markers are labeled or not, as well as the inherent noise in the data as well.

   After data is collected, marker labeling takes place, which refers to the process of manually analyzing the data and assigning part-of-body labels to each marker for all the frames. Existing software does simplify this process somewhat, but even still, marker labeling remains a key pain point of the lab, as it is a prerequisite for the analysis of almost all data that the lab collects. Automatic labeling tools exist, but they lack accuracy and robustness in cases where a specific marker falls off or is occluded. We design our pipeline to be able to both process already-labeled data, as well as automatically label if needed using the SOMA architecture [2]. We design this pipeline with fencing pedagogy in mind, but it shall work with all types of human motion data. Furthermore, the pipeline we construct is robust to the noise and inevitable inaccuracies in the motion capture data collection process. This is a result of a combination of using already robust architectures as well as intelligent interprocessing.

Figure 1-1: Experimental approach: (a), (b), and (c) show the three different modalities: watching then replicating on video, mimicking video, and mimicking in VR. Section (d) shows a connected body and sword with the tip trajectories overlayed. In (e) and (f) we see two different views of the same data, going left to right are datasets from setup (a), (b), and (c) respectively. We see that (c) shows data which is both more precise and accurate [5].

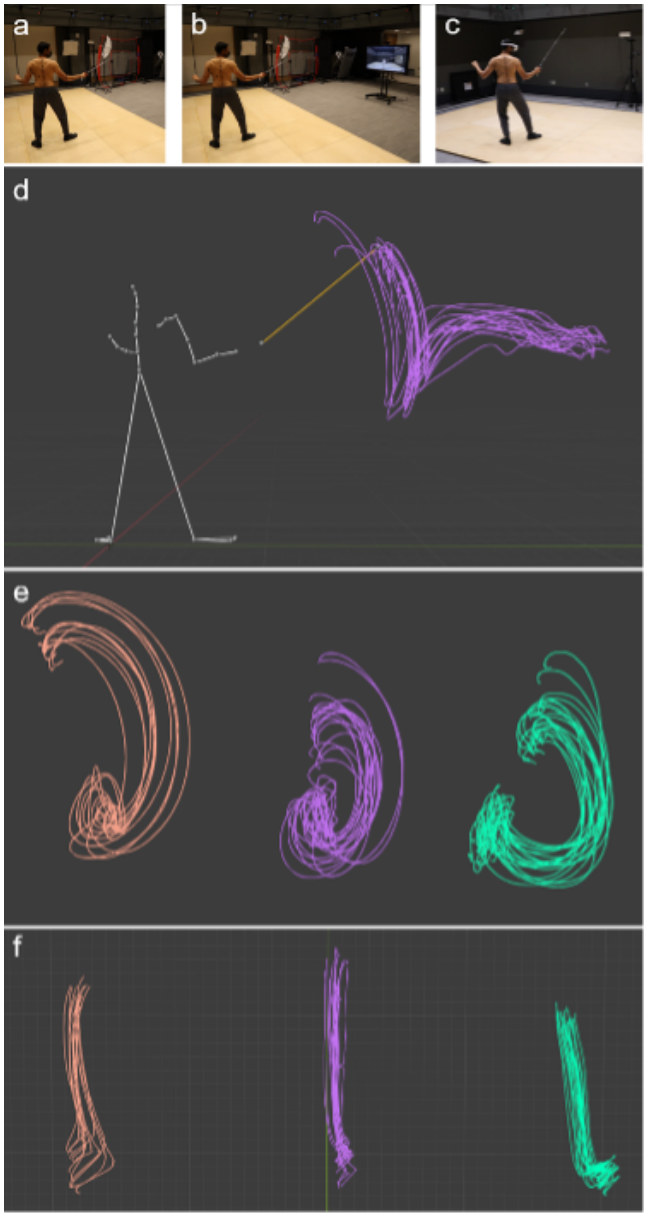


Figure 1-2: The view of the demonstrator through the VR headset, flattened into an image.

Chapter 2

Background

In this section, we will explain the key related works and modeling approaches that our work implements and extends. Namely, we introduce SMPL [9], a realistic 3D model of the human body, which is able to express a rich human body from a small set of parameters. We later introduce the variational autoencoder (VAE) [8], a generative model which learns to mimick the input data passed in. We use the SMPL body model extensively to represent our motion capture data, especially since its representation is particularly appreciated by the VAE.

2.1 Body Modeling

A human body model refers to some framework for representing the human body in a simplified way. The ultimate goal is to have a rendered mesh of the human body that can move/be controlled by changing just a few parameters. Rendered meshes contain thousands of vertices, and as such, a simplified representation is a highly effective way to compress and represent the body. A simple example could be a body model which represents a human body as a stick figure skeleton, which calculates a mesh around the skeleton by calculating each vertex position relative to the skeleton joints near it. This approach is not far from Linear-blend skinning (LBS), a key aspect of the SMPL model, which we describe below.

2.1.1 Linear-blend Skinning

LBS goes by many names: smooth skinning, blended skinning, and sometimes just skinning. Skinning is simply a method for calculating a mapping between rigid joints and a surface – like a human skeleton and skin. It is the basis of all animation rendering, allowing a complex mesh to be controlled by a simple set of underlying joints. In smooth skinning, each surface vertex of the mesh can be connected to multiple joints, with different weights associated to each joint. For example, the vertices in the forearm are simultaneously affected both by the elbow and wrist. A logical assumption is that vertices closer to the wrist attribute a higher weight to the wrist, and vice versa. This is the assumption made in LBS, and the final vertex position is calculated as a linear combination of the weights of the related joints. One final piece of this puzzle is the template pose $\boldsymbol{T}$, which contains the information on the default relationship between surface vertex and underlying joints. The linear combination of the template pose, as well as the weights of related joints, is where the name linear-blend skinning originates.

Let $\boldsymbol{N}$ be the number of vertices, and $\boldsymbol{K}$ be the number of joints in our underlying skeleton. As the body moves, the joint locations and rotations of the underlying skeleton change. The exact parameters of linear-blend skinning are:

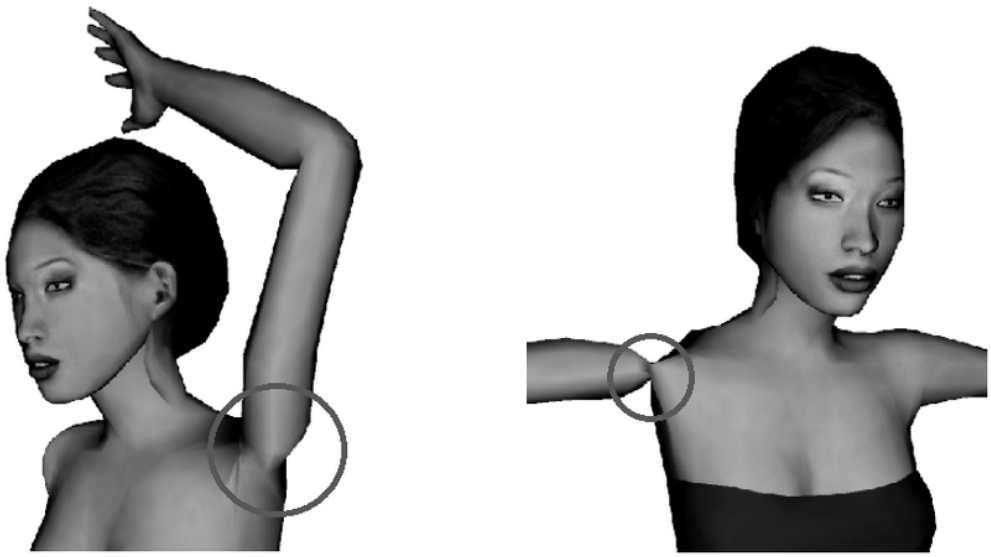
1. Blend skinning weights $\boldsymbol{W} \in \mathbb{R}^{N \times K}$
2. Joint locations $\boldsymbol{J} \in R^{3K}$
3. Joint rotations $\boldsymbol{\theta} \in \mathbb{R}^{3K}$
4. Mesh vertices in the template pose: $\boldsymbol{T} \in \mathbb{R}^{N \times K}$.

As a final note, LBS uses a kinematic tree to relate the joints in the skeleton together. As such, each joint in the tree has exactly one parent joint, and child joints (unless it is a terminus). The SMPL kinematic tree can be seen in Figure 2-2, with branches coming out of the pelvis, which is the root joint in this construction. As such, the joint rotations θ stores each rotation *relative* to its parent. Thus, the final

skinning process involves a recursive calculation up the kinematic tree, to first find a global orientation for any given joint. These rotations, along with locations, the default mesh, and blending weights are passed into LBS to produce a mesh from the base skeleton representation.

As a final note, a drawback of the LBS algorithm does not include any understanding of how a human body is constructed – it simply strives to create a smooth mesh from calculations of linear combinations. Human body deforms in different ways, many of which cannot be represented by the stick-figure skeleton representation of the human body. Consider for example the widening of the hips when a human attempts the splits. Without considering these unique aspects of the human body, the LBS algorithm can have many strange artifacts, with one of the most well-known being the candy-wrapper effect [20] as shown in Figure 2-1.

Figure 2-1: At joint intersections, linear blend skinning, although creating a smooth vertex mesh, can create completely unrealistic outputs. Pictured is the popular candy-wrapper effect [20].



2.1.2 SMPL

SMPL improves on the key shortcomings of LBS, while also extending the body model to include all kinds of human body shapes and types. Where previously the

parameters $\boldsymbol{W}, and \boldsymbol{T}$ would need to be manually crafted per subject in LBS, SMPL uses a condensed set of 10 shape parameters, collectively known as β, which are able to represent just about every realistic body shape possible.

At the high level, the SMPL model is additive by nature. It does not replace the LBS approach – instead, the formulation adds corrective components that combine with the existing LBS approach to prevent the candy wrapper effect, represent a variety of human shapes and sizes, and form more realistic outputs. Specifically, the SMPL model maps 23 joint rotations (24 after including the global rotation), as well as 10 shape parameters (β to an output mesh of 6890 vertex locations. The 6890 vertices form a mesh of the human body being represented. The joint rotations effectively capture the subject's unique pose, and are thus also called the pose vector, referred to as θ. θ represents rotations in axis-angle notation, which requires 3 parameters for a rotation. This gives $(24) * 3 = 72$ parameters required to represent pose θ, and 10 parameters for the body shape β. Figure 2-2 shows a graphical representation of the SMPL joints underneath a mesh [4] [9]. SMPL incorporates this construction with a set of three pre-trained models (for the male, female, and neutral body). This pre-trained model takes θ and β as input, performs skinning, and outputs a result mesh.

One of the key innovations in the SMPL model is that it is an additive model. We mentioned above that SMPL adds corrective components that are combined with the LBS skinning approach to be more realistic to the actual human body. One of those corrective components we have already introduced: β, which adds a correction for the specific shape and proportions of the body we wish to render.

Shape Blend Shapes:

Shape blend shapes modify the template mesh to encode the different body shapes that people have. Principal component analysis (PCA) was performed on shape displacements on a dataset of individuals of varying shapes, and the SMPL model uses the 10 most significant principal components to represent the different shapes. These parameters are referred to as β. These betas are human-interpretable, with the first β parameter primarily modifying height proportionally. Other β values affect hips,

waist, chest, limbs, and their proportion. It is possible to use more than 10 β values, as the shape blend shapes are calculated as a simple linear combination of the given β values for an individual with the matrix of principal components of shape displacements. SMPL picks 10, as their PCA analysis showed that an overwhelming majority of the variations in body shape could be represented in the 10 largest principal components.

Pose Blend Shapes:

Pose blend shapes solve the candy-wrapper problem as described above. These pose blend shapes effectively act as a correction between the rough skeleton approximation of the human body to a more realistic one. For example, a subject doing the splits would receive a pose blend correction in the hips to widen them, better representing the human body. The parameters of the pose blend shape function are learned from datasets of high-resolution scans containing subjects who each perform many poses; the specifics of how these were learned can be found in [9]. Importantly, the template position (noted $\theta*$) has no contribution of pose blend shapes, as there is nothing to correct – it is the standard position. From another perspective, the mesh vertices in the template pose (noted $\boldsymbol{T}$) represent a form of a pose-blend shape for the template pose. LBS does the erroneous assumption that the shape mapping stays the same through all poses, instead of it being pose-dependent. In conclusion, the corrective blend shapes add on to the template mesh $\boldsymbol{T}$ to deviate the template mesh for higher accuracy.

The combined process of the SMPL algorithm can be seen in Figure 2-3. This is a very powerful visualization. It shows the template mesh on the left, and moving to the right shows progressive SMPL corrections which are added in. In the second mesh, we add in shape blend shapes from β. The third image incorporates pose blend shapes B_P, and finally, the modified mesh is skinned. In this graphic, dual-quaternion blend skinning is used, which is interchangeable with LBS. The skinning algorithm takes the modified additive mesh and outputs the final vertex locations.

SMPL offers an intuitive model to represent human motion in a unified and compressed way. This model framework is key to enabling our pipeline. To conclude,

using the pre-trained SMPL models, with an input of θ and β, a global root rotation, and an optional global translation, we can generate a realistic mesh of a body in the desired pose.

2.2 Variational Autoencoders

The VAE forms the backbone of the generative modeling architecture we employ in our pipeline. Essentially, the VAE takes in input data, and learns a process to generate new data *similar* to the inputted data. It does so by numerically extracting the key properties of the input data, and then generating new random data which is still grounded in those same extracted properties. The connection between the VAE and SMPL is as follows: we will be passing human motion (fencing) data, represented as SMPL parameters, into the VAE for training. As a result, the VAE will output parameters in the SMPL space, which we can then use in turn to easily render realistic human movement meshes of bodies performing fencing actions. The SMPL body model gives us a compressed and noise-resistant parameterization of motion, and this ends up being critical for the VAE to learn a good representation.

To understand variational autoencoders, we must first review the autoencoder which it is based on. Autoencoders are an approach to representation learning by compression. In an autoencoder, we take some input x and pass it through an *encoder*, f which is a feedforward network that maps to some compressed latent representation $f(x)$. This latent representation is then passed into a *decoder* g, typically a mirror of the encoder, outputting a vector that is the same shape as x. We call this output x', so $x' = g(f(x))$.

The autoencoder then compares x' to x, and their difference is known as *reconstruction loss*. The decoder determines if the compressed latent representation still contains enough important information of x's data such that we can extract a representation close enough to x from the latent. This indicates a 'good' compression. The training process is fully self-supervised, as no ground truth labels are needed, and the weights for both the encoder and decoder are learned. After training, if successful,

the encoder f gives an effective compression of data. The goal of an autoencoder is to learn a good latent representation. If the latent encoded representation $f(x)$ is not expressive or large enough to encode the whole input, then the reconstruction loss will be high. The autoencoder will also fail to be effective if the layers of encoding (and decoding) are not effective in helping to extract the essence of the input. This reconstruction loss objective takes an L_2 norm, and can be expressed as:

$$f*, g* = \arg\min_{f,g} \mathbb{E}_x ||x - g(f(x))||_2^2.$$

It might appear that by using the decoder, we could effectively generate new samples of data by inputting a random latent value into the decoder. In a normal autoencoder, this isn't quite possible, since the latent space is not necessarily *dense*: there are likely many possible latent values that decode to jumbled-up garbage. To truly be able to sample from the latent space, it must be dense, meaning the architecture must be encouraged to tighten the distribution of this latent space. Figure 2-4 [16] shows the difference between a sparse and dense latent space. The variational autoencoder (VAE), introduced by Kingma and Welling in 2013 [8], solves this problem. Essentially, VAEs modify the objective to learn latent variables which are the means μ and the standard deviations σ of a mixture of Gaussians. The number of Gaussians is a hyperparameter d, representing the size of our latent space, and in many cases we might just have $d = 1$. Effectively, this architecture encourages the encoder to work in such a way that it produces and **fills** a set of Gaussian distributions.

Filling the distribution is important to create a dense latent space, and the only way to force the dense filling of the whole distribution is to add a stochastic element in between encoding and decoding. Otherwise, the architecture will learn pathways for specific sets of μ and σ only, with 'garbage' being outputted for the rest of the values on the distribution. However, simple stochastic sampling makes the VAE non-differentiable, so a reparameterization trick is used. The reparameterization trick replaces simply sampling from a distribution, noted as $z \sim \mathcal{N}(\mu, \sigma^2)$, which is non-differentiable, with a differentiable $z = \mu + \sigma \odot \epsilon$. We create a new variable $\epsilon \sim \mathcal{N}(0, 1)$,

and element-wise multiply it with our encoded σ. The $\odot$ represents element-wide multiplication. If we now consider backpropagation, the gradient will not be able to do a backward pass through ϵ, but we are indifferent, as we only need gradients to flow through μ and σ to train the encoder effectively. Our trick allows a stochastic input into our decoder, while keeping all stochasticity inside the noise variable ϵ, making our network differentiable once again.

Variational autoencoders include the same reconstruction loss as autoencoders, but they also introduce a new loss to manage the forming of the latent distribution. Specifically, we assume that every one of our d latent Gaussians will be centered around 0 with constant variance 1. We thus penalize per dimension d to punish any deviation of the latent distribution from this assumption. Such an assumption allows us to safely sample the latent space upon successful training as we know the expected means and variances. We use Kullback-Leibler divergence to calculate the divergence between distributions, and formulate the KL-loss as follows:

$$\sum_{d=1}^{D}(\mu_d^2 + \sigma_d^2 - \log \sigma_d - 1).$$

Combining the KL and reconstruction loss creates an effective loss function for the variational autoencoder. A successfully trained VAE allows us to randomly sample from the latent space and receive well-formed, synthetic, decoded outputs that are very similar to our input data. Using fencing data represented in SMPL parameters, we can train our VAE. The VAE can then be sampled to generate new, synthetic, motions of the fencing actions which the VAE was trained on. To be clear, an action refers to a concept in fencing, such as a lunge, or an advance, whereas a motion refers to data (either recorded or synthetically generated) of a body performing an action. With this high-level understanding clear, we will now discuss the precise methodology of the steps in the pipeline.

Figure 2-2: The 23 joints of SMPL, shown underneath a rendered mesh. The root vertex can be seen around the waist, and it represents the global rotation of the body in the coordinate system. It is important to note that only the joint rotations are inputted into SMPL, not the 3d locations. The distances between joints for a specific rendered mesh are determined by the β parameters, which affect the shape and proportions. Figure from [4].

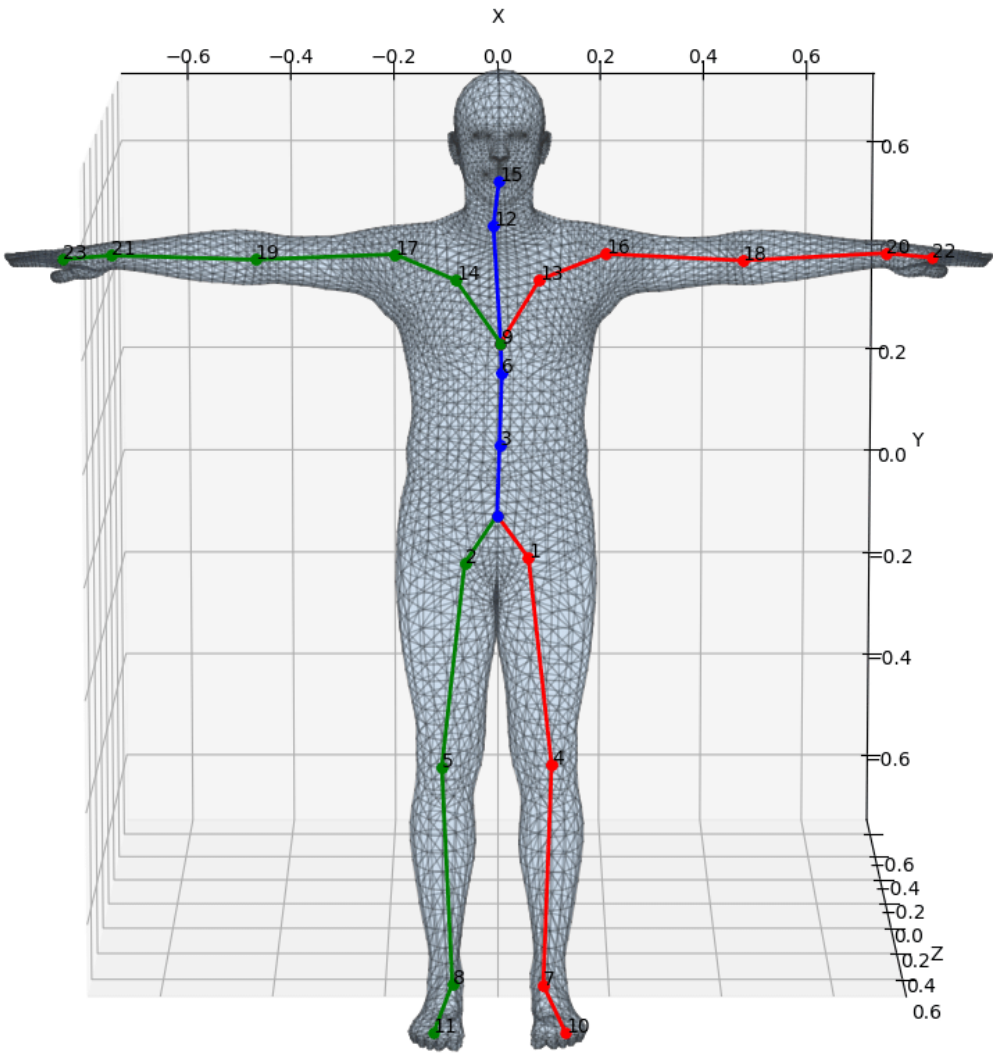
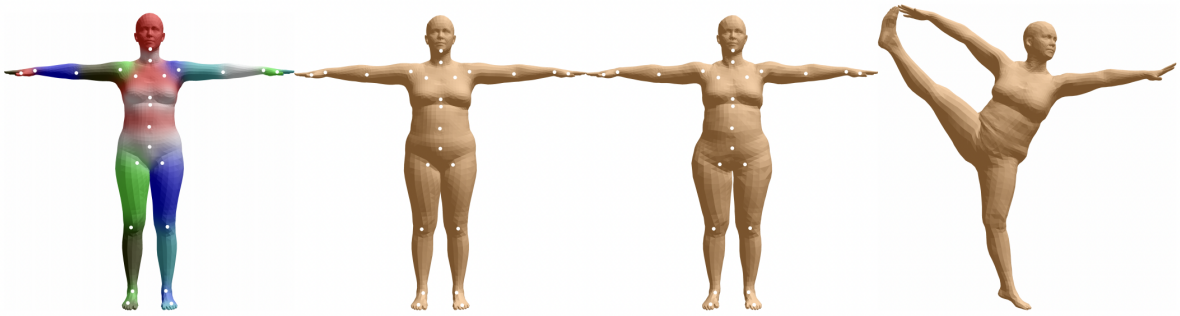
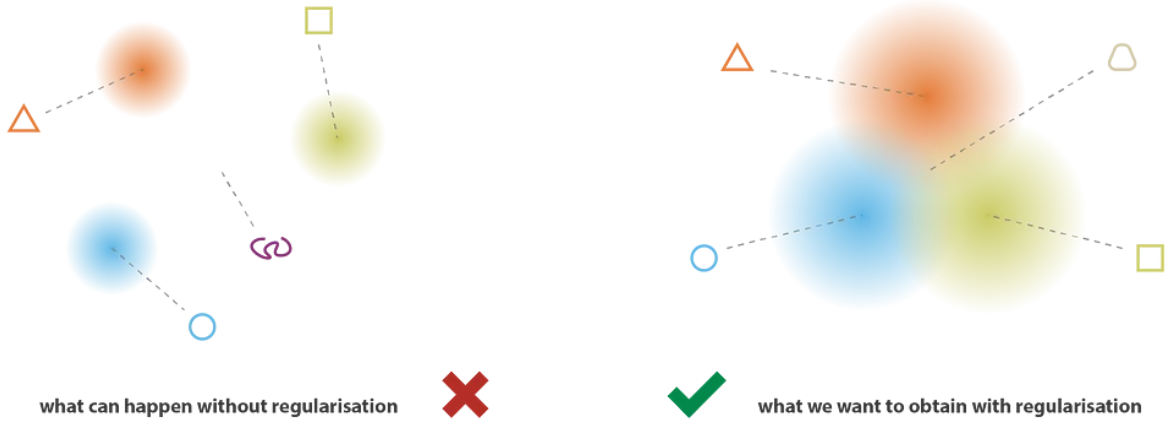


Figure 2-3: Figure from the original SMPL paper [9]. The template mesh is in (a). In (b), we see the mesh change in terms of proportion as by β. In (c) the pose-specific changes are introduced. Notice the hips are wider to match the wide-legged acrobatic pose the subject is performing. Where (b) stays the same for the subject, (c) changes in each pose vector θ. Finally, the modified mesh is skinned and vertices rendered.



(a) $\bar{\mathbf{T}}, \mathcal{W}$ **(b)** $\bar{\mathbf{T}} + B_S(\vec{\beta}), J(\vec{\beta})$ **(c)** $T_P(\vec{\beta}, \vec{\theta}) = \bar{\mathbf{T}} + B_S(\vec{\beta}) + B_P(\vec{\theta})$ **(d)** $W(T_P(\vec{\beta}, \vec{\theta}), J(\vec{\beta}), \vec{\theta}, \mathcal{W})$

Figure 2-4: An unregularized latent space (left) vs. a regularized one (right). In this example, the latent spaces encode information about specific shapes, but sampling the space in between produces garbage. By regularizing the latent space, we ensure that any random value passed into the decoder results in a meaningful generation.



Chapter 3

Methodology

We seek to build an end-to-end, intuitive system for processing raw motion capture data into a unified body model through which generative models can be trained and conditioned motions synthesized. We will walk through each component of the pipeline individually. At a high level, we want to do the following:

1. auto-label the markers to their respective locations on the body,
2. extract SMPL parameters for each movement using the labeled markers (called 'solving the body'), and
3. pass a compressed representation of our SMPL parameters into a VAE which can learn a generative action-conditioned distribution.

We implement a modified VAE construction, known as a conditional VAE [18] – one which we can pass the desired action type into the decoder. More on this architecture can be found in Section 3.5. The overarching flow of the pipeline can be seen in Figure 3-1.

This chapter follows the full life cycle of our data, from the point of data capture all the way to the generation of synthetic motions. The chapter is organized to desribe the pipeline and also provide brief overviews of the processing modules themselves. The three major sections match the three key components of the pipeline enumerated above.

3.1 Data Collection

The story starts with collecting raw data. The Immersion Lab houses a complete OptiTrack [6] optical motion capture system, capable of high-accuracy passive motion capture at up to 240fps. Motion Capture data is recorded by applying passive markers according to a specific layout on the body, followed by calibration and recording. After the data is recorded, it can be replayed and even labeled via the OptiTrack Motive software [7]. Data is usually extensively labeled by hand, matching the marker to its label name frame-by-frame. Motive contains a built-in proprietary skeletal solver, however, the values of this solved skeleton are not allowed to be exported into other use cases.

3.2 Data Preprocessing

Our goal is to convert our motion capture marker data into SMPL parameters, to give us a unified, compressed representation of our motion capture data. It is from these SMPL parameters that further generative modeling is possible. In data collection, data is recorded in long 'takes', and the subject typically performs multiple actions, even different kinds of actions, in each such take. Thus, the only manual part of our pipeline involves the labeling of actions by start and end timestamp. Our pipeline is robust to minor inaccuracies in this labeling process.

The preprocessing step consists of modules to export our data from OptiTrack, split the takes into files per individual action, and otherwise process the data to be easily ingested by SOMA. We use the SOMA [2] architecture then automatically labels the markers of our raw motion capture clouds.

3.3 SOMA Auto-labeling

As stated, our data is collected with passive markers. In a passive system, each marker simply reflects infrared light so that a camera system can track its location in 3D. Thus, if a marker is occluded, and then re-appears, the system cannot intrinsically

know that it is the same marker. This raw data from a passive motion capture system is often called a mocap point cloud (MPC).

The SOMA model can take inputs of MPCs, which are naturally noisy and often occlusion-filled, and label the markers without calibration or any other input data. This problem can be accurately described by Figure 3-2. Even though our data is unlabeled, in many cases, a marker can be tracked over a local interval of time. These tracked intervals are known as tracklets, representing a consistent marker through frames. The smallest 'tracklet' for technical completeness is only one frame long. However, in the event of an occlusion or a tight movement that creates ambiguity, the tracklet ends. The next time that same marker can be tracked, a **new** tracklet is created for it, as it is now recognized as separate from the tracklet which ended earlier. Thus our input MPC is converted to a series of many unmatched, unlabeled tracklets. The only upper bound on the number of disjoint tracklets a marker may form is the total number of frames itself (one tracklet per frame). SOMA performs two main steps, first finding correspondence between tracklets, and then combining matched tracklets and providing a part-of-body label.

3.3.1 Multi-headed Self-attention

We have just described a difficult problem. From an unlabeled point cloud, made up of disjoint tracklets, how do we even find correspondences between the different tracklets attributed to each marker, much less know that a specific marker refers to the left tibia? The key to understanding this comes from SOMA's attention-based Transformer approach [19]. We provide a high-level overview below.

Self-attention allows individual components of the data to interact with each other to improve their "self-understanding". This technique was first implemented in language translation to allow words in a sentence to improve their own embeddings based on specific important words around them. For our situation, our 'words' are the individual markers, and the sentence is the body. Imagine the situation where some raw marker understands that its own location is **highly dependent on the locations** of a few markers, all of which are above the ground, and all located to the right of the

point cloud center. This marker has just effectively understood that it is somewhere on the right arm!

The technical implementation involves learning key, query, and value matrices which help each data point understand exactly which other markers relate to it the best. Over time, a set of key, query, and value matrices may become good at finding a specific type of dependency. Each set of key, query, and value matrices forms an attention head. To learn multiple inter-dependencies in the data, multiple heads are used in parallel, with each head initialized to random values to encourage diversification. In a Transformer, these attention heads are applied layer by layer, repeatedly onto the data. At the deeper layers, the essence of the inter dependencies eventually filters through.

Figure 3-3 shows the effects of attention from the first layer (top) to the deepest layer (bottom). Here, a random attention head is sampled over 50 frames of the input, and the weight averages are pooled. We see the head closely finds association with the spine, the hand with the shoulder and arm, and the toe with the leg. The square indicates the marker of interest whose attention graph is being depicted. From left to right, this marker is on the head, the right wrist, and the right toe.

After the self-attention re-weights the markers, an optimal transport layer enforces constraints on the body, as well as constraints between mapping the labels to the markers. This layer outputs the final marker assignments. This is made possible by using a pre-trained model which is trained on a 'superset' of 89 potential marker locations. This superset is able to accurately label markers if they are placed in one of the 89 spots that it understands. The full extent of the training details, as well as a description of the optimal transport layer, can be found in the SOMA paper [2]

Currently, the Immersion Lab hand labels the markers for much of the collected motion capture data. This easy-to-use SOMA integration solves a large pain point, as labeled data lends itself to many kinds of kinematic analyses beyond simply the use case of our pipeline. In cases where the collected data is already labeled, we expose an easy-to-use interface that can be called to skip SOMA and process the existing label data into the correct formats as well. See the branch in Figure 3-1.

While marker-labeling is very powerful by itself, our goal lies ultimately in body-solving: extracting the SMPL parameters of shape β and pose θ for each action. For this, we use MoSh++ [11], a robust SMPL skeletal solver for labeled mocap data.

3.4 Skeletal Solving

MoSh++ [11] extends the previous MoSh model [10] to output solve bodies for SMPL shape parameters β and pose parameters θ. At a high level, this is done in a two-step, loss-minimizing way. We want to minimize vertex-to-vertex loss between our given markers and the parameterized model we are hoping to fit. For given labeled N markers and locations, with some guess of SMPL parameters θ and β, the pre-trained SMPL model is fed the parameters to generate the guess mesh. MoSh++ then simulates the same N markers on this mesh at their known locations. All 89 markers from the 'superset' SOMA model are known and understood by MoSh++. The loss is calculated as a function of the distance between these simulated markers on our guessed SMPL body with the actual marker locations in the labeled motion capture.

In the first step, we solve for β values, which represent the subject's proportions, height, hips, limbs, etc. This is done by taking 12 random frames from the motion capture data per subject, and optimizing each frame with its own θ (pose parameters) and a shared β. This process includes many improvements to prevent local minima and ensure that an optimal β which represents the subject's true shape parameters is learned.

In the second step, the β values are held constant, and the model iterates through the capture, minimizing the loss due to θ. This optimization includes a pose prior, which acts as a grounding to ensure the resulting pose values are well-formed and constrained to realistic motion, as well as a temporal smoothness factor to reduce jitter between poses, ultimately acting as a continuous pose prior through time.

Figure 3-4 shows a resulting frame from an unlabeled dataset processed with SOMA and MoSh++. The markers represent the raw data, and the mesh is the

SMPL-parameterized fit.

3.5 Generative Modeling

With our action classes and SMPL parameterization, we are now ready for generative modeling. We follow the architecture described in ACTOR [14] by Petrovich et. al., who train an action-conditioned Transformer VAE for human motion synthesis. We introduced the VAE architecture in Section 2.2. This architecture combines the VAE loss objective with the Transformer architecture, with the Transformer being used as both the encoder and decoder. The Transformer is essentially made up of many layers of self-attention, which we describe in Section 3.3.1. The downside of self-attention is that the attention heads have no knowledge of the order of input sequences – each embedded token (in our case, frame) is compared to other tokens, completely naive to the token's position. While this worked well in SOMA, as we were attending over all of our tracklets, which did not have any order, here, we are dealing with a sequential motion. Order does matter. ACTOR uses the solution defined in the Transformer paper [19] points to use a sinusoidal positional embedding added to each token's embedding (using addition so as to be differentiable) to indicate a position in the sequence.

As described in ACTOR's architecture, we add two `[class]` tokens at the start of our embedding into our encoder, one for μ, and one for σ, containing the means and standard deviations for our multi-Gaussian latent space, respectively. These `[class]` tokens follow the idea in BERT [1], allowing the attention heads to learn the desired values of μ and σ, and this contrasts the vanilla VAE we described in Section 2.2. In the basic VAE, the last layer of the encoder is a fully connected layer to outputs, which essentially averages or pools the layer into μ and σ values. The two `[class]` tokens operate in the same way as an aggregator of the encoder. Notably, these class tokens encourage the use of attention in the pooling process itself, as the μ and σ values are now attended to by the encoder architecture, rather than simply being an average over attention heads. Beyond this, the final modification involves

the incorporation of the action label token in both the encoder and decoder to learn a conditioned VAE [18]. To do this, the model learns a bias term for each action label which is incorporated upon decoding to essentially shift the location of latent sampling into the specific distribution space that is learned for the given action label.

Thus, for synthesis, the decoder acts as a generator, taking input of time information T, the number of generated frames desired, and an action label. Figure 3-5 shows the full VAE architecture described in the ACTOR paper.

Our pipeline postprocesses the MoSh++ outputs to prepare the data for training in the VAE model. Following guidance from ACTOR, this includes pose alignment, file conversion, downsampling and serialization.

Figure 3-1: Our end-to-end pipeline, starting from the data collection, and ending with synthesized motions.

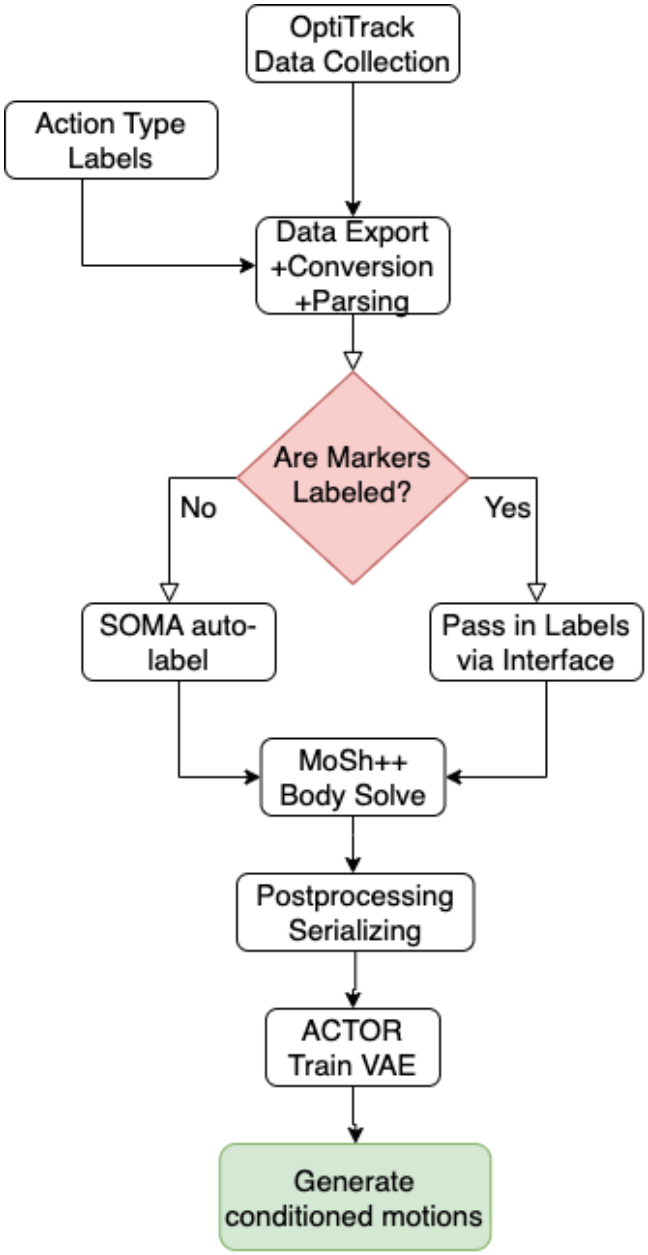


Figure 3-2: Showing the two key steps of SOMA. The inputted data is formed of many disjoint tracklets, visualized in the top section in grey. The x-axis represents time, and the y-axis represents tracklets of markers. In the first step, the matching tracklets are identified. This is visualized in the middle section, where matching colors are applied to different tracklets which represent the same marker. Then, these matching tracklets are combined robustly to handle occlusions. Gaps in the tracklet data are representative of occlusions (no data recorded). The final step involves assigning a label name to each marker for the part of the body (not pictured). Figure from the SOMA paper [2].

Figure 3-3: The red square highlights the marker whose attention weights are shown. The background meshes show the first attention layer, and we can see the deepest layer in the foreground. The attention is much more sparse at the deepest layer, as the key essence of the inter dependencies have been understood. Figure from the SOMA paper [2].

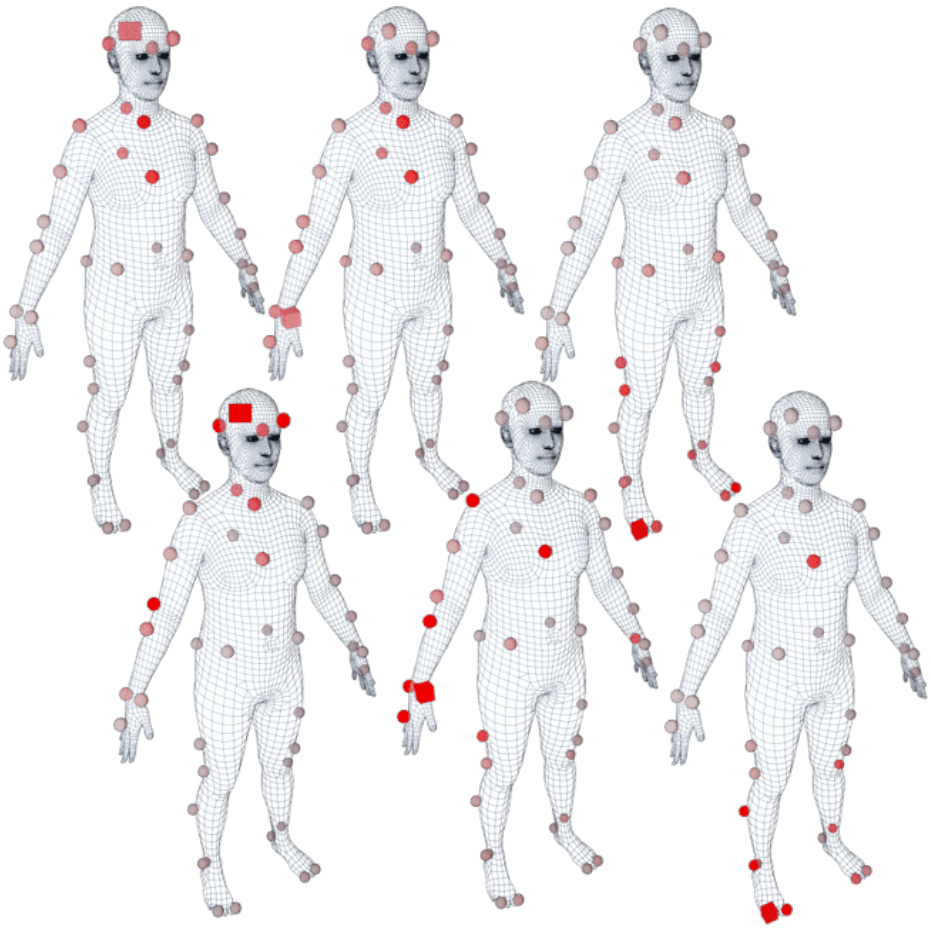


Figure 3-4: Visualized SMPL mesh from a raw motion capture cloud processed with SOMA and the MoSh++. Figure from the SOMA paper [2].

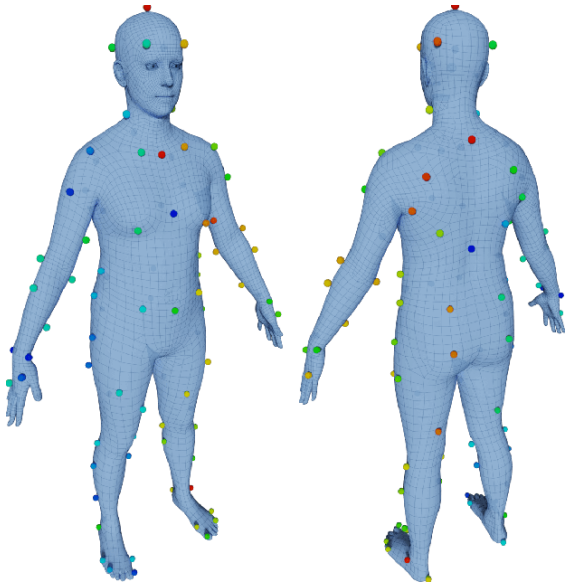
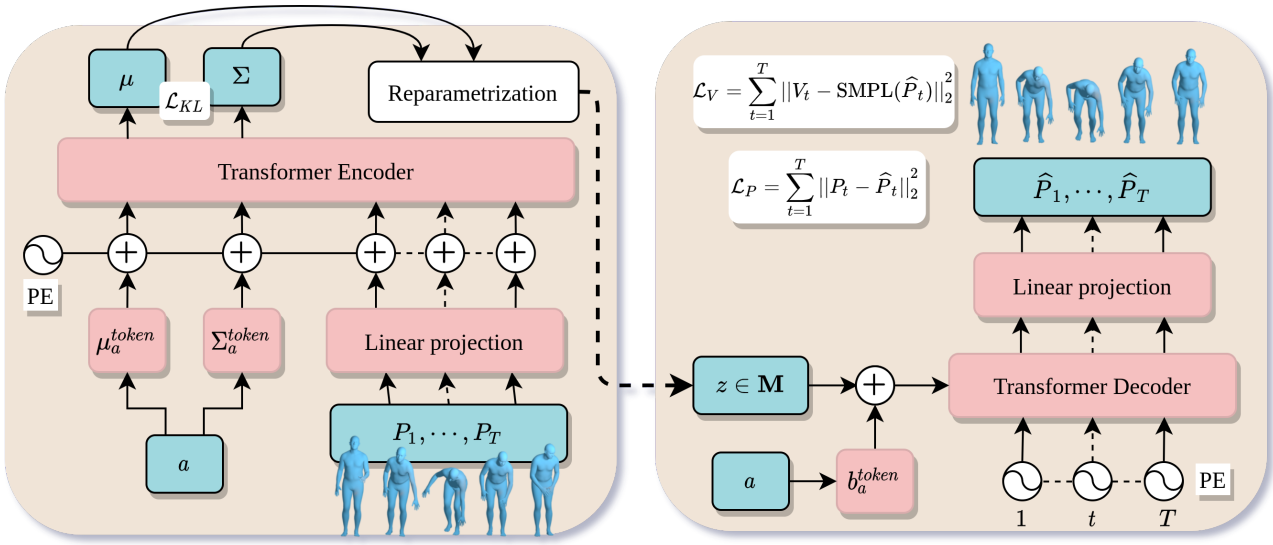


Figure 3-5: The action-conditioned Transformer VAE architecture as described in [14]. Note the `[class]` tokens which are initialized from the sequence's action label.



Chapter 4

Dataset

With our pipeline explained, we will now provide an overview of the data that we will pass through this pipeline and train generative models on. While our framework is robust to all kinds of sports and action data, our motivation comes from the goal of building a system to improve fencing pedagogy. As such, we use a dataset collected in the Immersion Lab, whose collection was led by Enya Ryu [17]. We assisted in the collection and analysis of this data as needed. This dataset contains 30 subjects, who are all explicitly beginners to fencing, with the goal of analyzing the effects of various XR technologies on motor learning. As such, the subjects are subjected to various learning modalities, with the task of mimicking the movements of a rendered avatar. The modalities include full virtual reality, augmented reality, 2D video, 360-degree video, and memory (no avatar). These modalities were created using the Varjo XR3 [13] headset, which the subjects wore throughout the experiment as they replicated the fencing actions. Thus, the data collected is from fencing beginners who are performing actions while wearing a VR headset and tracked with full-body markers. Figure 4-1 shows screenshots from four of the five modalities, with memory not pictured as it contains no demonstrator. Again, the modalities are randomized per subject to lower confounding variables.

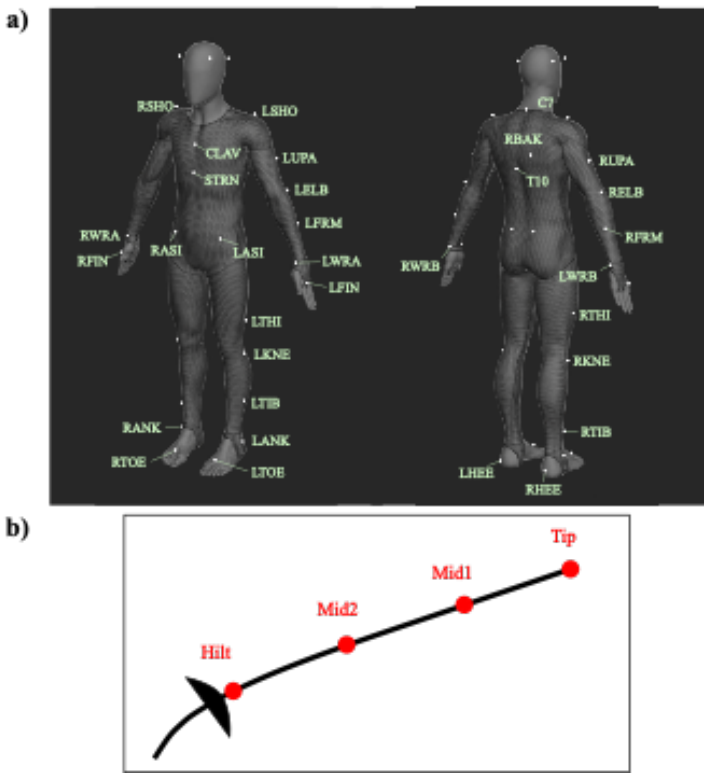
Figure 4-2 depicts a visual of the marker placements, with markers added to the fencing sword as well. It is important to note that our body modeling framework only models the body, and not the sword data directly. Thus, the sword data is not

Figure 4-1: Each of the modalities presented to the subjects in the data collection process. The fifth modality, memory has no demonstration. From [17].

passed into the generative architecture. It is possible to infer the sword position and orientation relative to the hand and wrist positions and rotations of the weapon arm.

Though the dataset collects data from beginners, it contains a rich set of thousands of actions in multiple classes for us to train our generative models. One inhibitor to our VAE successfully learning representations is the high variance of the nature of the data, as beginners all have varying posture, form, and flexibility. However, this can also be a strength, as the nature of beginners has high variance, and thus it can be argued to model it as such. This high variance contrasts with a potential dataset exclusively of experts. Although every fencer does have a unique style, it is fair to assume that the data of experts would have less variance, although this assumption is irrelevant to our process. A visualization of the dataset collected is shown in Figure 4-3. What we can see here is a relatively good clustering of the tip data, indicating some bound on the variance of the movement, which is a good sign that our data is well-positioned to train our generative model. The figure plots

Figure 4-2: The locations of the markers, on body and on the fencing sword. From [17].

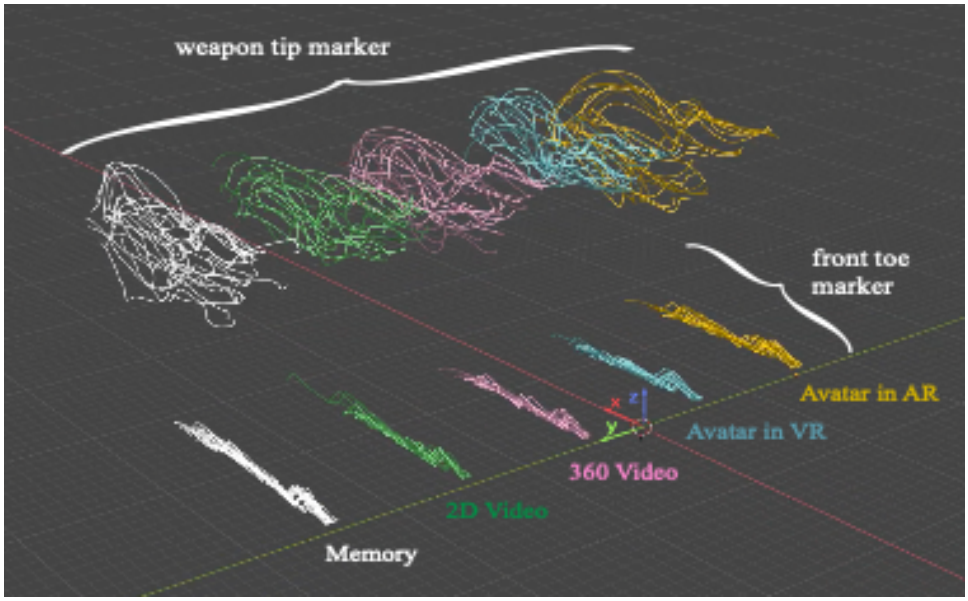


every collected action, plotting two traces – the movement of the front toe, and the weapon tip. The action data has been grouped by the modality it was collected in to highlight potential differences in the data based on modality, similar to the preliminary experiment referenced in Section 1.2.

4.1 Data Summary

Let us better understand the different action labels in our dataset. To summarize, this data contains 30 subjects, all beginners to fencing, observed under five modalities, recording a total of 4703 individual actions. We considered 5 different actions: the advance, the jump, the lunge, the combined advance-jump-lunge, and the special 'elements together' action. An advance represents a step forward, a jump is a two-footed jump while maintaining the hand position, and the lunge involves reaching the front foot forward with an open hip stance, and extending the weapon arm. The

Figure 4-3: A visualization of the data collected. Each color represents a different modality, with the lower lines representing the front toe, and the upper ones representing the sword tip. Principal component analysis was used on the front-toe marker to determine the major axis of movement, and the rotations from PCA were applied to tightly overlay all action data for accurate analysis. The modality labels are shown in the figure. Figure from [17].



advance-jump-lunge is a combination attack, consisting of an advance that moves fluidly into a jump, and then right into a lunge. In this combination attack, the point at which a single component ends and the next starts is indistinguishable. The special 'elements together' block consists of an advance, jump, then lunge – all done in succession, but includes a brief pause in between each element, The data is represented below:

Label	Count
Advance	591
Jump	672
Lunge	671
Advance-Jump-Lunge	2638
Elements Together	131
Total	4703

Our data appears to be skewed in favor of the advance-jump-lunge. However,

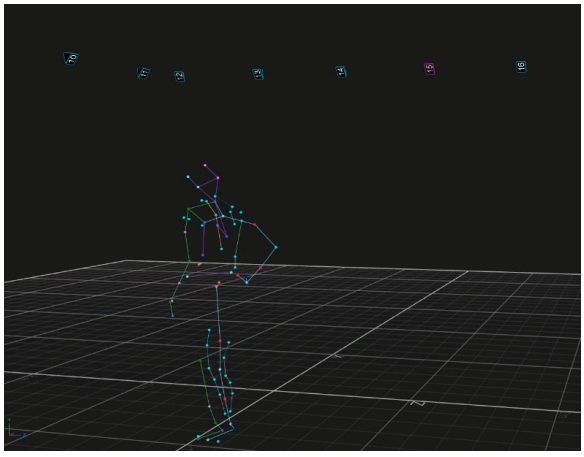
we still have many samples from the other action classes for the model to learn a meaningful representation. In the ACTOR paper [14], some models were trained with as few as 30 actions per class. However, more important than the amount of data is the data consistency. Our data naturally contains high variance due to the nature of the subjects.

Chapter 5

Results

In this chapter, we will visualize some of the key results from the SOMA [2] process, mainly focusing on the results from using the dataset in Chapter 4 with our aforementioned pipeline.

Figure 5-1: A frame of recorded raw motion capture data, rendered in OptiTrack Motive software [7].



Recall that the raw data collected by the optical tracking system consists of the (x, y, z) locations of each of these marker dots recorded frame-by-frame, visualized in Figure 5-1. No modeling engine can effectively use this data as is. We took this data and processed it iteratively until it contained the essence of human motion, videos for which exist, but are not included in this thesis.

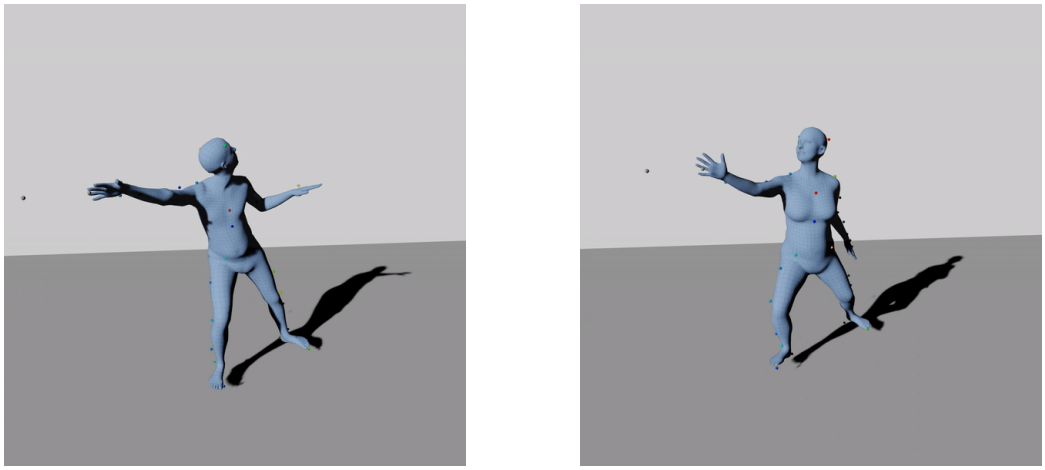


Figure 5-2: Left: Swapped head marker labels from SOMA occasionally create an inaccurate mesh. Right: a manually labeled mesh yields the correct body movement.

5.1 Auto-labeling

Auto-labeling does not always work. Where in most cases, the SOMA auto-labeler performed in line with the hand-labeled process, we highlight the following case where the two key head markers were labeled inversely, creating an inaccurate yet humanly possible mesh. Figure 5-2 shows this side-by-side comparison. The fencer on the left should be looking at their extended right arm, not the other way. Incorrectly swapped left and right head marker labels cause a flip. After the skeletons were solved, we inspected the rendered meshes to verify general correctness, manually editing marker labels in the rare cases where it was needed. (In situations where SOMA fails to find a marker labeling with high confidence, it self-reports an error. However, we did not experience this.)

5.2 Motion Generation

The final results are visualized in Figures 5-3 and 5-4. These are synthetic digital renderings conforming to human body limitations and extensible to perform other actions in the future. They preserve the larger essence of the sporting action. These figures represent the major accomplishment of the work that went into this thesis. The figures for the remaining three action classes are included in Appendix A.

In Figure 5-3, we visually represent motion outputs from the generative model, splitting a video into important frames. *Time progresses downwards row-by-row in the figure.* This figure contains five columns: in the first two on the left, we have a sampled real ground-truth action, followed by a reconstruction of that action sample. A reconstruction represents the output of the VAE when the training sample is passed through. The final three columns visualize a single action-conditioned generation from the model from three different camera viewpoints. *The limbs are color-coded as follows: the right arm is green, the left arm is blue, the right leg is red, and the left leg is magenta.* The top-most row shows the legs wide, at the start of a step. In the middle rows, the back leg approaches the front leg, completing the advance, and the final row shows a widening stance with an extending arm, indicating the start of a lunge.

Figure 5-4 depicts a generated lunge action. We see the lunge begin decisively right from the beginning of the motion, quickly reaching a wide stance and an extended arm. We see that the real lunge in the graphic depicts a right-handed lunge, whereas the generated lunge is left-handed. In training our model, we elected to not split right and left-handed subjects into different label classes, mainly due to an overwhelming majority of our data being right-handed, which would only imbalance our dataset labels even further.

Finally, our generative model predicts SMPL parameters θ, which can be used to skin a realistic mesh. In Figure 5-5, we show a top-down view of a generated mesh performing a lunge. Note that the β values here are set to their defaults, and they can be changed to generate the same action in a variety of different body types.

Successful generation of realistic motion outputs represents a working pipeline. Generative models are difficult by nature to evaluate since there is no simple error to be calculated as exists in classification. A precedent in generative motion introduced by Guo et al. [3] considers four key factors: the Frechet inception distance (FID), recognition accuracy, diversity, and multimodality. The FID effectively compares the distributions of *features* extracted from the generated actions with those from the ground truth or test data, with less difference between the two being better. These

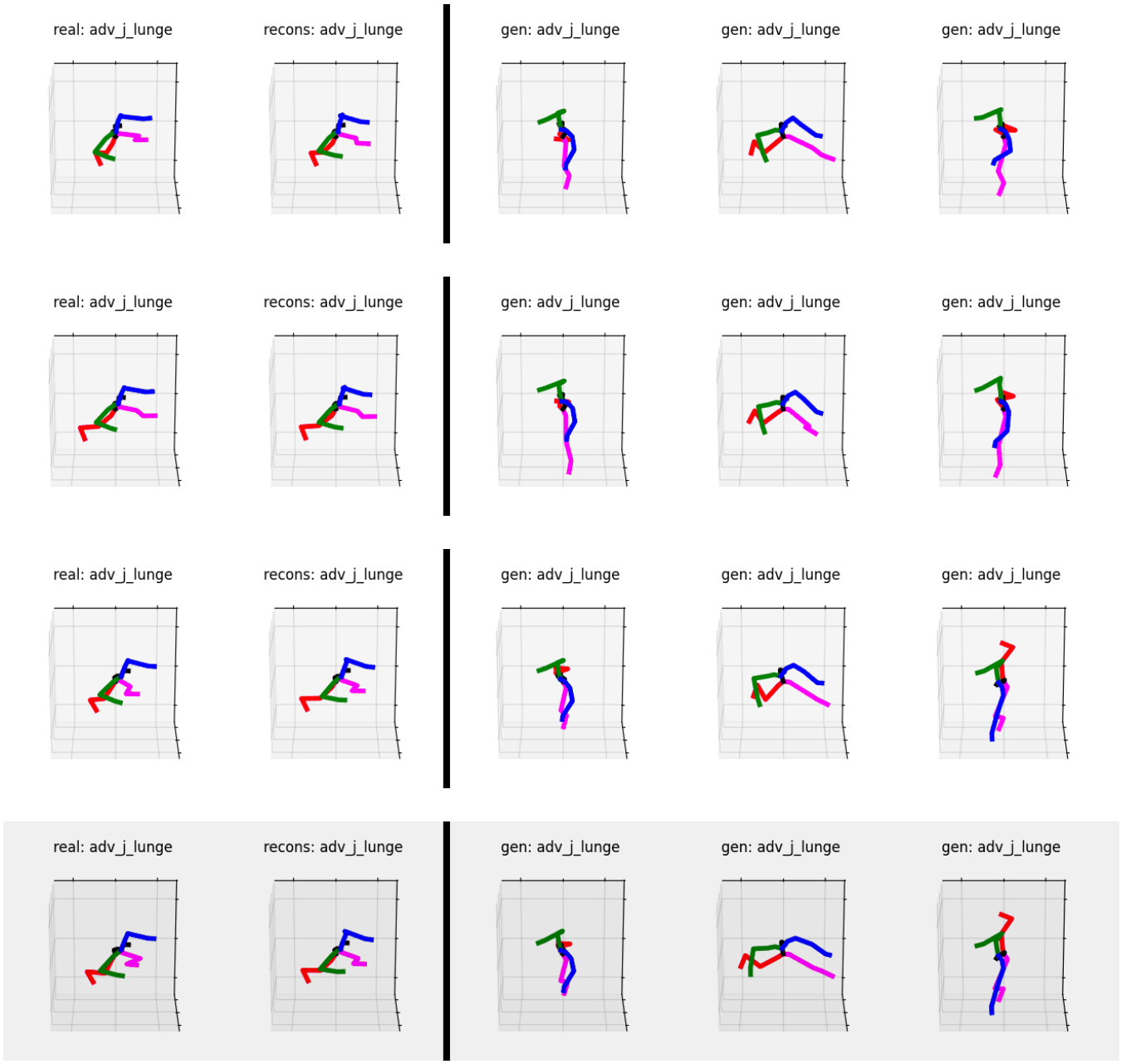


Figure 5-3: Generated motions for the advance-jump-lunge action. *Time progresses downwards row-by-row in the figure.*

features represent some high-level features of our data, and this requires a *separately trained* recognition model from which features can be extracted. For accuracy, again, a separately trained classification model (often the same model as for FID) is used to guess action categories for the generated actions. Diversity and multimodality ensure that the generated actions contain variance over all action categories as well as within an action category. Our true evaluation lies in the success of our downstream system to quantifiably improve trainee abilities – and this is not yet possible to test. We leave the design of a recognition model to quantify the FID and accuracy of the generated outputs as future work.

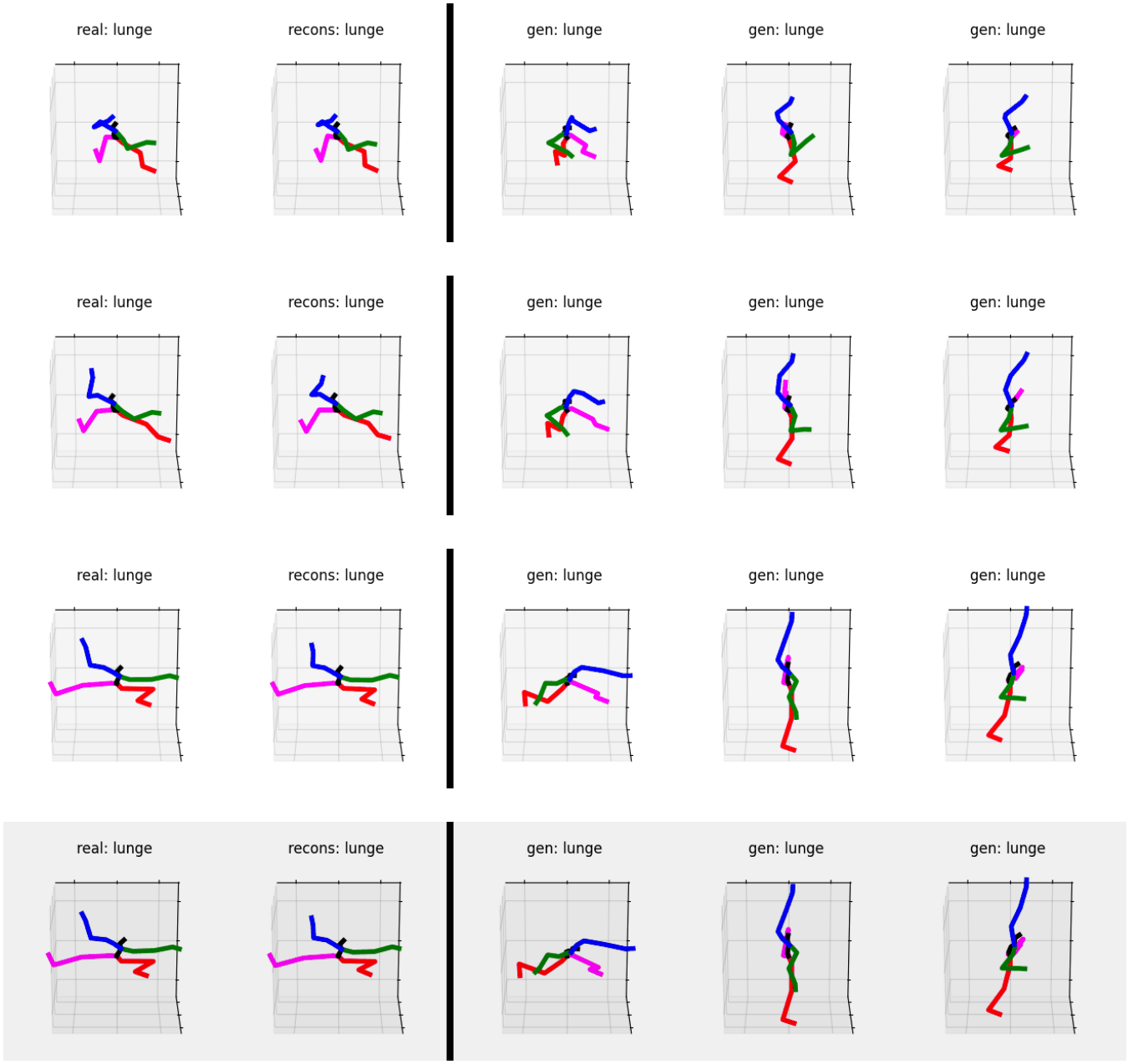


Figure 5-4: Generated motions for the lunge action. *Time progresses downwards row-by-row in the figure.*

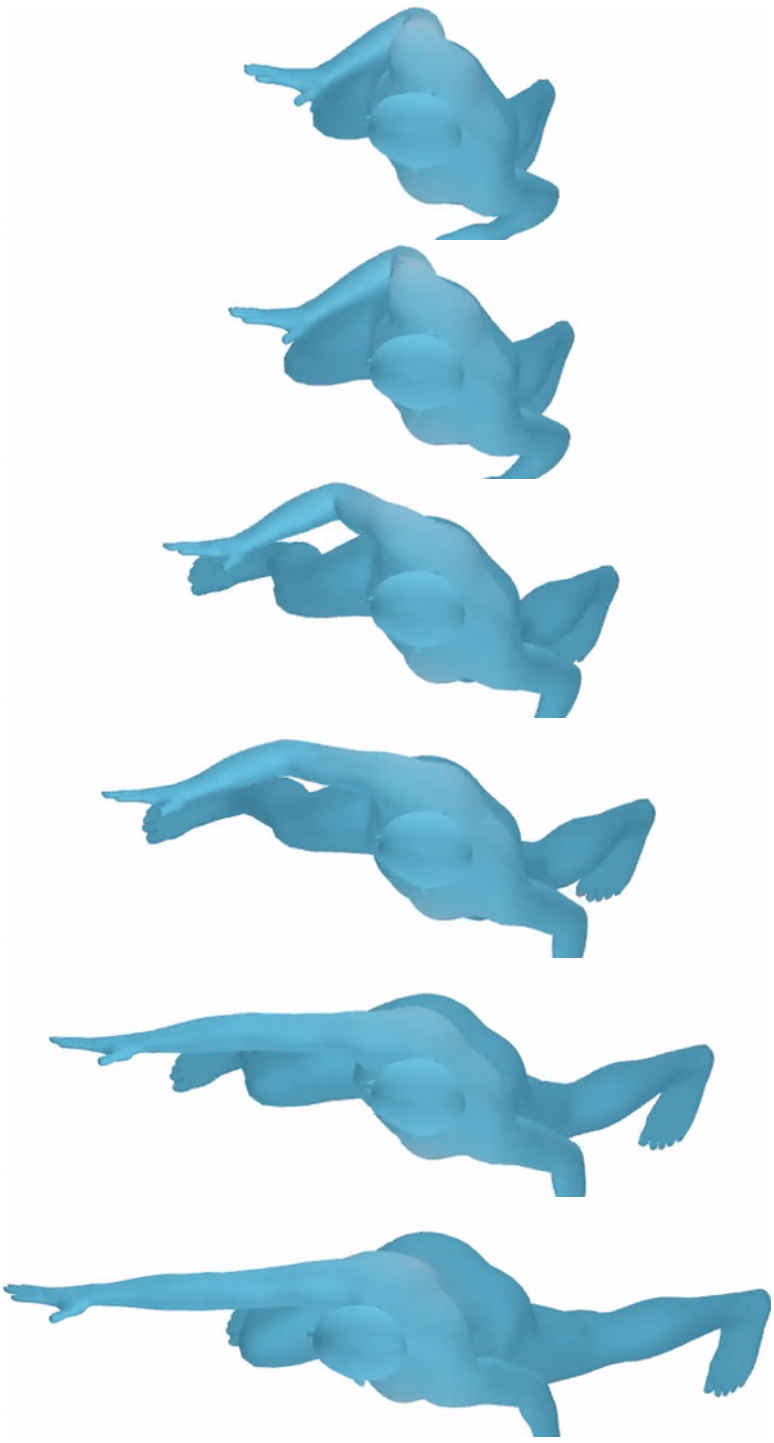


Figure 5-5: Generated motions for a lunge from bird's eye view, skinned with SMPL. *Time progresses downwards row-by-row in the figure.*

Chapter 6

Conclusion

6.1 Summary of Contributions

1. Gained an understanding of generative modeling technology and its data requirements.

2. Gathered raw data from motion capture and understood its limitations of how a generative model could not use it.

3. Used existing frameworks to process this data and converted it in a three-step process to a format that preserved its larger essence while standardizing its usability.

4. Wrote code for the generative model to iteratively import this data and debug until a clean, usable workflow emerged.

5. Created tools for visual rendering such that it can be used by a coach to show various actions with appropriate modifications as needed.

This thesis illustrates the creation of a working, end-to-end pipeline that takes motion capture data and synthesizes conditioned motion data from it. It enables downstream systems for improved athlete learning with realistic, customized generations. The implemented pipeline combines many different processes under a unified

hood, including the OptiTrack [6] hardware used to collect the raw data, the internal existing Immersion Lab tools, the various architectures for labeling and solving the motion capture, namely SOMA and MoSh++, and the generative modeling architecture. Importantly, the system is highly intuitive and well documented, which is the result of iterative testing in the Immersion Lab with other members to understand the pain points and fix bugs in the process.

The integration of these many different systems together was no small feat. Whereas the architectures used in the workflow all had open-source published implementations, every single one required the construction of a custom fork to fix bugs, incorporate performance enhancements, and incorporate interfacing to work with our desired data types and structures. All of these customized forks were then combined into a library which was integrated with the Immersion Lab codebase for seamless use with existing workflows.

6.2 Future Work

The most prominent future work would be the use of our modeling – either with the dataset we used as a proof-of-concept for this thesis, or another dataset – to create a virtual training system for fencing. This would require rendering the outputted generated SMPL mesh in a virtual environment, as well as incorporating the customizability of our generative model. Additionally, this would require the rendering of the sword in the virtual environment. With subject experimentation, we could analyze the efficacy of this pedagogical technique, and compare the results to the non-customized generic experience.

Additionally, we would like to quantitatively analyze the generated motions to have a numerical sense of success beyond just a minimization of the VAE loss. This would require training a recognition model to evaluate the generated actions and predict a label. Thus, an effective generative model could be described as one which generates actions that the recognition model can identify with high accuracy.

This work progresses towards the full, deep integration of human movement in

sports and technology. An action-conditioned generative model could be extended from single actions to generate full suites of competition-like movements on request, and even become a live competing partner. Additionally, effective action recognition models can be leveraged to provide immersed, real-time trainee feedback, helping enact live corrections to the trainee's form, posture, and movement. The status quo, in which trainees copy the movements of experienced athletes before the requisite abilities and skills have developed, can increase (1) the predisposition to injury, (2) the development of poor technique, and (3) an overall delay in the learning process due to learned bad habits. Our work makes meaningful progress in developing a system which addresses all three concerns.

Finally, where we leveraged a motion capture system for high-accuracy movement data, high-accuracy marker-less tracking techniques can be used at comparable accuracies, lowering the barriers in data collection and making such training systems widely accessible. Such systems have transformative applications in the world of movement and sport.

Appendix A

Supplementary Results

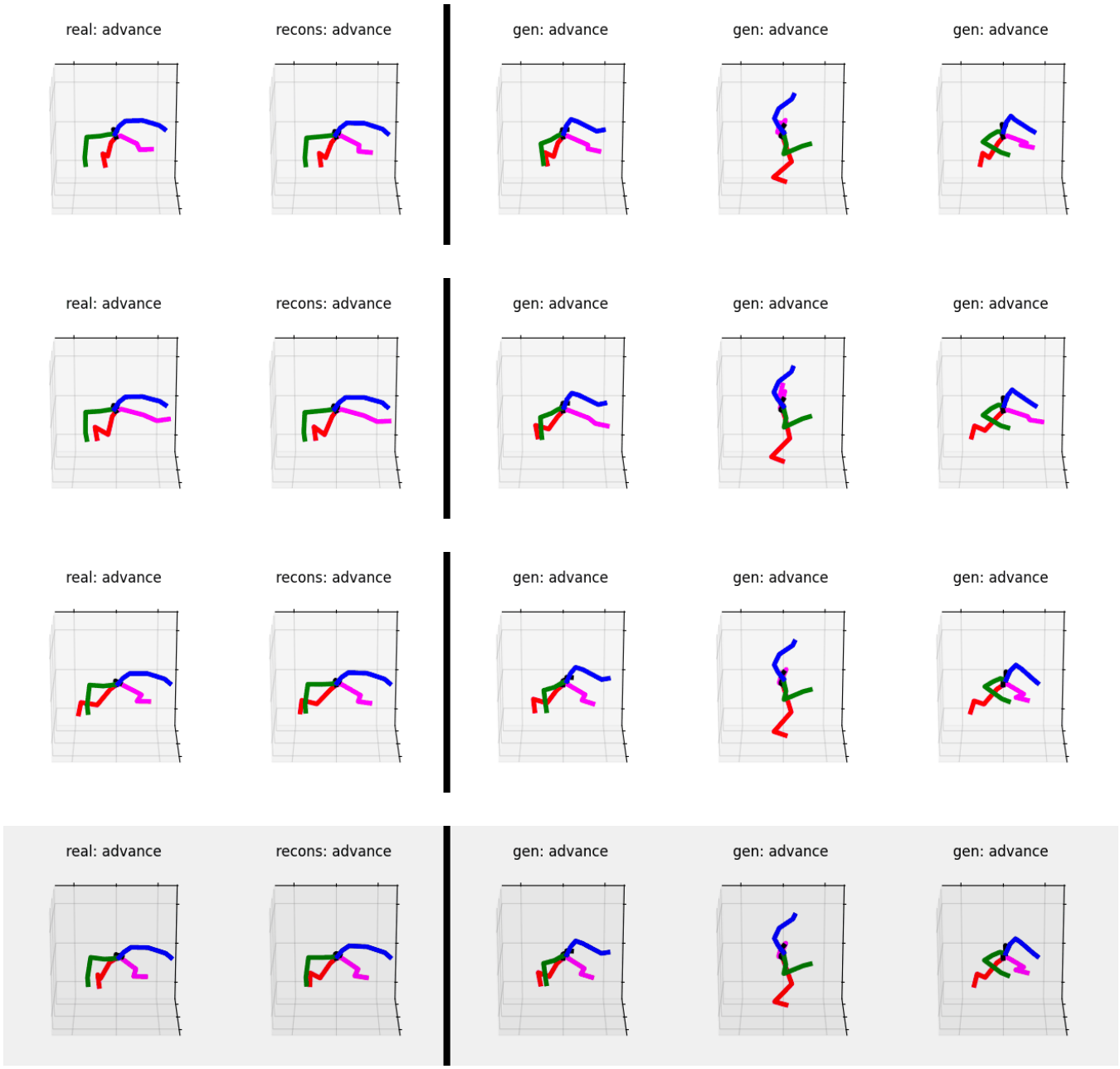


Figure A-1: Generated motions for the advance action. *Time progresses downwards row-by-row in the figure.*

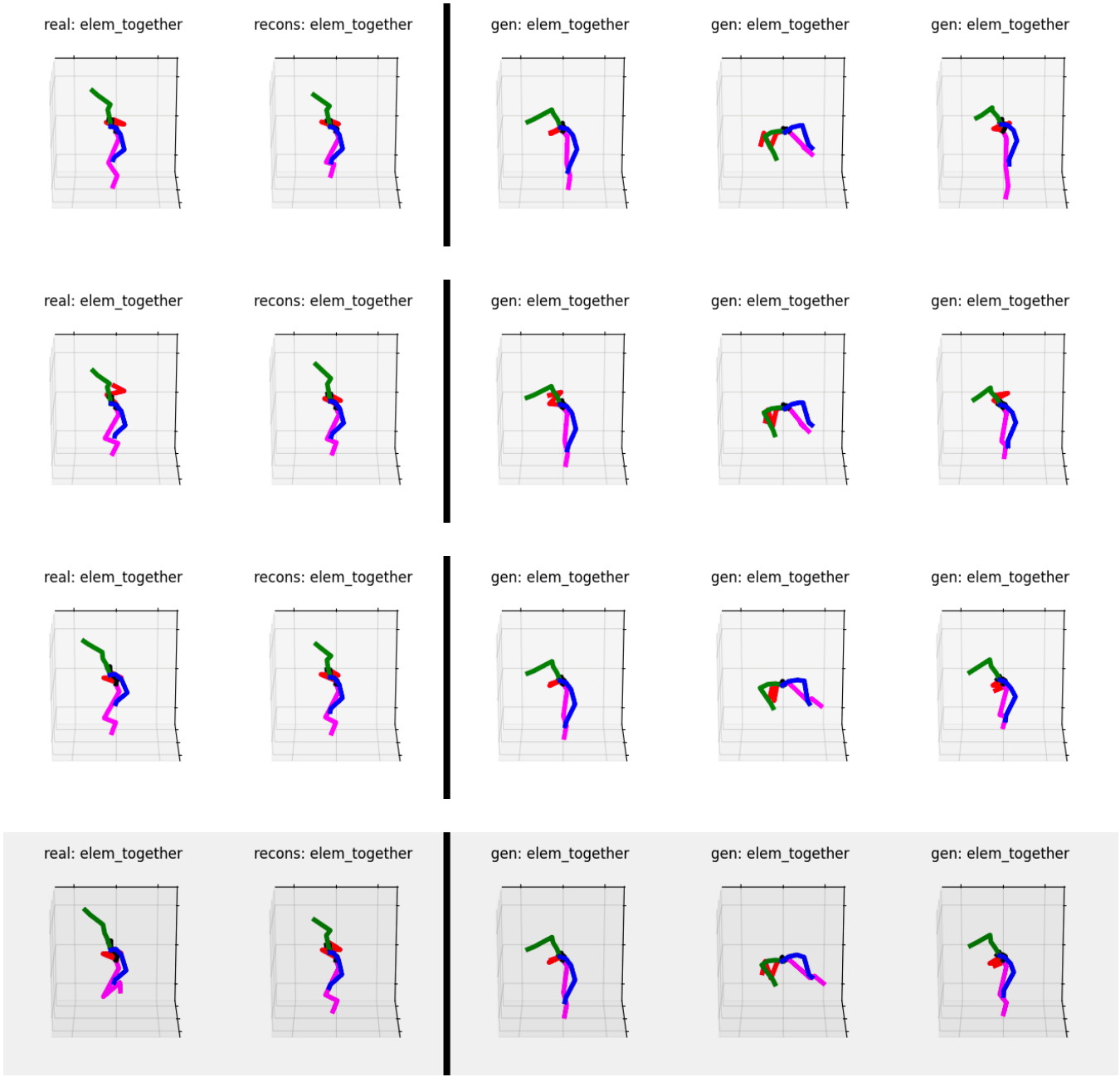


Figure A-2: Generated motions for the 'elements together' action. *Time progresses downwards row-by-row in the figure.*

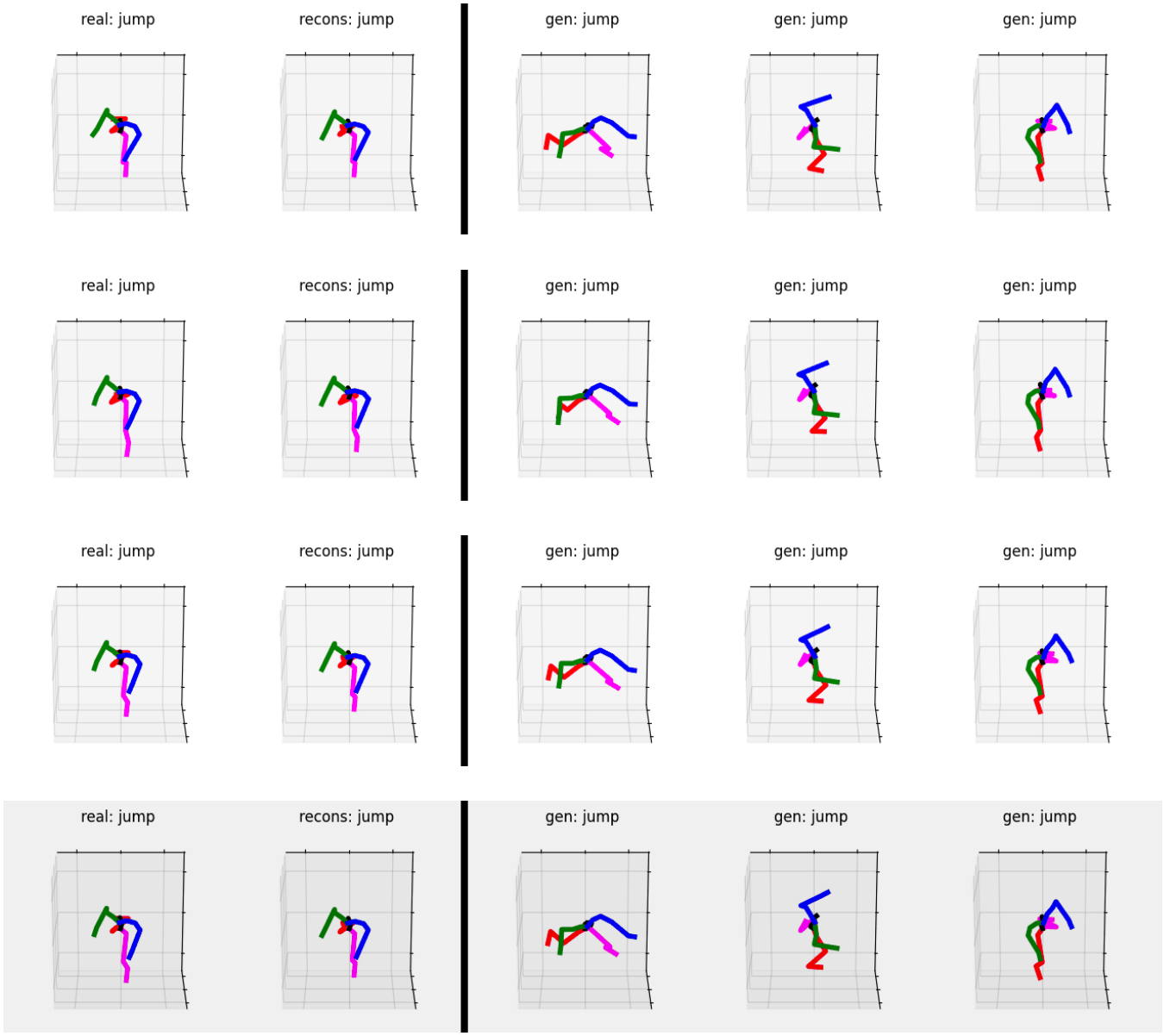


Figure A-3: Generated motions for the jump action. *Time progresses downwards row-by-row in the figure.*

Bibliography

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Nima Ghorbani and Michael J. Black. SOMA: Solving optical marker-based mocap automatically. In *Proc. International Conference on Computer Vision (ICCV)*, pages 11117–11126, October 2021.

[3] Chuan Guo, Xinxin Zuo, Sen Wang, Shihao Zou, Qingyao Sun, Annan Deng, Minglun Gong, and Li Cheng. Action2motion: Conditioned generation of 3d human motions. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 2021–2029, 2020.

[4] Pengpeng Hu, Edmond Ho, and Adrian Munteanu. 3DBodyNet: Fast reconstruction of 3d animatable human body shape from a single commodity depth camera. *IEEE Transactions on Multimedia*, PP:1–1, 04 2021.

[5] Robert Hupp. Advancing fencing pedagogy through xr technology. Research Proposal by Robert Hupp for MIT.nano, January 2022.

[6] NaturalPoint Inc. Motion Capture Systems — optitrack.com. `https://optitrack.com`. [Accessed 16-08-2023].

[7] NaturalPoint. Inc. Motive — optical motion capture software. `https://optitrack.com/software/motive/`. [Accessed 16-08-2023].

[8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[9] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. Smpl. *ACM Transactions on Graphics*, 34(6):1–16, 2015.

[10] Matthew M. Loper, Naureen Mahmood, and Michael J. Black. MoSh: Motion and shape capture from sparse markers. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 33(6):220:1–220:13, November 2014.

[11] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. Amass: Archive of motion capture as surface shapes. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019.

[12] Alexander Mathis, Pranav Mamidanna, Kevin M Cury, Taiga Abe, Venkatesh N Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature neuroscience*, 21(9):1281–1289, 2018.

[13] Varjo Technologies Oy. Varjo XR-3 - The industry's highest resolution mixed reality headset | Varjo — varjo.com. `https://varjo.com/products/xr-3/`. [Accessed 16-08-2023].

[14] Mathis Petrovich, Michael J Black, and Gül Varol. Action-conditioned 3d human motion synthesis with transformer vae. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10985–10995, 2021.

[15] Zion Market Research. 3d motion capture system market size, share, industry growth.

[16] Joseph Rocca. Understanding Variational Autoencoders (VAEs) — towardsdatascience.com. `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`. [Accessed 18-08-2023].

[17] Enya Ryu. Effects of XR Technology on Motor Learning in Fencing. Bachelor's Thesis, Massachusetts Institute of Technology, 2023.

[18] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[20] Adam Wojciechowski and Piotr Napieralski. *Computer Game Innovations*. Lodz University of Technology, 12 2016.