An AI-based service for motion data extraction

Developer: Zhang Zichuan,
School of Software Engineering,
Tongji University

Duration: 2021.04-2021.06

Contents

1.Indroduction to Project

1.1 Background

3D character animation is a critical part of video games, visual special effects, and digital art creation, which enables virtual characters to perform realistic movements and expressions. A large proportion of the workload involved in the creation of 3D animation is relevant to character animation, and motion animation creation is the central part of that. At present, there are mainly two ways to produce motion animation. The first one is to utilize professional editing software, through which the animator manually edits the skeleton. However, this is highly dependent on human experience, and with high labor costs. Another choice is to capture the motion of a real actor using sensing equipment. Although it can provide high-precision motion data so as to create high-naturalness character animation, it relies on expensive motion capture equipment and specific sites, which would be unaffordable for many teams. If the production of motion animation could be automated, it would greatly improve the practitioners' working efficiency and reduce the cost.

This project aims to explore the possibility of automatizing the production of motion animation based on artificial intelligence technology. To do that, a primary prototype system was developed which could automatically extract motion capture data from an input video and create a BVH file, and made some attempts to improve the performance of related algorithms. Codes for this project are available at my github repository : https://github.com/a389071432/VideoMotionCapture.

1.2 Technical Solutions

The overall workflow of the system is shown in Figure 1.1, which contains four steps: human body detection, 2D pose estimation, 3D pose estimation, and BVH file generation.

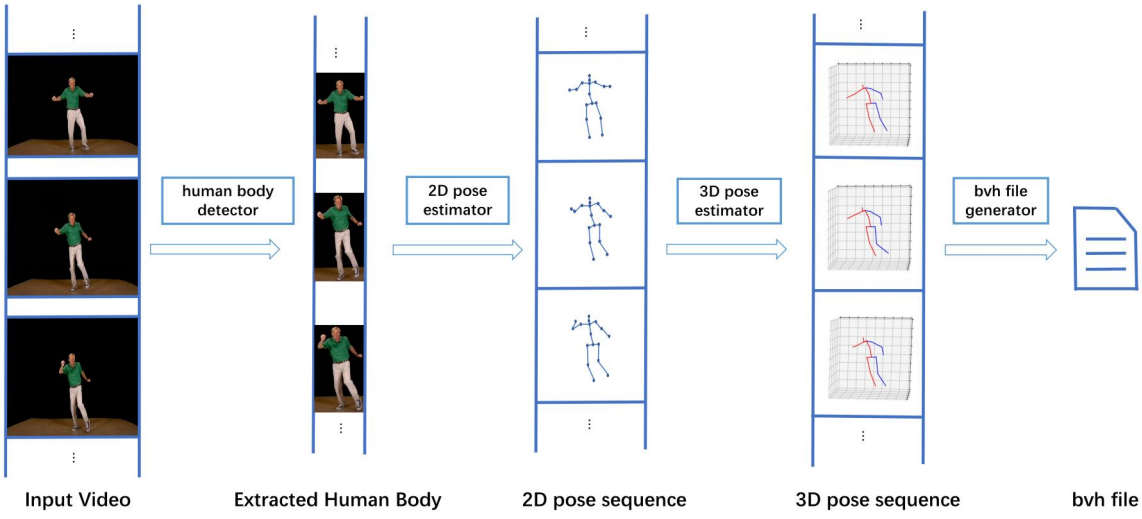

Figure 1.1: Workflow of the project's technical solution

The first three steps are done by three distinct neural network models, respectively, Yolov3-spp, FastPose and VideoPose3D. Each of the three models is described in Table 1.1.

Model	Input	Output	Description
Yolov3-spp	RGB image sequence, a tensor of size [batch, 3, 608, 608]	detection results for each image region, a tensor of size [batch, 22743, 85] , containing the confidence level of the target object, the object bounding box and the object class	detect target object from the original image, post-processing is required to filter out the human body
FastPose	image sequences containing only the human body, a tensor of size [batch, 3, 320, 256]	2D human pose sequence, each pose contains the 2D coordinates of 33 body joints in the image, a tensor of size [batch, 33, 2]	extract 2D poses from human body sequence
VideoPose3D	2D human pose sequence, a tensor of size [batch, T, 17, 2], where T is the length of sequence	3D human pose sequence, each pose contains the 3D coordinates of 17 body joints in the image, a tensor of size [batch, 17, 3]	estimate 3D human pose sequence from 2D human pose sequence

Table 1.1: Description of the three neural network models used in the system

Both Yolov3-spp and FastPose are included in the AlphaPose[1] research, and VideoPose3D is a model from a research[2] by Facebook. Please note that this project paid attention mainly to the 3D pose estimator as the methods for human body detection and 2D pose estimation have achieved excellent performance. For the detailed structure of the 2D pose estimator used in the project, please refer to reference[1]. For the 3D pose estimator, detailed description is given in section 3.2.

1.3 System Architecture

The system is developed and deployed with separate front-end and back-end, and the overall structure is shown in Figure 1.2 and Figure 1.3.

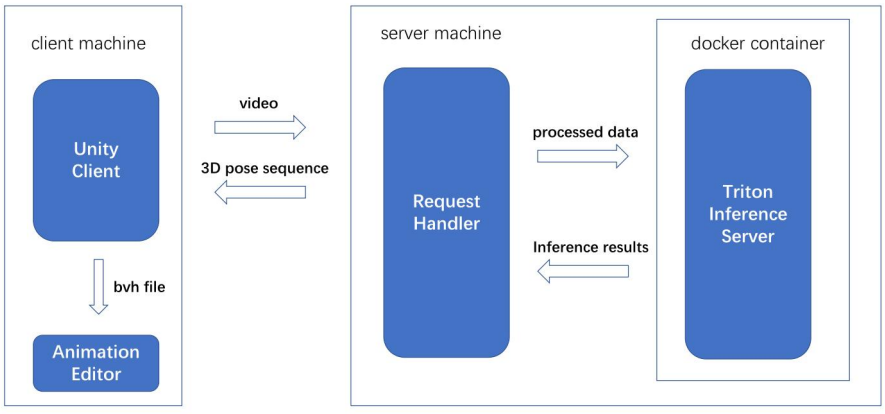

Figure 1.2: System structure overview, the generated BVH file could be directly used in animation editing tools.

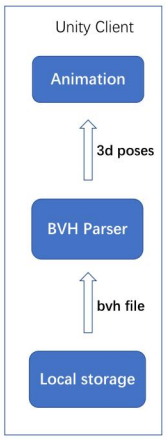

Figure 1.3: Workflow of action visualization, all on the client side.

Unity client works as the front-end of the system, which provides basic interactions such as file submission, visualization of pose data, and generation or parsing of BVH files. Triton Inference Server runs as the back-end, which is responsible for inferences using the deployed neural network models and returning inference results. Request handler is the intermediate module connecting the clients and the Triton server. Detailed description of each part is given in Section 5.

Due to the financial constraints of the project, I didn't rent a large-scale GPU server to deploy sufficient neural network models simultaneously, so the system's performance was not tested in a high-concurrency scenario. However, the designed framework is close to that of a production-level application, and the system can be easily extended to a cloud service serving a large number of users if given sufficient hardware resources.

1.4 Functionality

Two main functions are supported by the developed system, and both are described in the following use cases.

use case 1: generation of BVH file

workflow: The user selects and uploads a demo video on the Unity client and clicks "Send",

The system backend receives the request for 3D human pose estimation and returns the result to Unity client,

Unity client asks the user to make several settings to create a BVH file, including the factor for data smoothing, interpolation of frames and the path for saving the file. Then, a BVH file will be created and saved.

use case 2: visualization of motion

workflow: The user selects a local BVH file and loads it into the Unity client.

Unity client parses the BVH file, converts the recorded motion data into Unity character animation, and plays it.

2. Deep Learning Based 3D Human Pose Estimation

2.1 Research Status

Before the introduction of deep learning, people estimate 3D human poses from videos using feature engineering, specifically, by combining the dynamics of human motion. This method is highly dependent on expert experience and has low accuracy. In recent years, deep learning has been introduced into the field of human pose estimation, and the neural network models trained on large-scale datasets could make high-accuracy predictions of 3D human pose in images or videos.

According to the research paradigm, deep learning methods for 3D human pose estimation can be divided into two main categories, namely single-image-based estimation and video-based estimation. The former attempts to recover the 3D pose directly from a single image. Since the 2D pose shown in a single image may correspond to multiple 3D poses, and there is a problem of limb occlusion, this method cannot make stable pose estimation for actual scenes. Video-based pose estimation uses a series of adjacent frames to jointly infer the 3D pose of the target frame. Temporal information is involved in such models, and the richer context enables relatively stable inferences.

Video-based pose estimation methods can also be divided into two main categories according to the model structure. The first one is based on the structure composed of a set of recurrent neural networks (e.g., RNN, LSTM) [3], where each frame is encoded and fed into the network sequentially. This is a natural way to model a pose sequence, and such models have achieved excellent performance on 3D pose estimation tasks. However, the training and inference of the network are slow because the forward propagation of recurrent units cannot be conducted in parallel. The second approach uses a temporal convolutional network(TCN) structure[2], which captures the contextual information contained in adjacent frames with convolutional operations. Compared to the temporal-network-based structure, this kind of model can aggregate information from frames farther away from the target frame rather than just utilizing the information involved in previous frames. In addition, the convolution can be operated in parallel, making its training and inference fast.

3D pose estimation models can be further divided into end-to-end models and two-stage models. The end-to-end model takes the original image as input and outputs the pose in the form of coordinates or a 3D heat map directly[4]. However, the network may be difficult to train and the prediction is not stable due to the high-nonlinear of the mapping from a 2D image to a 3D pose. The two-stage model firstly takes the original images as input and outputs 2D poses, and then takes the resulting 2D poses as input and outputs 3D poses[2,5]. The two components of the two-stage model can be trained independently, and then they conduct inference jointly. In the two-stage framework, the first-stage model can be regarded as the feature extractor, which downscales high-dimensional information involved in the original images into a low-dimensional 2D coordinate sequence, and can be trained with supervision, which greatly reduces the nonlinearity of the mapping from video to 3D poses.

2.2 Dataset

High accuracy datasets with sufficient samples and are cruicial for deep learning research. In tasks of 3D human pose estimation, experiments are usually conducted based on two datasets, namely Human3.6m[6] and HumanEva[7,8].

Human3.6m

The Human3.6m dataset contains 3.6 million frames with corresponding annotations of human body bounding boxes and 3D poses. The image sequences were captured by multiple high-resolution cameras at a frequency of 50 Hz, and the pose annotations were obtained by capturing 11 actors performing 17 different actions using a high-precision motion capture system. Due to its high accuracy, Human3.6m has become a standard dataset in related fields.

Figure 2.1: An example of sample frames in Human3.6m.

HumanEva

HumanEva is a relatively small dataset containing 40,000 images and 3D pose annotations. The annotations were obtained by a motion capture system for 4 actors in 6 action scenes.

2.3 Evaluation Metrics

There are two widely used evaluation metrics in 3D pose estimation task, the first one is the mean per joint position error (MPJPE), which computes the average Euclidean distance between predicted joint positions and ground-truth joint positions. MPJPE is computed as

$$E_{MPJPE}(f, gt) = \frac{1}{N_s} \sum_{i=1}^{N_J} \left\| p_f(i) - p_{gt}(i) \right\|_2 \qquad \text{(Eq. 2.1)}$$

where N_s denotes the number of joints in a skeleton, $p_f(i)$ denotes the predicted position of joint i, $p_{gt}(i)$ denotes the ground truth position.

Another metirc is the procrustes mean per joint position error (P-MPJPE), which aligns the predicted pose to ground truth by a rigid transformation called Procrustes before computing MPJPE.

3. Ideas and Experiments on Deep learning Models for 3D Pose Estimation

3.1 introduction

During the project's development, some research on deep learning models for 3D human pose estimation was conducted. To be precise, modification was made to the structure of the VideoPose3D model, and experiments were conducted to validate the effectiveness of proposed ideas. Please note that some of the modified model does not perform better than the original model. This section simply record proposed ideas and related experiments when doing this project. Further exploration will be done in the future.

3.2 VideoPose3D Model

Overview

VideoPose3D is a deep learning model for 3D human pose estimation proposed and opened by a Facebook research team in 2018, which is the second stage model under the two-stage framework (as described in Section 2.1). It takes 2D human pose sequences as input and outputs 3D human pose sequences. VideoPose3D uses temporal convolution modules to perform one-dimensional convolution operations on a feature sequence.

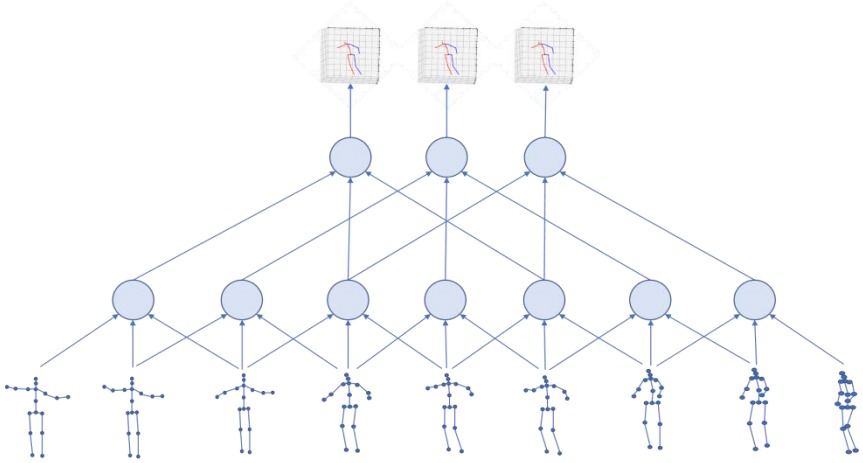

Figure 3.1: An illustration of VideoPose3, which takes 2D poses as input and output corresponding 3D poses.

The input 2D pose can be obtained from any of the 2D pose estimation models, so that the model can be flexibly interfaced with other 2D pose detectors to make up a complete 3D pose detector.

Skeleton

The skeleton and the names of joints used by VideoPose3D are shown in Figure 3.2. The skeleton contains 17 human body joints, and they are indexed in an order consistent with that of the skeleton in Human3.6m.

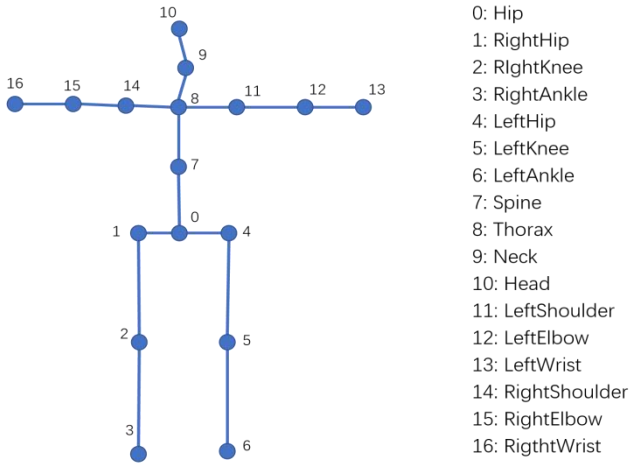

Figure 3.2: Human skeleton used in VideoPose3D.

Model Structure

The structure of the VideoPose3D model is shown in Figure 3.3, which is consisted of a set of convolutional layers with residual blocks for aggregating both high and low dimensional features. In the process of forward propagation, the channel number of the feature vector is kept at 1024, and the perceptual field of the convolutional kernel increases exponentially by a factor of 3. The maximum perceptual field could be 243, 81, or 27.

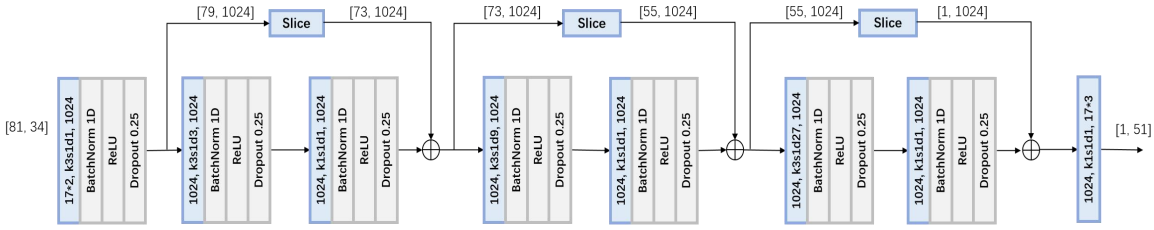

Figure 3.3: VideoPose3D model with a maximum perceptual field of 81 frames. The sequence's length reduces from 81 to 1 after several convolutional layers, and the first layer is used to increase the channel number of input features. Convolutional layers are in blue where *1024, k3s1d3, 1024* denotes 1024 input channels, kernels of size 3 moving at step 1 with dilation 3, and 1024 output channels.

It can be seen that the model has a fully convolutional structure whose input and output can be of indefinite length, and can therefore handle pose sequences of random lengths. For example, for a 2D pose sequence of length N, a padding operation is performed on both its left and right sides to make a sequence of length (40+N+40), which is then fed into the network and will result in an output of the same length as input.

Dilated Convolution

Typically, convolution kernels operate on a contiguous segment of a sequence, where the width of the receptive field is equal to the size of the kernel. By contrast, dilated convolution operates on a discontinuous segment of a sequence, which enables it to expand the width of receptive field without introducing additional parameters, and thus capture richer contextual information. All the

convolution operations in the VideoPose3D model are performed in the form of dilated convolution.

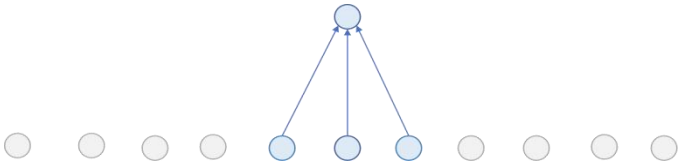

Figure 3.4(a): Dense Convolution

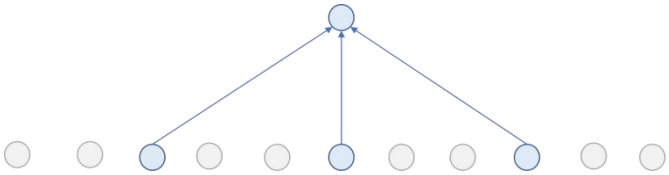

Figure 3.4(b): Dilated Convolution

Figure 3.4: Comparison of dense and dilated convolution. The blue part in the diagram shows the convolution kernel's operating field. Two types of convolution kernels have the same number of parameters. The receptive field size of the dense convolution kernel is 3, while the receptive field size of the dilated convolution kernel is 7.

Model Optimization

The research team at VideoPose3D proposed two models, a model that can handle inputs with indefinite inputs(as described previously), and a single-frame prediction model with fixed-length inputs. The indefinite-length model uses regular convolution, while the single-frame prediction model uses sparse convolution. Two types of convolution are shown in Figure 3.5.

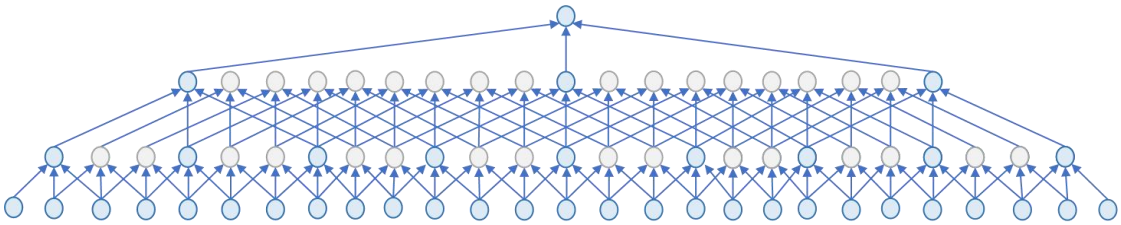

Figure 3.5(a): Regular convolution. The convolution kernel moves in steps of 1. For single-frame prediction models, some of the intermediate outputs (colored grey) are redundant.

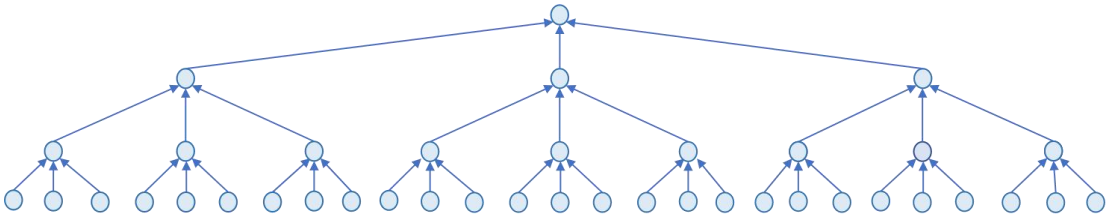

Figure 3.5(b): Sparse convolution. The convolution kernel moves in steps equal to its receptive field size.

Figure 3.5: Comparison of regular convolution and sparse convolution.

In a single-frame prediction model, many of the intermediate outputs produced by dense convolution layers are not involved in the subsequent forward propagation, which means that a large number of parameters in the model are redundant. This could be avoided by making the convolution kernel move at a step of the size equal to its perceptual field size.

Training Setup

For both the indefinite-length model and the single-frame prediction model, VideoPose3D trains them in the form of single-frame prediction to ensure the good generalization ability of the model. Specifically, the frames in the training set are divided into sequences of equal length, and the length is equal to the width of the maximum receptive field of the convolution kernel, which ensures that the output is a single 3D pose. MPJPE metric is used as the loss function, with the initial learning rate set to 0.001 and decayed at a rate of 0.95 after each epoch. The batch size is set to 1024, and the model is trained for 80 epochs.

3.3 Experiment 1: model the potential connections between different frame fields

Idea

The VideoPose3D model is essentially a convolutional structure operating on sequential information, which controls the receptive field's width by applying convolution kernels with different sizes. A larger receptive field would capture more contextual information. However, the convolution kernel operates on independent parts of the input sequence, which results in independence between the intermediate features output by the convolutional layer. In the 3D pose estimation task, frames further away from the target frame may provide more critical information than adjacent frames, so it is necessary to introduce certain mechanism into the network to model potential links between distinct frame fields.

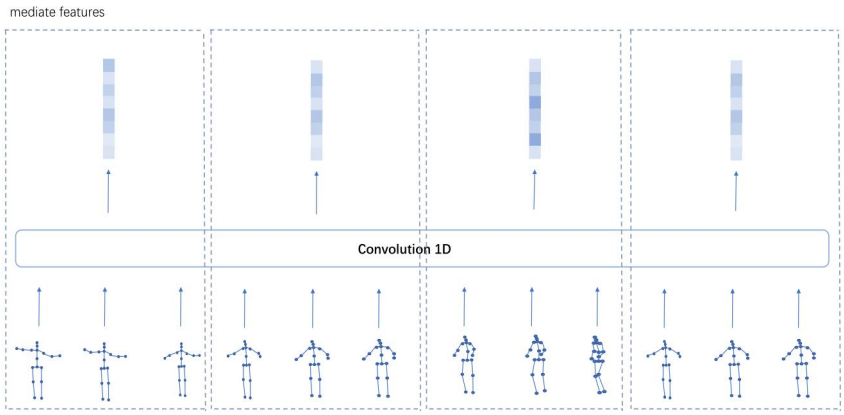

Figure 3.6(a): Independence between distinct frame fields.

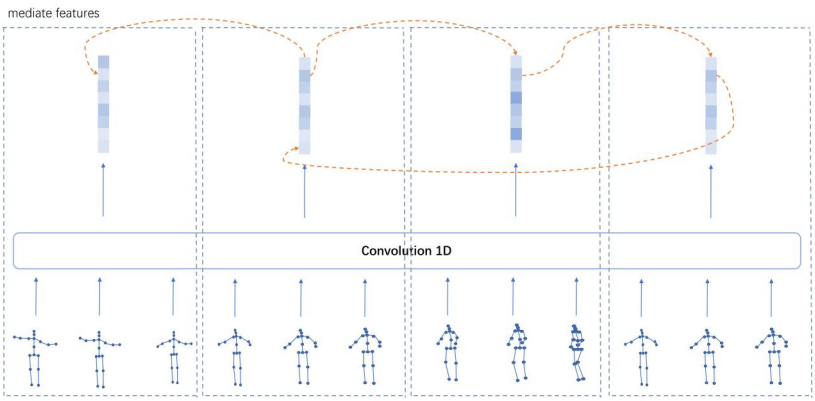

Figure 3.6(b): Protential links between distinct frame fields.

Figure 3.6: Each of the three input poses inside the dashed box in Figure (a) constitutes a frame field. Under the convolution operation, each frame field produces a feature, and the features are independent of each other. Figure (b) shows the potential links that may exist between fields.

Here, frame fields are modeled as a graph, where each of the intermediate features produced by a frame field is considered as a node of the graph, and the potential connections between the frame fields are considered as edges. The graph referred to here is implicit, that is, there are no explicit edges. However, the unknown connection between two frame fields can be represented by learnable parameters. To be precise, the intensity of a connection could be represented by a variable, which will be iteratively updated in training process. Furthermore, the introduction of the attention mechanism in the graph enables the network to determine the intensity of connections between nods based on input data.

Graph Convonlutional Network

Graph convolutional network(GCN)[9] is a model that operates on the graph data structure and makes use of the graph's topological information. GCN has been widely used in tasks related to the graph data. In a graph convolution network, each node is assigned a feature (usually a one-dimensional vector) ,and the graph convolution is essentially an operation that aggregates the feature of nodes. Usually, a node's new feature is determined by its own feature together with those of adjacent nodes. The forward propagation of a single-layer GCN can be described by equation 3.1:

$$X^{(l+1)} = \sigma(AX^{(l)}W^{(l)}) \qquad \text{(Eq. 3.1)}$$

where A is the constant adjacency matrix defined by the topology of the graph. $X^{(l)}$ is a matrix consisting of the feature vectors of all node at layer l . $W^{(l)}$ is a learnable parameters matrix used to transform the dimension of features. σ is the activation function.

Graph Attention

When aggregating features in GCN, there is an assumption that all adjacent nodes have the equal contribution to the central node, while this may not be a good way to model the relationship between nodes. By introducing attention mechanism into GCN, the network is able to learn the

contribution of a node to its neighbours based on the values of node features. For example, let *N(i)* be the set of adjacent nodes of node i, then the weight of node j in the set *N(i)* when computing the new feature of node i is defined as equation 3.2:

$$\alpha_{ij} = softmax\ (\alpha \cdot [h_i \parallel h_j]) = \frac{e^{\sigma(\alpha \cdot [h_i \| h_j])}}{\sum_{k=1}^{N(i)} e^{\sigma(\alpha \cdot [h_i \| h_k])}} \quad \text{（Eq. 3.2）}$$

where h_i and h_j are the feature vectors of nodes i and node j respectively, $\parallel$ denotes concantenation operation, and α is a learnable parameter vector. The results of equation 3.2 are normalized weights across all adjacent nodes of node i. All the weights a_{ij} make up the attention matrix denoted as *Atten*. When introducing attention module into GCN, the forward propagation of the network could be described by equation 3.3.

$$X^{(l+1)} = \sigma((softmax\ (Atten) \otimes A) X^{(l)} W^{(l)}) \quad \text{（Eq. 3.3）}$$

where $\otimes$ denotes element-wise multiplication of two matrices.

Apply Graph Attention to VideoPose3D

Graph attention is applied to the feature sequence output by the convolution layer in VideoPose3D model. Specifically, the features associated with different frame fields are considered nodes of a graph, and the attention mechanism is used to calculate the intensity of connection between features. The attention module here differs from the classical graph attention mechanism described previously: firstly, the feature sequence is not an explicit graph so there is no a adjacency matrix defining its topology. For this reason, a learnable parameter matrix, denoted as *IntrinA*, is introduced into the model to represent the potential intrinsic connections between frame fields, which will be itetatively updated during the training process but will be set as a constant matrix when training is over. Secondly, to reduce the complexity of the model, the attention module is applied directly to the feature sequence without introducing an additional matrix of projection (i.e., the parameter matrix W in Eq.3.1). The expression for the forward propagation of the modified graph attention is described as equation 3.4.

$$X^{(l+1)} = \sigma(softmax(softmax(Atten) \otimes IntrinA) X^{(l)}) \quad \text{(Eq. 3.4)}$$

Model Structure

Experiments for graph attention were conducted based on the single-frame prediction model with an input length of 27 frames. The structures of the original model and the model integrated with the graph attention block are shown in figure 3.7(a) and 3.7(b), respectively.

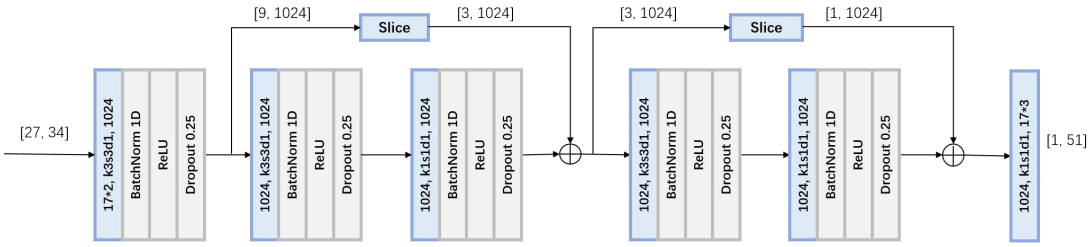

Figure 3.7(a): The original single-frame prediction model with an input length of 27 frames.

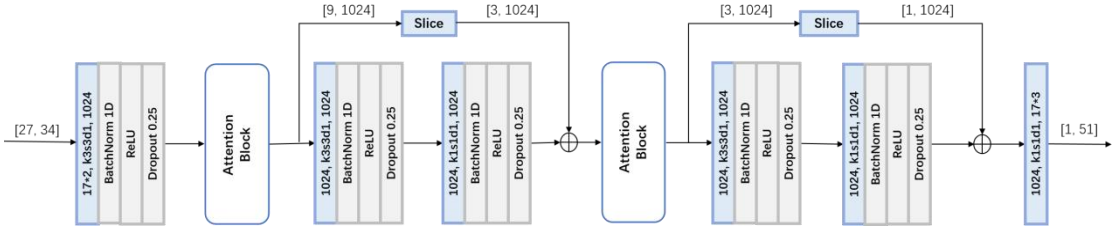

Figure 3.7(b): Single-frame prediction model with attention block.

The structure of the modified model is generally similar to that of VideoPose3D. The only difference is that the graph attention block is added between the convolutional layers. Please refer to Appendix A.1 for detailed internal structure of the graph attention block.

Experimental Results

The training parameters were consistent with the original VideoPose3D paper (as described in section 3.2), and the variation of loss in training process is shown in figure 3.8.

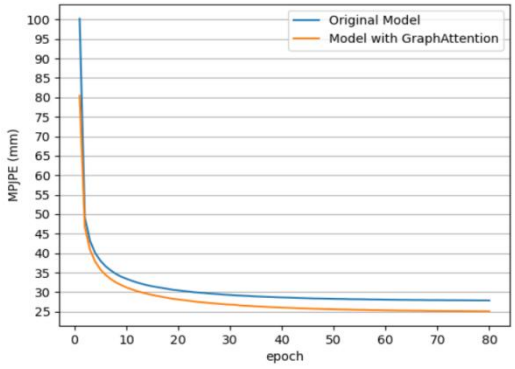

Figure 3.8: Variation of loss in training process.

After training, the performance of the model was evaluated on the test dataset involving 11 action categories, using both MPJPE and P-MPJPE metrics. The reconstruction error and the model's number of parameters are shown in Table 3.1.

Model	Dir.	Disc.	Eat	Greet	Phone	Photo	Pose	Purch.	Sit	Sit.D	Smoke	Wait	WalkD	Walk	WalkT.	Avg	Parameters
VideoPose3D (single-frame prediction)	46.1	49.4	45.2	48.7	51.5	58.8	46.9	45.1	59.6	66.0	50.2	46.9	51.8	36.4	38.9	49.4	8.56M
VideoPose3D + GraphAttention	44.0	47.7	38.3	46.2	46.2	52.2	46.9	43.1	50.2	54.7	45.9	47.5	50.1	39.2	40.1	46.2	8.56M

Table 3.1(a): Reconstruction error (MPJPE)

Model	Dir.	Disc.	Eat	Greet	Phone	Photo	Pose	Purch.	Sit	Sit.D	Smoke	Wait	WalkD	Walk	WalkT.	Avg	Parameters
VideoPose3D (single-frame prediction)	35.0	38.1	36.1	39.0	39.1	45.0	35.8	34.6	47.6	52.1	40.0	35.6	40.8	27.9	31.7	38.6	8.56M
VideoPose3D + GraphAttention	32.9	36.7	30.9	35.9	33.5	39.9	33.6	31.6	38.0	42.7	35.2	34.6	39.4	30.6	31.0	35.1	8.56M

Table 3.1(b): Reconstruction error (P-MPJPE)

Compared to the original model, the error of the model with the graph attention decreased by 6.5% and 9.1% for MPJPE and P-MPJPE, respectively, while the number of model parameters increased by no more than 10,000.

Ablation Study

As mentioned above, the graph attention block contains two main components, the parameter matrix *IntrinA* used to model the potential intrinsic connections between frame fields, and the data-dependent attention module. To evaluate the effectiveness of each of the two components, they were added separately to the original VideoPose3D, and experiments were conducted on the modified models. The evaluation results are shown in Table 3.2.

Model	Dir.	Disc.	Eat	Greet	Phone	Photo	Pose	Purch.	Sit	Sit.D	Smoke	Wait	WalkD	Walk	WalkT.	Avg	Parameters
VideoPose3D+IntrinA+Attention	44.0	47.7	38.3	46.2	46.2	52.2	46.9	43.1	50.2	54.7	45.9	47.5	50.1	39.2	40.1	46.2	8.56M
VideoPose3D + IntrinA	45.8	51.1	42.7	48.7	50.3	59.6	49.1	47.7	59.7	65.4	49.6	50.0	54.2	40.5	43.5	50.5	8.56M
VideoPose3D + Attention	46.0	49.1	40.1	47.8	47.4	54.5	48.0	44.3	52.2	55.6	47.1	48.0	51.4	40.7	42.3	47.6	8.56M

Table 3.2(a): Reconstruction error (MPJPE)

Model	Dir.	Disc.	Eat	Greet	Phone	Photo	Pose	Purch.	Sit	Sit.D	Smoke	Wait	WalkD	Walk	WalkT.	Avg	Parameters
VideoPose3D+IntrinA+Attention	32.9	36.7	30.9	35.9	33.5	39.9	33.6	31.6	38.0	42.7	35.2	34.6	39.4	30.6	31.0	35.1	8.56M
VideoPose3D + IntrinA	35.4	39.4	34.1	39.0	35.7	45.2	36.7	34.7	44.0	47.8	37.5	36.6	42.2	32.3	34.7	38.3	8.56M
VideoPose3D + Attention	34.6	37.3	32.3	37.6	34.2	41.1	34.3	32.8	39.7	43.6	36.1	34.8	40.2	32.6	32.6	36.2	8.56M

Table 3.2(b): Reconstruction error (P-MPJPE)

The evaluation results show that neither the potential adjacent matrix nor the attention module alone works better than the combination of the two.

3.4 Experiment 2: Encode the pose sequence using Graph Convolutional Network

Idea

VideoPose3D uses a set of 2D coordinates to encode a frame as the initial input, and then uses a convolutional layer to raise the feature's channel size from 34(17*2) to 1024. It is a natural way to encode a 2D pose with a set of 2D coordinates, but the elevation from the coordinate set to a 1024-length vector seems to lack interpretability. It is possible to encode 2D pose sequences in a more intuitive way. The human pose can be naturally represented as a graph: the joints are considered as nodes ,and the bones as edges. In this view, a 2D pose sequence can be encoded using a GCN layer.

GCN as the Pose Encoder

For a 2D pose, take its set of coordinates as the initial features, the connectivity of the joints as the adjacency matrix, then encoding could be conducted by applying graph convolution to the pose sequence. In this way, the feature corresponding to a pose is two-dimensional: the number of joints and channel number of the feature vector.

Model Structure

When using GCN to encode poses, the intermediate inputs and intermediate outputs of the network are two-dimensional, so the 1D convolutional layers in the original model need to be replaced with 2D convolutional layers. Figure 3.9 is an illustration of using GCN to encode a pose sequence. The graph convolution layer and the regular convolution layer together make up a network module, and each frame is encoded into a feature of size [17, *C*], where 17 denotes the number of joints in skeleton, and *C* denotes the channel number in the feature vector.

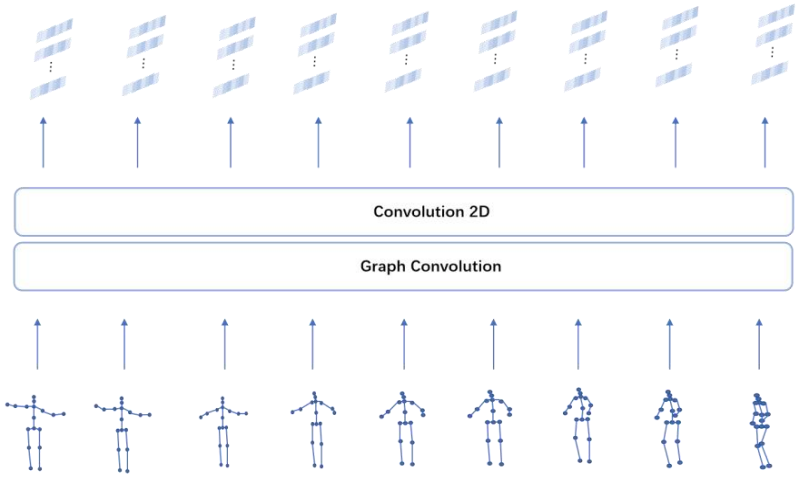

Figure 3.9: An illustration of encoding a pose sequence with GCN.

If the channel size of the feature is kept at 1024 in forward propagation (as in the original VideoPose3D model), the model's parameter amount will be very large, which may lead to difficulties in training, and the intermediate layer output will occupy a large amount of memory. This needs to be simplified due to insufficient hardware and limited time. Inspired by the ResNet [10], which is widely used in the field of image understanding., the model is modified to a structure where the channel size of the feature gradually increases from 2 to 512 in forward propagation. The structure of the modified model is shown in Figure 3.10.

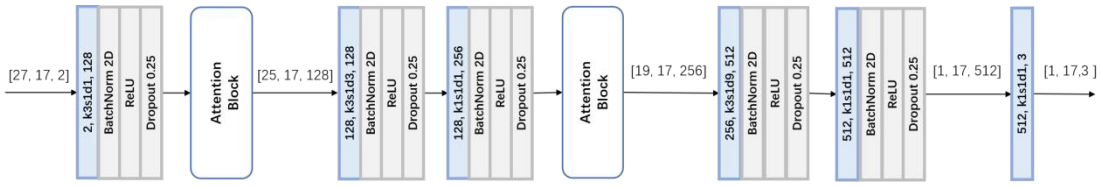

Figure 3.10: The model with a modified structure where the graph convolutional network is used to encode the 2D pose sequence. Compared to the original model, the feature dimension increases with the depth of the network, and

the residual blocks are removed to reduce the model complexity.

In GCN layer, the connections between the joints naturally define an adjacent matrix, while two modification is made based on the original GCN. Firstly, a learnable parameter matrix is used to model the inherent but unknown links between joints. Secondly, the attention module is added to enable the network to learn the data-dependent links between joints. Both of the ideas are similar to that described in section 3.3.

When applying the modified GCN to encode pose sequences, the process of feature aggregation consists of three parts: the first is feature aggregation based on the adjacent matrix defined by the skeleton (denoted as A). The second is based on the matrix of learnable parameters (denoted as $IntrinA$). The third is based on the attention module (denoted as $Atten$). For a given joint, the first part of the feature aggregation is performed only on its neighbouring joints, while the second and third parts are performed on all joints. The corresponding forward propagation is described as equation 3.3, where $W^{(l)}$ denotes the parameter matrix used to perform linear transform on the input feature sequence $X^{(l)}$:

$$X^{(l+1)} = \sigma(softmax(Atten + IntrinA + A)X^{(l)}W^{(l)}) \quad \text{(Eq. 3.11)}$$

Two types of aggregation are shown in Figure 3.11.

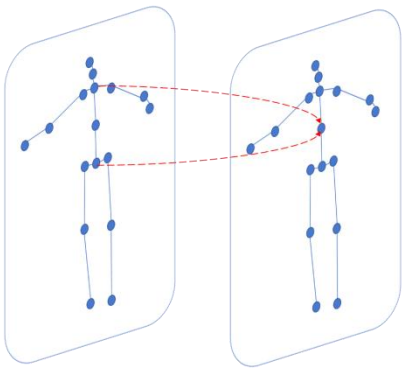

Figure 3.11(a): Feature aggregation based on the adjacency matrix defined by skeleton.

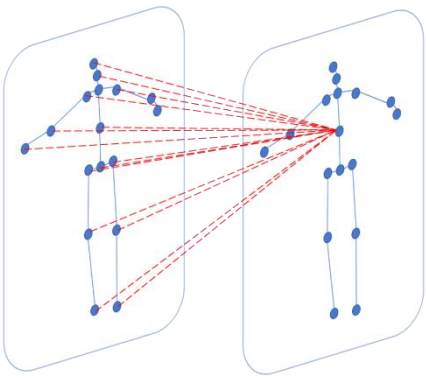

Figure 3.11(b): Feature aggregation based on potential learnable adjacent matrix or attention module. All joints are

assigned a weight to the target joint.

Figure 3.11: An illustration of feature aggregation in the modified GCN.

Please refer to Appendix A.2 for the detailed structure of the modified graph convolution layer with attention module.

Experimental Results

Experiments were conducted based on the multi-frame prediction model with a maximum receptive field of 27 frames. The original VideoPose3D model and the model with the modified GCN(named as GCNAttention) were trained and evaluated respectively. The variation of loss in training process is shown in Figure 3.12.

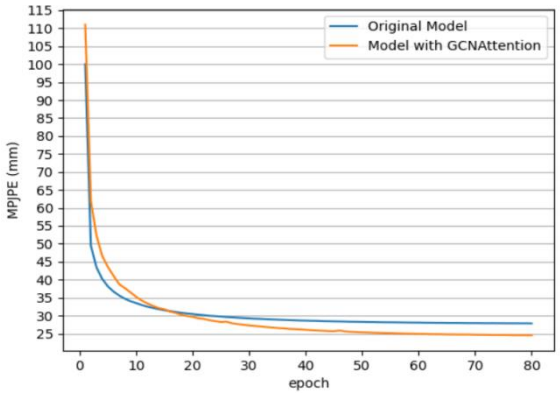

Figure 3.12: Variation of loss in training process.

After training, the performance of the model was evaluated on the test dataset involving 11 action categories, using both MPJPE and P-MPJPE metrics. The reconstruction error and the model's number of parameters are shown in Table 3.3.

Model	Dir.	Disc.	Eat	Greet	Phone	Photo	Pose	Purch.	Sit	Sit.D	Smoke	Wait	WalkD	Walk	WalkT.	Avg	Parameters
VideoPose3D (indefinite input)	46.2	48.7	45.4	48.0	51.5	58.2	47.1	44.6	58.2	65.5	49.3	46.3	51.9	36.5	39.1	49.1	8.56M
GCNAttention	54.1	56.4	50.9	54.6	56.1	66.4	52.0	52.8	66.9	86.7	54.0	53.2	57.2	39.9	40.9	56.1	2.90M

Table 3.3(a): Reconstruction error (MPJPE)

Model	Dir.	Disc.	Eat	Greet	Phone	Photo	Pose	Purch.	Sit	Sit.D	Smoke	Wait	WalkD	Walk	WalkT.	Avg	Parameters
VideoPose3D (indefinite input)	34.6	37.6	35.7	38.4	38.7	44.4	35.6	34.0	46.3	51.8	39.3	35.1	40.2	27.8	31.3	38.0	8.56M
GCNAttention	38.2	42.0	40.3	42.3	41.4	49.6	38.6	39.4	53.2	66.7	42.5	38.9	42.8	29.3	31.3	42.4	2.90M

Table 3.3(b): Reconstruction error (P-MPJPE)

The number of parameters of the modified model is only 33.9% of that of the original model, while the errors in two evaluation metrics increased by 14.3% and 11.6% respectively.

Ablation Study

As mentioned previously, the adjacent matrix in the graph convolution layer used to encode the pose sequence consists of three components: a constant adjacent matrix defined by the skeleton, a

weight matrix computed by the attention module, and a learnable matrix representing the potential connectivity between joints. Three modules were added separately to the original model to evaluate the effectiveness of each of the three components. The evaluation results are shown in Table 3.4.

Model	Dir.	Disc.	Eat	Greet	Phone	Photo	Pose	Purch.	Sit	Sit.D	Smoke	Wait	WalkD	Walk	WalkT.	Avg	Parameters
GCNAttention	54.1	56.4	50.9	54.6	56.1	66.4	52.0	52.8	66.9	86.7	54.0	53.2	57.2	39.9	40.9	56.1	2.90M
GCNAttention remove A	56.2	58.4	51.8	56.0	58.2	70.6	52.7	54.7	70.5	77.6	56.6	54.9	59.9	41.7	42.3	57.5	2.90M
GCNAttention remove IntrinA	231.9	218.1	206.9	234.8	207.5	250.3	231.3	203.2	200.7	198.1	205.1	213.0	210.2	210.1	228.4	216.6	2.90M
GCNAttention remove Attention	253.3	247.6	229.7	258.8	233.6	300.8	251.7	233.7	232.8	288.1	227.8	243.0	245.0	219.7	240.4	247.1	2.90M

Table 3.4(a): Reconstruction error(MPJPE)

Model	Dir.	Disc.	Eat	Greet	Phone	Photo	Pose	Purch.	Sit	Sit.D	Smoke	Wait	WalkD	Walk	WalkT.	Avg	Parameters
GCNAttention	38.2	42.0	40.3	42.3	41.4	49.6	38.6	39.4	53.2	66.7	42.5	38.9	42.8	29.3	31.3	42.4	2.90M
GCNAttention remove A	39.4	42.2	41.2	43.4	42.1	51.8	39.3	40.1	54.8	61.2	43.4	40.4	45.1	30.3	32.0	43.1	2.90M
GCNAttention remove IntrinA	249.7	219.3	212.7	247.1	209.7	266.6	234.3	206.5	197.9	187.7	202.1	219.4	206.5	202.6	223.8	219.1	2.90M
GCNAttention remove Attention	257.5	235.0	235.0	260.6	221.7	277.4	246.6	223.6	217.6	218.0	215.5	233.6	226.8	211.2	234.8	233.6	2.90M

Table 3.4(b): Reconstruction error(P-MPJPE)

It is clear that removing any of the three modules in GCNAttention would reduce the performance of the model, and there is a significant difference between the three models removing different modules. Taking the MPJPE metric as an example, removing the constant adjacent matrix A increased the error by only 2.5%, while removing the learnable parameter matrix *IntrinA* and attention module increased the error by 286% and 340%, respectively, indicating that the *IntrinA* and attention module contributed much more to the model's performance than the constant matrix A.

4.Generate and visualize a BVH file

4.1 Introduction

The pose estimation model outputs a sequence of 3D coordinates of joints, which needs to be converted into the standard file format in motion capture and character animation system so that it can be directly used in animation editing tools. In section 4, the method of converting 3D coordinate sequences into BVH files will be given.

4.2 Human pose representation in character animation

In character animation, the human body is represented hierarchically, with each joint associated with a parent and child joint, and a joint being set to be the root (usually the hip). The diagram shows a common hierarchical representation of the human skeleton.

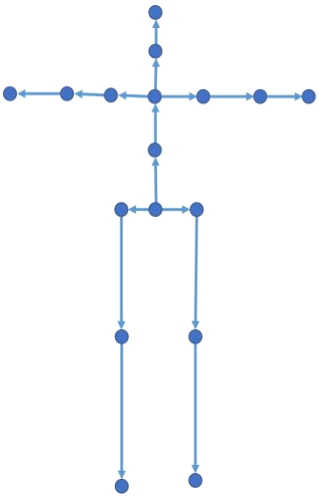

Figure 4.1: Hierarchical skeleton. Each arrow points from the parent joint to the child joint.

Other than the end joints (i.e., the joints without a children joint), each joint is associated with a bone and a local coordinate system, which is used to describe the orientation of the bone (i.e., the amount of rotation). In the animation system, the body pose in each frame is determined by the initial pose and the rotation information in the current frame. Specifically, the root joint has six degrees of freedom, including three translation degrees and three rotation degrees, while the other joints have only three rotation degrees. This is because once the root joint's position is determined, the local rotation of each joint relative to its parent is the only information needed to uniquely determine a pose.

Here is a detailed description of how the animation system determines the body's pose for each frame. Figure 4.2(a) shows three joints and two associated bones, where the two local coordinate systems remain parallel. When joint B rotates by a certain angle relative to joint A, the bones transform to a pose shown in Figure4.2(b), and the two local coordinate systems get unparallel. In this case, some transformation is required to get joint P's coordinates under coordinate system A.

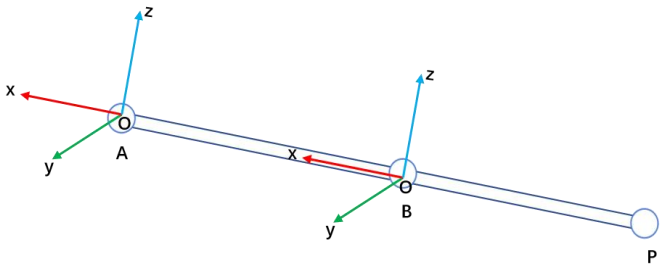

Figure 4.2(a): Before transformation, local coordinate system A and B keep parallel.

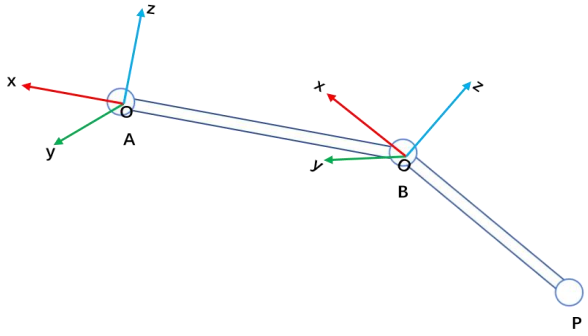

Figure 4.2(b): After transformation.

To better illustrate the calculation, we first translate the coordinate system A to make its origin coincides with that of joint B, and denote the new coordinate system as A'. Assuming that the transition matrix from system A' to system B is $R_{A'B}$, then the relationship between the basis vectors of two coordinate systems can be described as:

$$[\mathbf{e}_{Bx}, \mathbf{e}_{By}, \mathbf{e}_{Bz}] = [\mathbf{e}_{A'x}, \mathbf{e}_{A'y}, \mathbf{e}_{A'z}] R_{A'B} \qquad \text{(Eq. 4.1)}$$

where the bolded letter **e** denotes the basis vectors of a coordinate system.

Suppose that the coordinates of joint P under the two local coordinate systems are $p_{A'}$ and p_B, respectively, then we have:

$$[\mathbf{e}_{A'x}, \mathbf{e}_{A'y}, \mathbf{e}_{A'z}] p_{A'} = [\mathbf{e}_{Bx}, \mathbf{e}_{By}, \mathbf{e}_{Bz}] p_B \qquad \text{(Eq. 4.2)}$$

Substituting equation 4.1 into 4.2 yields:

$$[\mathbf{e}_{A'x}, \mathbf{e}_{A'y}, \mathbf{e}_{A'z}] p_{A'} = [\mathbf{e}_{A'x}, \mathbf{e}_{A'y}, \mathbf{e}_{A'z}] R_{A'B} p_B \qquad \text{(Eq. 4.3)}$$

Since $[\mathbf{e}_{A'x}, \mathbf{e}_{A'y}, \mathbf{e}_{A'z}]$ is a linearly independent matrix, simultaneous elimination of this matrix on both sides of Eq. 4.3 yields:

$$p_{A'} = R_{A'B} p_B \qquad \text{(Eq. 4.4)}$$

The transformation from coordinate system A' to coordinate system A is performed by simply adding an offset:

$$p_A = OFFSET_{AB} + R_{A'B} p_B \qquad \text{(Eq. 4.5)}$$

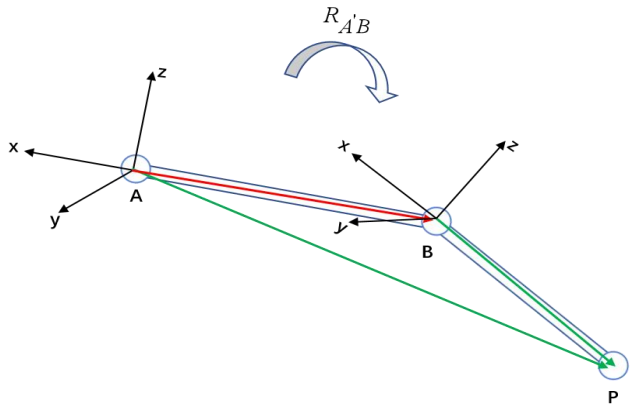

Figure 4.3: An illustration of transforming coordinates from one system to another. The green vectors represents joint P's coordinates in system A and B respectively, and the transformation between them is performed using rotation matrix $R_{A'B}$ and offset AB(the red vector).

The above derivation only gives a description of transforming coordinates from the child system to its first level of parent. Suppose that there are multiple levels of parent joints of joint P, set as $F_0, F_1, ...F_n$,then the transformation from the local system B to the global system F_0 can be recursively calculated as:

$$p_A = OFFSET_{AB} + R_{AB} p_B$$
$$p_{F_n} = OFFSET_{F_n A} + R_{F_n A} p_A$$
.....
$$p_{F_0} = OFFSET_{F_0 F_1} + R_{F_0 F_1} p_{F_1}$$

Substituting the variables in turn, we get:

$$p_{F_0} = OFFSET_{F_0 F_1} + R_{F_0 F_1}(...(OFFSET_{F_0 A} + R_{F_n A}(OFFSET_{AB} + R_{AB} p_B))...)$$

Expanded the above equation, we get:

$$p_{F_0} = OFFSET_{F_0 F_1} + R_{F_0 F_1} OFFSET_{F_1 F_2} + R_{F_0 F_1} R_{F_1 F_2} OFFSET_{F_2 F_3} + ... + R_{F_0 F_1} R_{F_1 F_2} ... R_{F_n A} R_{AB} OFFSET_{AB}$$

joint A's coordinates in the global system F0 **rotation transformation from system B to F0**

(Eq. 4.6)

The first n+1 terms in equation 4.5 are actually the coordinates of joint A in the global system F_0 , and $R_{F_0 F_1} R_{F_1 F_2} ... R_{F_n A} R_{AB}$ can be combined into $R_{F_0 B}$, which represents the transformation matrix from the global system F_0 to the local system B.

According to equation 4.6, the global coordinates of any joint can be determined given the initial offset of the joint from its parent(or the bone length if the initial pose is set to T-pose) and the joint's local rotation with respect to its parent.

It is worth noting that the character animation system uses local rotations rather than

coordinates to describe human pose in order to make the motion animation portable (i.e., motion redirection). This is because the bone lengths may vary between characters, whereas local rotations are uniquely defined.

4.3 BVH file

BVH is a standard human motion capture file format introduced by BioVision and is currently supported by mainstream modeling software (e.g., Maya, 3D Max, Blender, etc.) to parse BVH files. Figure 4.4 shows the structure of the BVH file content.

```
HIERARCHY
ROOT Hips
{
    OFFSET  0.00    0.00    0.00
    CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
    JOINT RightHip
    {
        OFFSET   0.50    0.00    0.00
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT RightKnee
        {
            OFFSET   0.00    0.00   -1.00
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT RightAnkle
            {
                OFFSET   0.00    0.00    -1.00
                CHANNELS 3 Zrotation Xrotation Yrotation
                End Site
                {
                    OFFSET   0.00    0.50    0.00
                }
            }
        }
        JOINT LeftHip
        {
            ...
        }
            ...
}

MOTION
Frames:     2
Frame Time:  0.033333
-14.60  7.88   43.80 -3.41   14.78 -164.35 13.09  40.30   0.00  -3.61  -41.40  5.82   10.08  0.00   10.21  97.95
-13.53  -2.14  -101.86 -80.77 -98.91  0.69   0.03   0.00  -14.04  0.00  -20.50 -85.52 -13.72 -102.93 61.91 -61.18
65.18 -1.57   0.69   0.02   15.00  20.74 -5.52   14.83  49.99  6.62   0.00  -1.14   0.00  -16.58 -10.51 -3.11  5.38
52.66 -21.80  0.00  -23.95  0.00  8.03   35.01  82.26
 5.80   36.20  86.57 -3.78   12.94 -166.97 12.64  12.57 -22.34  7.67   43.61  0.00  -4.23  -41.30  4.89   19.10
0.00   4.16   92.12 -9.69  -9.43   132.67 -81.86  16.80 0.70   0.37   0.00  -8.62   0.00  -21.82 -88.31 -25.57 -
100.09 56.16  -61.56  58.72 -1.63   0.95   1.03   13.16  15.44 -3.56   7.97   59.29  4.97   0.00   1.64   0.00  -
17.18 -10.02  -3.08   13.56  53.38 -18.07  0.00  -25.53  0.00
```

Figure 4.4: An example of BVH file format

The keyword HIERARCHY in the file marks the beginning of the structure part, which records the tree-like hierarchical relationships between the joints of the skeleton. Where ROOT marks the root joint, CHANNELS marks the degrees of freedom a joint has, and OFFSET marks the initial offset of a joint relative to its parent. As described in Section 4.2, the root joint has six degrees of freedom, while the others have three degrees.

Keyword MOTION marks the beginning of the motion information. Frames marks the total number of frames, and Frame Time marks the interval time between frames. The rest of motion part in each row corresponds to the motion data of each frame, arranged in the order in which CHANNELS appears in the HIERARCHY section, and the rotation information is described in terms of Euler angles.

4.4 Generate BVH files using 3D coordinates

As described in Sections 4.2 and 4.3, given the 3D coordinates of each joint of the human skeleton, we need to calculate the local rotation of each joint with respect to its parent and convert them into Euler angles to generate a BVH file so that the motion data can be directly used in animation editing tools. The workflow consists of three main steps:

(a) Define the local coordinate system of each joint based on its 3D coordinates.

(b) Calculate the local rotation matrix of each joint based on local coordinate systems.

(c) Convert the rotation matrices into Euler angles.

A detailed description of each step is given below.

Firstly, we need to define the local coordinate system of each joint. Use three joints and associated two bones as an example. The initial local coordinate system of joint B is shown in Figure 4.2(a). When B is rotated by a certain angle, given the 3D coordinates of joints A,B,C, the basis vectors of the coordinate system B can be calculated as follows:

$\mathbf{e}_{Bx}$: parallel to the vector CB, normalize CB to get $\mathbf{nCB}$, then $\mathbf{e}_{Bx} = \mathbf{nCB}$.

$\mathbf{e}_{Bz}$:calculate the normalized direction vector $\mathbf{nBA}$, and the cross product of $\mathbf{nCB}$ and $\mathbf{nBA}$, then $\mathbf{e}_{Bz} = \mathbf{nCB} \times \mathbf{nBA}$.

$\mathbf{e}_{By}$:calculate the cross product of $\mathbf{e}_{Bz}$ and $\mathbf{e}_{Bx}$, then $\mathbf{e}_{By} = \mathbf{e}_{Bz} \times \mathbf{e}_{Bx}$.

Note that the method above just gives a general description of how to define a local system. For detailed description for each of the joints in skeleton, please refer to appendix A.3.

After obtaining the local coordinate systems of each joint as described above, we need to calculate the relative rotation matrix between the coordinate systems. Let the set of basis vectors of systems A and B be $[\mathbf{e}_{Ax}, \mathbf{e}_{Ay}, \mathbf{e}_{Az}]$, $[\mathbf{e}_{Bx}, \mathbf{e}_{By}, \mathbf{e}_{Bz}]$, respectively, and take vector $\mathbf{e}_{Bx}$ as an example. Assume that $\mathbf{e}_{Bx}$'s angles to $\mathbf{e}_{Ax}$, $\mathbf{e}_{Ay}$, $\mathbf{e}_{Az}$ are α , β , γ , respectively, then $\mathbf{e}_{Bx}$ can be expressed as:

$$\mathbf{e}_{Bx} = \cos\alpha \cdot \mathbf{e}_{Ax} + \cos\beta \cdot \mathbf{e}_{Ay} + \cos\gamma \cdot \mathbf{e}_{Az} \qquad \text{(Eq. 4.7)}$$

Rewrite equation 4.7 in fom of vectors as:

$$\mathbf{e}_{Bx} = [\mathbf{e}_{Ax}, \mathbf{e}_{Ay}, \mathbf{e}_{Az}] \begin{bmatrix} \cos<\mathbf{e}_{Bx}, \mathbf{e}_{Ax}> \\ \cos<\mathbf{e}_{Bx}, \mathbf{e}_{Ay}> \\ \cos<\mathbf{e}_{Bx}, \mathbf{e}_{Az}> \end{bmatrix}$$

Since the basis vectors are unit vectors, the above equation can be rewritten in dot product form :

$$\mathbf{e}_{Bx} = [\mathbf{e}_{Ax}, \mathbf{e}_{Ay}, \mathbf{e}_{Az}] \begin{bmatrix} \mathbf{e}_{Bx} \cdot \mathbf{e}_{Ax} \\ \mathbf{e}_{Bx} \cdot \mathbf{e}_{Ay} \\ \mathbf{e}_{Bx} \cdot \mathbf{e}_{Az} \end{bmatrix} \qquad \text{(Eq. 4.8)}$$

In the same way, the other two basis vectors $\mathbf{e}_{By}$ and $\mathbf{e}_{Bz}$ can also be represented by the basis

vectors of system A. Then the transformation from system A to system B can then be expressed as:

$$[\mathbf{e}_{Bx},\mathbf{e}_{By},\mathbf{e}_{Bz}] = [\mathbf{e}_{Ax},\mathbf{e}_{Ay},\mathbf{e}_{Az}]\begin{bmatrix} \mathbf{e}_{Bx}\cdot\mathbf{e}_{Ax} & \mathbf{e}_{By}\cdot\mathbf{e}_{Ax} & \mathbf{e}_{Bz}\cdot\mathbf{e}_{Ax} \\ \mathbf{e}_{Bx}\cdot\mathbf{e}_{Ay} & \mathbf{e}_{By}\cdot\mathbf{e}_{Ay} & \mathbf{e}_{Bz}\cdot\mathbf{e}_{Ay} \\ \mathbf{e}_{Bx}\cdot\mathbf{e}_{Az} & \mathbf{e}_{By}\cdot\mathbf{e}_{Az} & \mathbf{e}_{Bz}\cdot\mathbf{e}_{Az} \end{bmatrix} \quad \text{(Eq. 4.9)}$$

From equation 4.9, we can see that the transition matrix R between the two coordinate systems can be calculated from the dot product between the basis vectors, which can be regarded as a rotation matrix in the geometric sense, also known as the direction cosine matrix.

Finally, we need to convert the rotation matrix to Euler angle form. Here the Euler angles are in intrinsic rotation, and the rotation order is Z-X-Y. In three-dimensional space, assume that a vector v has coordinates $\mathbf{v}A$ under coordinate system A. Then, when it follows system B to rotate around the Z,X,Y axes for angles of (θ,φ,ψ) , the coordinates of rotated vector $\mathbf{v}'A$ under system A can be calculated as follows:

$$\mathbf{v}'\mathrm{A} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{bmatrix}\mathbf{v}\mathrm{A}$$

$$\mathbf{v}'A = \begin{bmatrix} \cos\psi\cos\varphi-\sin\psi\sin\theta\sin\varphi & -\sin\psi\cos\varphi & \cos\varphi\sin\varphi+\sin\psi\sin\theta\cos\varphi \\ \sin\psi\cos\varphi+\cos\psi\sin\theta\sin\varphi & \cos\psi\cos\theta & \sin\psi\sin\varphi-\cos\psi\sin\theta\cos\varphi \\ -\cos\theta\sin\varphi & \sin\theta & \cos\theta\cos\varphi \end{bmatrix}\mathbf{v}A$$

(Eq. 4.10)

As described in Section 4.2, the transformation from $\mathbf{v}A$ to $\mathbf{v}'A$ can also be defined by the transition matrix between two coordinate systems:

$$\mathbf{v}'A = R_{AB}\mathbf{v} \quad \text{(Eq. 4.11)}$$

Let the matrix on the right side of equation 4.10 be T. Comparing equation 4.10 and 4.11, it is clear that the matrix T and R_{AB} are equivalent, and R_{AB} can be determined by equation 4.9, so that we have:

$$\begin{bmatrix} \mathbf{e}_{Bx}\cdot\mathbf{e}_{Ax} & \mathbf{e}_{By}\cdot\mathbf{e}_{Ax} & \mathbf{e}_{Bz}\cdot\mathbf{e}_{Ax} \\ \mathbf{e}_{Bx}\cdot\mathbf{e}_{Ay} & \mathbf{e}_{By}\cdot\mathbf{e}_{Ay} & \mathbf{e}_{Bz}\cdot\mathbf{e}_{Ay} \\ \mathbf{e}_{Bx}\cdot\mathbf{e}_{Az} & \mathbf{e}_{By}\cdot\mathbf{e}_{Az} & \mathbf{e}_{Bz}\cdot\mathbf{e}_{Az} \end{bmatrix} = \begin{bmatrix} \cos\psi\cos\varphi-\sin\psi\sin\theta\sin\varphi & -\sin\psi\cos\varphi & \cos\varphi\sin\varphi+\sin\psi\sin\theta\cos\varphi \\ \sin\psi\cos\varphi+\cos\psi\sin\theta\sin\varphi & \cos\psi\cos\theta & \sin\psi\sin\varphi-\cos\psi\sin\theta\cos\varphi \\ -\cos\theta\sin\varphi & \sin\theta & \cos\theta\cos\varphi \end{bmatrix}$$

(Eq. 4.12)

Given the two local coordinate systems A and B, the left side matrix is known. Comparing the elements on both sides, we can get the three Euler angles performing the rotation from A to B:

$$\theta = \arcsin(\mathbf{e}_{By} \cdot \mathbf{e}_{Az})$$

$$\varphi = \arctan(-\frac{\mathbf{e}_{Bx} \cdot \mathbf{e}_{Az}}{\mathbf{e}_{Bz} \cdot \mathbf{e}_{Az}})$$

$$\psi = \arctan(-\frac{\mathbf{e}_{By} \cdot \mathbf{e}_{Ax}}{\mathbf{e}_{By} \cdot \mathbf{e}_{Ay}})$$

The obtained Euler angles will be written as motion information into the BVH file. Codes for creating a BVH file using 3D coordinates are available in the file 'BvhHandler.cs' in the repository mentioned in section 1.1.

4.5 Motion visualization

Once the BVH file has been generated, the user can load the file and visualize it in the Unity client. A character has been set in client, and the poses recorded in the BVH file will be mapped to the character. With the API provided by Unity, it is easy to create corresponding character animation, while two details need to be considered.

The first point is that the local rotation in the BVH file created by the system is performed in the form of intrinsic Euler Angles(as mentioned in section 4.4), while the pose of the character in Unity is calculated by extrinsic Euler angles. Therefore, before mapping the local rotation to the character skeleton, the intrinsic Euler angles need to be converted to extrinsic ones using DCM matrix as the intermediary. Here gives a detailed description.

For each joint in a frame, let $(\theta_{in}, \varphi_{in}, \psi_{in})$ and $(\theta_{ex}, \varphi_{ex}, \psi_{ex})$ be its local rotation in two types of Euler rotation (in Z-X-Y order), respectively. Firstly, calculate the DCM matrix using the information in the loaded BVH file:

$$DCM = \begin{bmatrix} \cos\psi_{in}\cos\varphi_{in} - \sin\psi_{in}\sin\theta_{in}\sin\varphi_{in} & -\sin\psi_{in}\cos\varphi_{in} & \cos\varphi_{in}\sin\varphi_{in} + \sin\psi_{in}\sin\theta_{in}\cos\varphi_{in} \\ \sin\psi_{in}\cos\varphi_{in} + \cos\psi_{in}\sin\theta_{in}\sin\varphi_{in} & \cos\psi_{in}\cos\theta_{in} & \sin\psi_{in}\sin\varphi_{in} - \cos\psi_{in}\sin\theta_{in}\cos\varphi_{in} \\ -\cos\theta_{in}\sin\varphi_{in} & \sin\theta_{in} & \cos\theta_{in}\cos\varphi_{in} \end{bmatrix}$$

(Eq. 4.13)

Then, rewrite the matrix by extrinsic Euler angle:

$$DCM = \begin{bmatrix} \cos\varphi_{ex} & 0 & \sin\varphi_{ex} \\ 0 & 1 & 0 \\ -\sin\varphi_{ex} & 0 & \cos\varphi_{ex} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_{ex} & -\sin\theta_{ex} \\ 0 & \sin\theta_{ex} & \cos\theta_{ex} \end{bmatrix} \begin{bmatrix} \cos\psi_{ex} & -\sin\psi_{ex} & 0 \\ \sin\psi_{ex} & \cos\psi_{ex} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\psi_{ex}\cos\varphi_{ex} + \sin\psi_{ex}\sin\theta_{ex}\sin\varphi_{ex} & -\cos\varphi_{ex}\sin\psi_{ex} + \sin\varphi_{ex}\sin\theta_{ex}\cos\psi_{ex} & \cos\theta_{ex}\sin\varphi_{ex} \\ \cos\theta_{ex}\sin\psi_{ex} & \cos\psi_{ex}\cos\theta_{ex} & -\sin\theta_{ex} \\ -\sin\varphi_{ex}\cos\psi_{ex} + \cos\varphi_{ex}\sin\theta_{ex}\sin\psi_{ex} & \sin\psi_{ex}\sin\varphi_{ex} + \cos\psi_{ex}\sin\theta_{ex}\cos\varphi_{ex} & \cos\theta_{ex}\cos\varphi_{ex} \end{bmatrix}$$

(Eq. 4.14)

where $(\theta_{in}, \varphi_{in}, \psi_{in})$ are known information in the BVH file, and $(\theta_{ex}, \varphi_{ex}, \psi_{ex})$ are unknown variables we need. Comparing equation 4.13 and 4.14, we get :

$$\theta_{ex} = \arcsin(-(\sin\psi_{in}\sin\varphi_{in} - \cos\psi_{in}\sin\theta_{in}\cos\varphi_{in}))$$

$$\varphi_{ex} = \arctan(\frac{\cos\varphi_{in}\sin\varphi_{in} + \sin\psi_{in}\sin\theta_{in}\cos\varphi_{in}}{\cos\theta_{in}\cos\varphi_{in}})$$

$$\psi_{ex} = \arctan(\frac{\sin\psi_{in}\cos\varphi_{in} + \cos\psi_{in}\sin\theta_{in}\sin\varphi_{in}}{\cos\psi_{in}\cos\theta_{in}})$$

the obtained Euler angles can be directly mapped to the skeleton of character to create a pose for each frame, and then create an animation.

The second point is that the Zero-pose (a pose in which local rotation of all bones is set to zero) of the character in Unity may not be in T-pose, while the rotation recorded in BVH are based on T-pose. Therefore, the rotation from Zero-pose to T-pose needs to be considered when setting local rotation for a bone. Figure 4.5 gives the workflow of creating the pose for a frame.

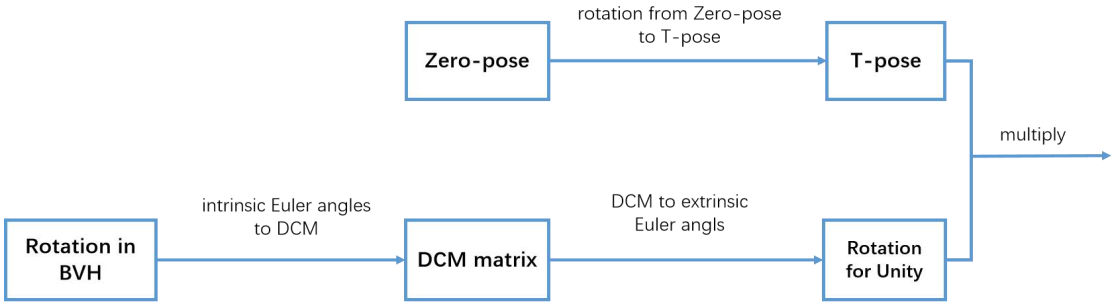

Figure 4.5: Workflow of creating the pose using data in BVH file.

For details of implementation, please refer to codes in the file ‘AnimationHandler.cs’ in the repository mentioned in section 1.1.

where

5. Deployment

5.1 Overview

As described in section 1.3, the system consists of three modules, that is, the Unity client working as the user interface, the request handler connecting the client and Triton, and the Triton inference server responsible for neural network inference. A critical feature of the system is that Triton runs as a separate part in the backend, which means that it could be deployed in an independent machine with abundant hardware resources(mainly the GPU). Section 5 gives a detailed description of the request handler and Triton.

5.2 Request Handler

In fact, the request handler consists of three parts, the proxy service Nginx, the gateway service Gunicorn and the web app. figure 5.1 is an illustration of the internal structure of the module.

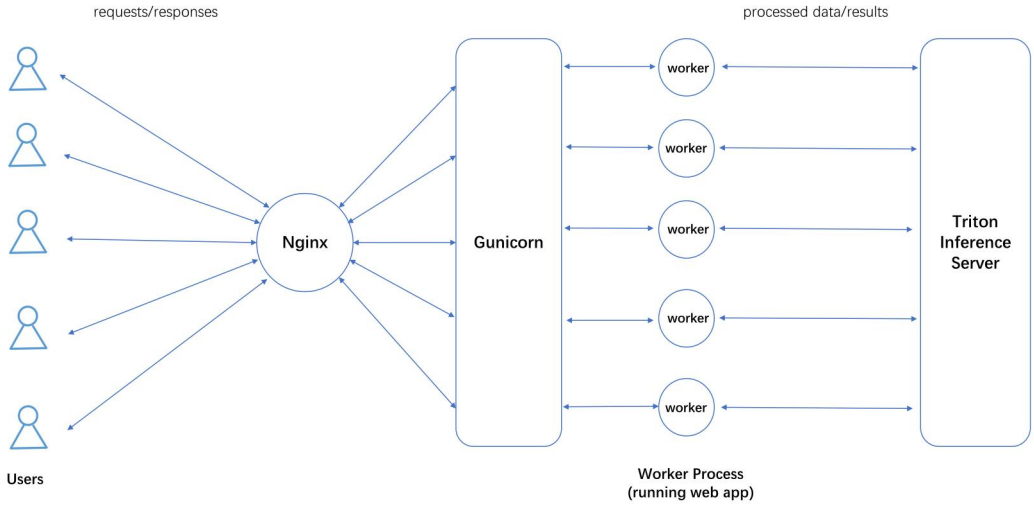

Figure 5.1: The structure of the request handler. Arrows are bidirectional representing the workflow of handling requests and returning responses respectively.

During system operation, Nginx acts as an intermediate proxy to receive requests from clients and return responses from the backend. It manages large numbers of concurrent requests and responses in an asynchronous and efficient way, avoiding the server's computational resources being occupied by communication tasks. The web app is based on the Python Flask framework, which defines how to handle a single user request. In this system, the app is responsible for data processing and communication with Triton. Gunicorn is used to implement concurrent processing of requests. To be precise, gunicorn creates multiple worker processes to run app at the same time and dynamically manages them according to the system's running conditions, and when multiple requests arrive, it assigns the requests to worker processes.

5.3 Neural Network Deployment

Triton Inference Server

Triton Inference Server (hereafter referred to as Triton) is a service framework developed by

NVIDIA for AI application deployment. Triton integrates various functions including model management, inference engine management, and resource scheduling to enable developers to deploy AI applications in a convenient and efficient way. Some of the key features of Triton used in this project are described below.

Model Management

Triton provides a convenient way to organize AI models for developers. They can create a repository with a specific structure to manage multiple models and different versions of models, and write configuration files to specify the inference characteristics of the models at runtime. The backend of this project contains three neural network models, Yolo, FastPose and VideoPose3D, which are running simultaneously to perform a complete inference process from video to 3D pose sequences.

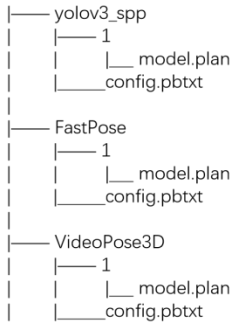

Figure 5.2: An example of Triton repository. The plan file describes a model's inference based on TensorRT, and the pbtxt file specifies the model's running configuration.

Accelerate Inference Based on TensorRT

Triton integrates with common inference engines, including TensorRT and OnnxRuntime. Developers can specify which engine to use for inference in the configuration file. TensorRT is a high-performance deep learning inference engine from NVIDIA that enables model inference acceleration through deep optimization of network structures and data streams, and methods for optimization include graph optimization, layer fusion and dynamic memory management. In this project, the Yolo and FastPose models run on TensorRT to provide inference services. To do this, firstly convert the a model's weight file obtained in the training process to onnx format. (Open Neural Network Exchange, a unified format specification for neural network models). The onnx file is then compiled into the .plan or .engine format required by TensorRT, which is the object code generated by TensorRT for a specific network structure in a specific GPU environment. (Note that the VideoPose3D model runs on onnxruntime engine as it contains indexing operations, which is not currently supported by TensorRT.)

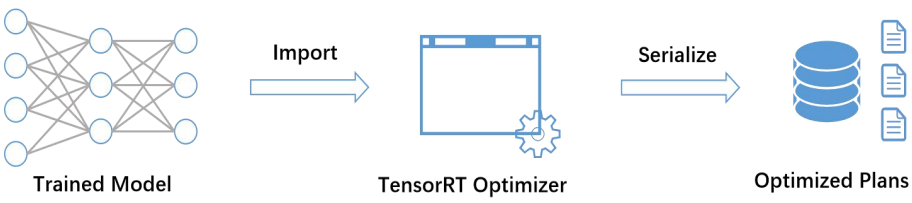

Figure 5.3(a): Workflow of converting trained model to TensorRT format.

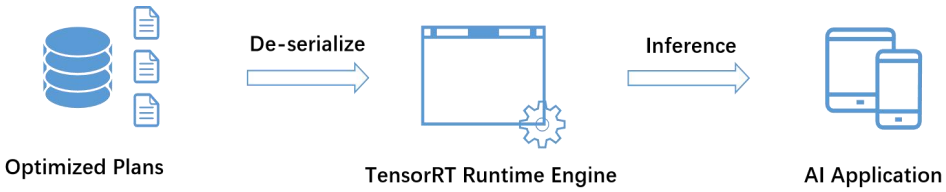

Figure 5.3(b): TensorRT in runtime.

Figure 5.3: An illustration of how TensorRT works before and in runtime.

Multiple Model Instances

Triton allows developers to start multiple instances of a model to perform parallel inferences in order to handle a large number of requests simultaneously. This could be done by simply specifying the number of instances in the configuration file as long as sufficient hardware resources are given. Such a feature of Triton helps build a system designed for high concurrency cases.

Communicate using gRPC

Triton provides a communication interface based on the gRPC protocol, which allows external applications to communicate with Triton server. In this project, requests handling and inference services are designed to be two indepent modules. The Request Handler receive requests from clients and pre-process data to make it compatible with the input of the neural network, while the Triton side is responsible for inference. During runtime, the Python Flask side sends the pre-processed data to the Triton side to perform inference and receives the results from Triton for post-processing. All communication between the two sides is based on the gRPC interface.

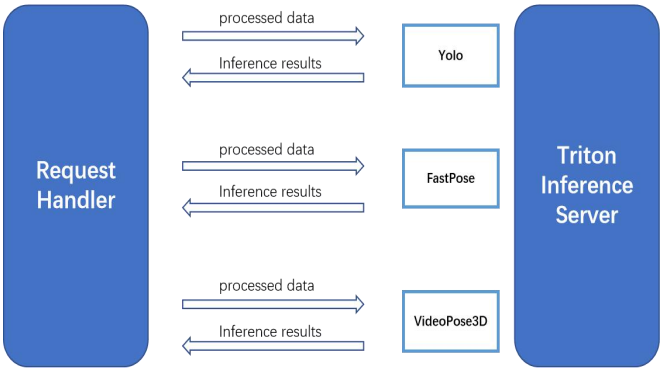

Figure 5.4: Communication between Request Handler and Triton when performing the whole process of estimating 3D poses from the input video.

5.4 Running environment

The running environment of the deployed system is shown in table 5.1.

GPU	NVIDIA RTX A5000 (24GB)
CPU	Intel Xeon platinum 8163 (32 cores)
RAM	64GB
OS	Ubuntu 20.04
CUDA	11.7
Triton Inference Server	2.25.0

Table 5.1: Running environment.

References

[1] Fang, H. S., Xie, S., Tai, Y. W., & Lu, C. (2017). Rmpe: Regional multi-person pose estimation. In Proceedings of the IEEE international conference on computer vision (pp. 2334-2343).

[2] Pavllo, D., Feichtenhofer, C., Grangier, D., & Auli, M. (2019). 3d human pose estimation in video with temporal convolutions and semi-supervised training. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 7753-7762).

[3] Lee, K., Lee, I., & Lee, S. (2018). Propagating lstm: 3d pose estimation based on joint interdependency. In Proceedings of the European conference on computer vision (ECCV) (pp. 119-135).

[4] Pavlakos, G., Zhou, X., Derpanis, K. G., & Daniilidis, K. (2017). Coarse-to-fine volumetric prediction for single-image 3D human pose. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7025-7034).

[5] Martinez, J., Hossain, R., Romero, J., & Little, J. J. (2017). A simple yet effective baseline for 3d human pose estimation. In Proceedings of the IEEE international conference on computer vision (pp. 2640-2649).

[6] Ionescu, C., Papava, D., Olaru, V., & Sminchisescu, C. (2013). Human3. 6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. IEEE transactions on pattern analysis and machine intelligence, 36(7), 1325-1339.

[7] Sigal, L., & Black, M. J. (2006). Humaneva: Synchronized video and motion capture dataset for evaluation of articulated human motion. Brown Univertsity TR, 120(2).

[8] Sigal, L., Balan, A. O., & Black, M. J. (2010). Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. International journal of computer vision, 87(1), 4-27.

[9] Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

[10] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Appendix

A.1 Detailed structure of the graph attention module in experiment 1

Figure A.1 shows the internal structure of the graph attention module and an illustration of forward propagation. The module takes an input of a feature sequence and output a new feature sequence. The blue box identifies an operation on the variables, the light blue identifies the intermediate output , and the orange identifies the learnable parameters. Each variable is labeled with a shape. For the sake of simplicity, only the case with an input of single batch is given here, and only one pair of features (i.e., $[h_i, h_j]$ in Figure A.1) is shown in the diagram of forward propagation.

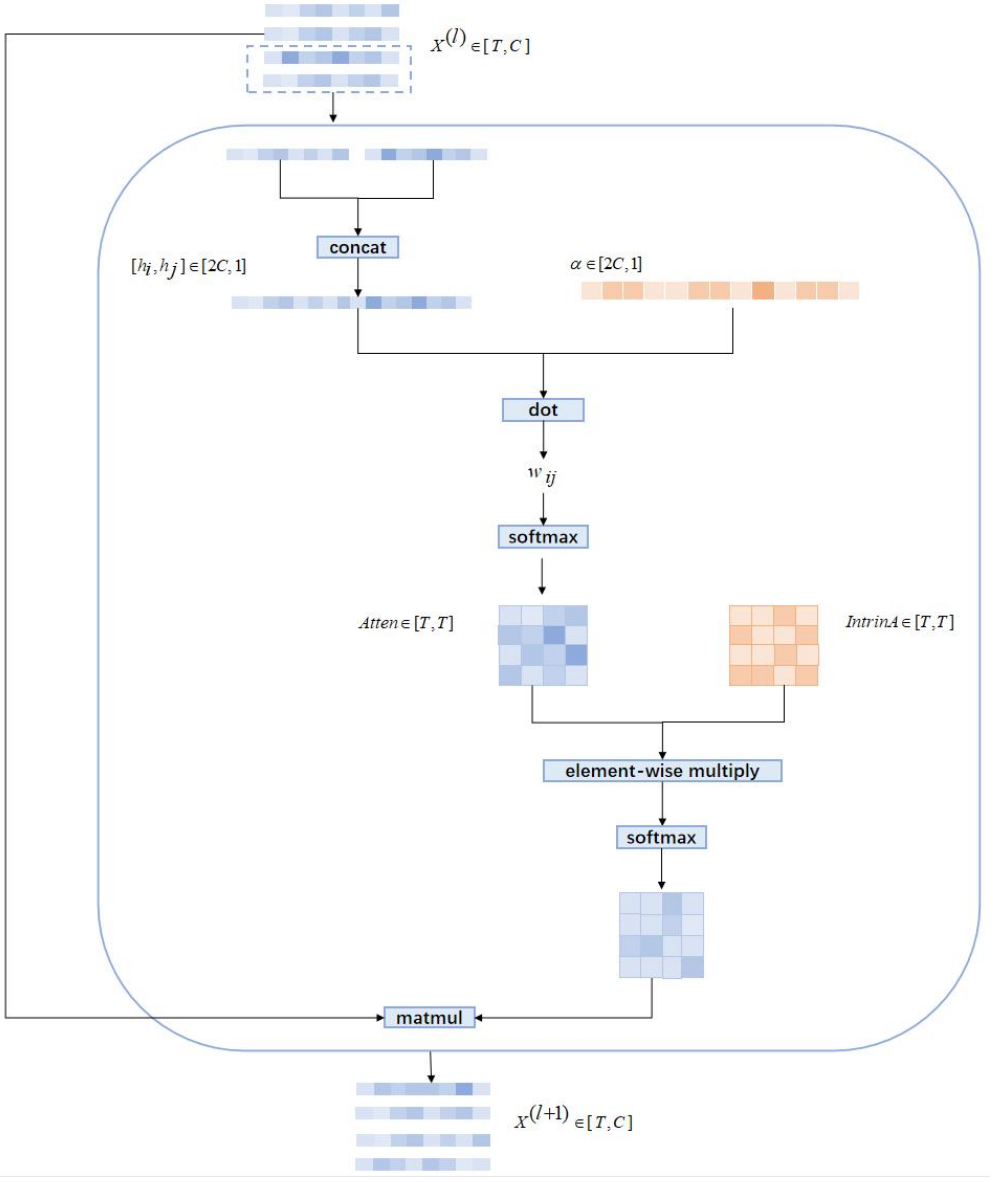

Figure A.1: Detailed structure of graph attention module and forward propagation in experiment 1.

Where T denotes the length of the sequence, C denotes the channel numbers of the feature vector. The variables are described in section 3.4.

A.2 Detailed structure of the graph attention module in experiment 2

Figure A.2 shows the internal structure of the graph attention module and an illustration of forward propagation. The overall structure is similar to that shown in Figure A.1, while the difference is that this module operates on the two-dimensional feature for each frame, and an additional constant matrix A is added. For the sake of simplicity, only one frame is selected to illustrate forward propagation.

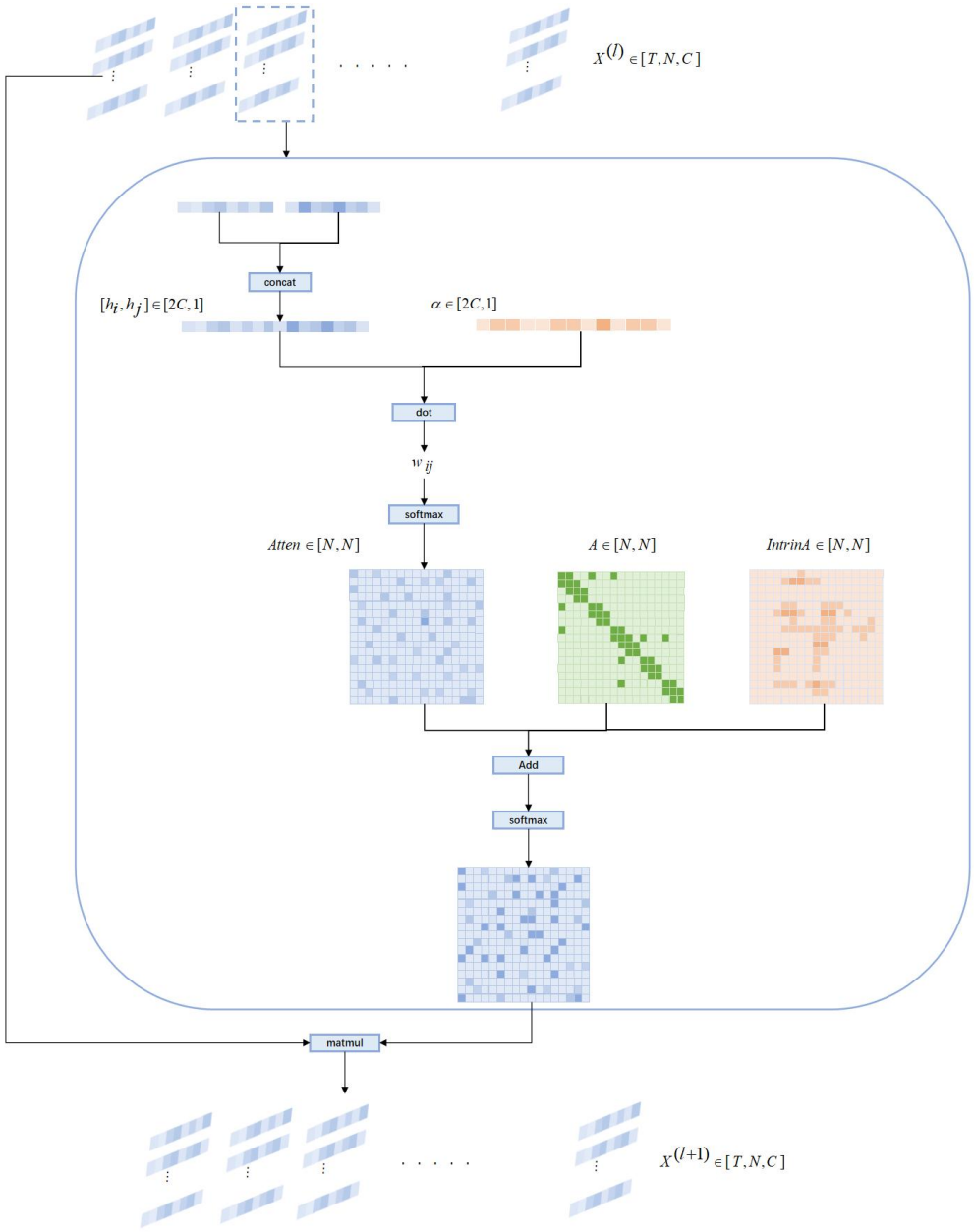

Figure A.2: Detailed structure of GCNAttention module and forward propagation in experiment 2.

Where T denotes the length of the sequence, C denotes the channel numbers of the feature vector, and N represents the number of joints in a skeleton. The variables are described in section 3.5.

A.3 Define local coordinate system for joints

As described in section 4.4, defining local coordinate systems for all joints is the precondition for calculating local rotation and generating the BVH file. A detailed description of defining local systems is given here.

Joint Name	forward	up	left
Hip	$(P_{LH}-P_S)\times(P_{RH}-P_S)$	$forward \times(P_{RH}-P_{LH})$	$up \times forward$
RightHip	$(P_{RH}-P_{RK})\times(P_{Hip}-P_{RH})$	$P_{RH}-P_{RK}$	$up \times forward$
RightKnee	$(P_{RH}-P_{RK})\times(P_K-P_{RA})\times up$	$P_{RK}-P_{RA}$	$up \times forward$
RightAnkle	$(P_{RH}-P_{RK})\times(P_K-P_{RA})\times up$	$P_{RK}-P_{RA}$	$up \times forward$
LeftHip	$(P_{Hip}-P_{LH})\times(P_{LH}-P_{LK})$	$P_{LH}-P_{LK}$	$up \times forward$
LeftKnee	$(P_{LH}-P_{LK})\times(P_{LK}-P_{LA})\times up$	$P_{LK}-P_{LA}$	$up \times forward$
LeftAnkle	$(P_{LH}-P_{LK})\times(P_{LK}-P_{LA})\times up$	$P_{LK}-P_{LA}$	$up \times forward$
Spine	$(P_{RS}-P_S)\times(P_{LS}-P_S)$	P_T-P_S	$up \times forward$
Chest	$(P_{RS}-P_S)\times(P_{LS}-P_S)$	P_T-P_S	$up \times forward$
Thorax	$(P_{LS}-P_N)\times(P_{RS}-P_N)$	$forward \times(P_{RS}-P_{LS})$	$up \times forward$
Neck	$(P_{LS}-P_N)\times(P_{RS}-P_N)$	$forward \times(P_{RS}-P_{LS})$	$up \times forward$
Head	$(P_{LS}-P_N)\times(P_{RS}-P_N)$	$forward \times(P_{RS}-P_{LS})$	$up \times forward$
LeftShoulder	$(P_{LS}-P_{LE})\times up$	$(P_T-P_{LS})\times(P_{LS}-P_{LE})$	$up \times forward$
LeftUpArm	$(P_{LS}-P_{LE})\times up$	$(P_T-P_{LS})\times(P_{LS}-P_{LE})$	$up \times forward$
LeftElbow	$(P_{LE}-P_{LW})\times up$	$(P_{LS}-P_{LE})\times(P_{LE}-P_{LW})$	$up \times forward$
LeftWrist	$(P_{LE}-P_{LW})\times up$	$(P_{LS}-P_{LE})\times(P_{LE}-P_{LW})$	$up \times forward$
RightShoulde	$(P_{RE}-P_{RS})\times up$	$(P_{RS}-P_{RE})\times(P_T-P_{RS})$	$up \times forward$
RightUpArm	$(P_{RE}-P_{RS})\times up$	$(P_{RS}-P_{RE})\times(P_T-P_{RS})$	$up \times forward$
RightElbow	$(P_{RW}-P_{RE})\times up$	$(P_{RE}-P_{RW})\times(P_{RS}-P_{RE})$	$up \times forward$
RightWrist	$(P_{RW}-P_{RE})\times up$	$(P_{RE}-P_{RW})\times(P_{RS}-P_{RE})$	$up \times forward$

Table A.1(a): Definition of the local coordinate system for joints.

Abbreviation	Full name
Hip	Hip
RH	RightHip
RK	RightKnee
RA	RightAnkle
LH	LeftHip
LK	LeftKnee
LA	LeftAnkle
S	Spine
T	Thorax
N	Neck
Hd	Head
LS	LeftShoulder
LE	LeftElbow
LW	LeftWrist
RS	RightShoulder
RE	RightElbow
RW	RightWrist

Table A.1(b): Full names for the abbreviations in Table A.1(a).

P_i in the table denotes the 3D coordinates of a joint output by the pose estimation model, and $\times$ denotes the cross product of two vectors. A generic way of representing a coordinate system is used here, i.e., a tuple of three vectors forward, up and left. In the right-hand system, the three vectors correspond to the y-axis, z-axis and x-axis respectively.

In order to provide more freedom for users to edit character poses using the generated BVH file, three additional joints, Chest, LeftUpArm and RightUpArm, are added to the 17-joint skeleton used in VideoPose3D. These three joints are not included in the output of the VideoPose3D model, and their local coordinate system is simply set to be consistent with their parent joints.