PUMPS: Skeleton-Agnostic Point-based Universal Motion Pre-Training for Synthesis in Human Motion Tasks

Supplementary Material

A. Additional Fine-tuning Demonstrations

A.1. Real-world validation: Monocular motion capture learning

Since our main experiments purely centred around 3D annotations, we seek to demonstrate here the effectiveness of PUMPS with regards to handling real-world data. Specifically, we observe the importance of motion synthesis pre-training for the video-to-skeletal motion task, i.e. monocular motion capture. We approach this task in an end-to-end manner for the purposes of validating real world data, using a ViT-L/16 image encoder [1] to extract frame-wise image features as input conditions, followed by our PUMPS Φ^{LMS} architecture to produce the motion capture prediction. The Human3.6M dataset is used for both training and testing.

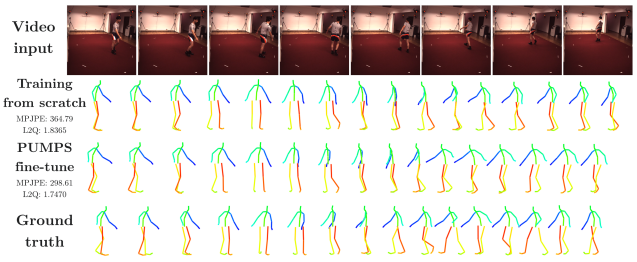

Figure 1. Motion capture results produced by a PUMPS model trained from scratch, (top skeletal row) compared to a fine-tuning process from pre-trained weights (middle skeletal row).

From Fig. 1, we can observe a clear improvement in motion capture predictions with a fine-tuning approach. The incompleteness of monocular video conditions (i.e. occlusion and lack of depth data) places the burden of understanding natural motion behaviours on the synthesis model. Given this notion, the provision of this understanding through pre-trained weights significantly improves the MPJPE of the resulting model, which is reflected visually through the closer adherence to the ground truth's body position. L2Q rotational error also observes an improvement, though mostly through positional improvements, as roll rotation understanding of the pre-trained PUMPS model remains quite limited due to its obscure presence in TPCs.

A.2. Semantic learnability of latent space: Text-to-motion diffusion

Our exploration of our model's motion synthesis capabilities has thus far been limited to spatially defined conditions. As such, we seek to complete our analysis by demonstrating our model's ability to facilitate motion synthesis from non-spatial inputs. Using semantic text conditions, we train a new *latent diffusion* model based on FLAME [4], with the objective of reconstructing latent features produced by our PUMPS encoder Φ^{enc}. To train this model, we use text-motion pairs from the HumanML3D dataset [3], which contains text annotations for a subset of AMASS data.

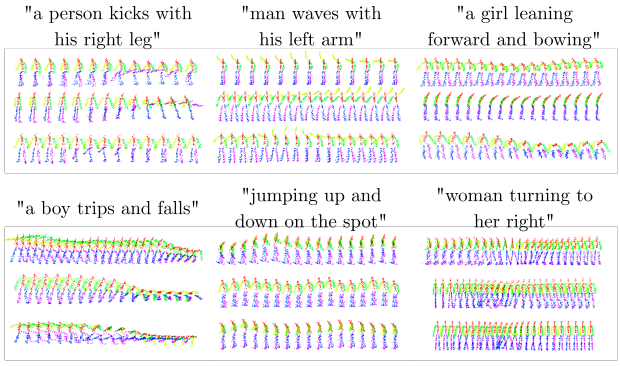

Figure 2. Diffusion-based text-to-motion samples produced by our PUMPS-based FLAME checkpoint.

The example evaluations produced by the resulting FLAME diffuser model show a clear pairing between human action descriptions and TPC representations of motions. Crucially, despite the unstructured nature of point clouds, our latent space provides an adequate foundation for transforming TPCs to produce diverse, human-like, and conditionally adherent pose sequences, comparable to behaviours exhibited on fixed skeletal structures. In future work, we seek to devise approaches for extracting 3D point cloud-based pose data from video representations of real-world motions, which would enable the possibility of incorporating much larger video annotation datasets for PUMPS-based text-to-motion learning.

B. Algorithmic Definitions

B.1. Quaternion-based Representations and Forward Kinematics

A quaternion $\mathbf{q} = [w, x, y, z]$ consists of a scalar part w and a vector part $[x, y, z]$. When normalised, quaternions provide a stable and compact representation for 3D rotations.

Rotating a 3D coordinate $\mathbf{p}$ using a unit quaternion $\mathbf{q}$ is performed through quaternion conjugation:

$$\text{rotate}(\mathbf{p}, \mathbf{q}) = \mathbf{q}\mathbf{p}\mathbf{q}^{-1},$$

where $\mathbf{p}$ is treated as a pure quaternion $[0, \mathbf{p}]$, and $\mathbf{q}^{-1}$ is the quaternion inverse, which for unit quaternions is simply the conjugate: $\mathbf{q}^{-1} = [w, -x, -y, -z]$ if $|\mathbf{q}| = 1$.

A skeleton comprises a tree hierarchy of joints, where each non-root joint j has a parent joint k. Using forward kinematics, the position and orientation of each joint in world coordinates can be computed by propagating transformations through the skeletal hierarchy.

Each joint j has a local rotation quaternion $\mathbf{q}_j$, which defines its orientation relative to its parent joint. The global rotation $\mathbf{q}_j^{\text{global}}$ of the joint is computed recursively as:

$$\mathbf{q}_j^{\text{global}} = \mathbf{q}_k^{\text{global}} \cdot \mathbf{q}_j,$$

where $\mathbf{q}_k^{\text{global}}$ is the global rotation of the parent joint. For the root joint, the global rotation is simply $\mathbf{q}_{\text{root}}^{\text{global}} = \mathbf{q}_{\text{root}}$.

Each joint also has a local translation $\mathbf{t}_j = [t_x, t_y, t_z]$, representing its offset relative to its parent joint. The global position $\mathbf{p}_j^{\text{global}}$ of the joint is computed recursively by applying the parent joint's global rotation to the local translation, then adding the parent's global position:

$$\mathbf{p}_j^{\text{global}} = \mathbf{p}_k^{\text{global}} + \text{rotate}(\mathbf{t}_j, \mathbf{q}_k^{\text{global}}),$$

where $\mathbf{q}_{\text{parent}(j)}^{\text{global}}$ rotates the local translation into the global coordinate system. This is the **forward kinematics** equation. For the root joint, $\mathbf{p}_{\text{root}}^{\text{global}} = \mathbf{t}_{\text{root}}$.

Finally, to extend joint definitions into *bones*, we simply provide a joint j with a 3D tail position $\mathbf{t}_j^{\text{tail}}$, which is defined relatively to $\mathbf{p}_j^{\text{global}}$. The joint tail's global position can be obtained as if the joint head, i.e. $\mathbf{p}_j^{\text{global}}$, was the parent of the tail position:

$$\mathbf{p}_j^{\text{tail}} = \mathbf{p}_j^{\text{global}} + \text{rotate}(\mathbf{t}_j^{\text{tail}}, \mathbf{q}_j^{\text{global}}),$$

This allows us to define the bone form of j as a line segment $[\mathbf{p}_j^{\text{global}}, \mathbf{p}_j^{\text{tail}}]$, with a rotation of $\mathbf{q}_j^{\text{global}}$. **For clarity, j will be referring to its bone form from this point on.**

B.2. Skeleton to Temporal Point Cloud

We follow the same TPC sampling method as described in Mo et al. [5]. The method involves defining a set of points within the proximate volume of each bone of the skeleton, and obtaining the global positions of each point via forward kinematics based on the skeleton's motion.

To produce any given point for the TPC from a skeleton, we define a local translation of a point p relative to the head of its associated bone j. First, we select a position along the bone's line segment using $\alpha \sim \mathcal{U}_{[0,1]}$, where $\alpha = 0$ and $\alpha = 1$ correspond to the head and tail positions respectively. Next, we offset this position using a random 3D vector $\beta \in \mathbb{R}^3$ from a Normal distribution of $\mathcal{N}(\mu = 0, \sigma \in [0.025, 0.075])$. Formally, the local translation of point p is sampled as:

$$\mathbf{t}_p = \alpha \mathbf{t}_j^{\text{tail}} + \beta$$

The global position of p is evaluated in the same forward kinematics manner as the joint tail's global position:

$$\mathbf{p}_p = \mathbf{p}_j^{\text{global}} + \text{rotate}(\mathbf{t}_p, \mathbf{q}_j^{\text{global}}),$$

The temporal point cloud $\mathcal{P}$ should be expected to represent the human body with a generally even density of points, in order to successfully obfuscate the original skeletal hierarchy and homogenise motion data. To achieve this, the bone that a given point $p \in \mathcal{P}$ is associated with should be selected by a weighted random sampler, where the line segment length of each bone determines the probability of the bone. This allows larger bones to be represented with more points, and vice versa, thus creating an even distribution of points. Formally, given the set of all bones $\mathcal{J}$, the probability of selecting a given bone j is:

$$P(j) = \frac{|\mathbf{p}_j^{\text{tail}}|}{\sum_{j' \in \mathcal{J}} |\mathbf{p}_{j'}^{\text{tail}}|}$$

B.3. Interpolating with Classical Methods

In our paper, we used both linear interpolation (LERP) and spherical linear interpolation (SLERP) as both interpolation benchmarks, as well as a correction step for the TCP-to-skeletal pose conversion processes performed by PC-MRL.

To explain these aspects, let us first denote that the t-th pose of a motion x_t for a skeleton of $|\mathcal{J}|$ joints contains a 3D root position part $x_t^{\text{root}} \in \mathbb{R}^3$ and a set of quaternions part $x_t^{\text{quat}} \in \mathbb{H}^{|\mathcal{J}|}$, where $\mathbb{H}$ is the set of all quaternions. As such, we can describe the composition of x_t as $x_t = x_t^{\text{root}} \oplus x_t^{\text{quat}}$. To simplify explanations, assume quaternion conjugations between two **quaternion sets** apply the conjugation to all corresponding pairs of quaternions.

B.3.1. Motion Keyframe Interpolation with LERP and SLERP

LERP represents a constant trajectory between two scalar values over a period of time. This is commonly used for interpolating Euclidean properties such as positions, but is also generally compatible with spherical properties like quaternions, albeit with a normalisation process to remain spherical. SLERP, on the other hand, is only applied to spherical properties, and aims to represent the same constrant trajectory as LERP, but in spherical space.

To apply LERP between two poses x_a and x_b (assuming $b > a$), we first perform an element-wise subtraction to find the total Euclidean difference. Then, we divide this total difference by the number of intervals between x_a and x_b (i.e. frames, defined by $b - a$). Finally, we cumulatively increment x_a with the interval difference. Formally,

to produce the LERP-interpolated motion $\text{LERP}(x_a, x_b) = \{x'_{a+1}, x'_{a+2}..., x'_{b-2}, x'_{b-1}\}$, we can define each x'_{a+t} as:

$$x'_{a+t} = x_a + t\frac{x_b - x_a}{b - a}$$

As a post-processing step for quaternions, each quaternion is normalised to a norm of 1 in order to retain its spherical representation.

With SLERP, the interpolation of root positions and quaternions are applied separately. Since root positions are a Euclidean property, the LERP strategy is still used for the root positions, i.e. $\text{LERP}(x_a^{\text{root}}, x_b^{\text{root}})$. For quaternions, we employ spherical interpolation between x_a^{quat} and x_b^{quat}. Formally, to define the interpolated quaternions $\text{SLERP\_Q}(x_a^{\text{quat}}, x_b^{\text{quat}}) = \{x'_{a+1}{}^{\text{quat}}, x'_{a+2}{}^{\text{quat}}, ..., x'_{b-2}{}^{\text{quat}}, x'_{b-1}{}^{\text{quat}}\}$, we compute each $x'_{a+t}{}^{\text{quat}}$ using the following quaternion conjugation:

$$x'_{a+t}{}^{\text{quat}} = x_a^{\text{quat}}(x_a^{\text{quat}-1} x_b^{\text{quat}})^t$$

The final SLERP interpolation between two poses x_a and x_b is defined as:

$$\begin{aligned}\text{SLERP}(x_a^{\text{quat}}, x_b^{\text{quat}}) = \text{LERP}(x_a^{\text{root}}, x_b^{\text{root}}) \oplus \\ \text{SLERP\_Q}(x_a^{\text{quat}}, x_b^{\text{quat}})\end{aligned}$$

B.3.2. Interpolation-based Correction for PC-MRL

The TCP-to-skeletal pose conversion process via PC-MRL [5] is approximate and imperfect over a long motion sequence $\mathcal{X} = \{x_0, x_1, ..., x_{|\mathcal{X}|-1}\}$, due to the relative space of rotational representations in PC-MRL. Since the input keyframes $\mathcal{X}' = \{x_{k_0}, x_{k_1}, \dots\} \subset \mathcal{X}$ in a motion keyframe interpolation problem are known poses, we can gauge the error of the predicted PC-MRL pose $\mathcal{Y}' = \{y_{k_0}, y_{k_1}, \dots\}$ and the input pose to determine the amount of correction $\mathcal{E} = \{\epsilon_0, \epsilon_1, ..., \epsilon_{|\mathcal{X}|-1}\}$ needed for anchoring the resulting motion to the input keyframes. Assume $\{k_0, k_1, ...\}$ is an ordered list.

To find $\mathcal{E}$, we initially measure the keyframe-wise error $\mathcal{E}' = \{\epsilon_{k_0}, \epsilon_{k_1}, \dots\}$. For each real and predicted keyframe pair $x_k \in \mathcal{X}$ and $y_k \in \mathcal{Y}$, the error is measured differently for the 3D root positions and the quaternion set. Specifically, the root position error ϵ_k^{root} is obtained by a simple subtraction:

$$\epsilon_k^{\text{root}} = y_k^{\text{root}} - x_k^{\text{root}},$$

while the quaternion errors ϵ_k^{quat} are obtained using an inverse quaternion conjugation:

$$\epsilon_k^{\text{quat}} = y_k^{\text{quat}-1} x_k^{\text{quat}}$$

As usual, we can declare $\epsilon_k = \epsilon_k^{\text{root}} \oplus \epsilon_k^{\text{quat}}$. We can then infer the full correction sequence $\mathcal{E}$ as a SLERP interpolation between each adjacent error pair in $\mathcal{E}'$. We formally define the full correction sequence $\mathcal{E}$ as:

$$\mathcal{E} = \mathcal{E}' \cup \{\cup_{\epsilon_{k_a} \in \mathcal{E}} \text{SLERP}(\epsilon_{k_a}, \epsilon_{k_{a+1}})\}$$

Finally, we can apply the correction to the original PC-MRL prediction, creating a corrected sequence $\mathcal{C} = \{c_0, c_1, ..., c_{|\mathcal{X}|-1}\}$. For the root position, the correction is applied via a simple addition:

$$c_t^{\text{root}} = y_t^{\text{root}} + \epsilon_t^{\text{root}},$$

while quaternion corrections are applied via a conjugation:

$$c_t^{\text{quat}} = y_k^{\text{quat}} \epsilon_k^{\text{quat}}$$

Finally, $c_t = c_t^{\text{root}} \oplus c_t^{\text{quat}}$.

B.4. Linear Assignment for Point Cloud Matching Using the Hungarian Algorithm

In this section, we detail the linear assignment approach for matching two point clouds, $\mathcal{P}_x = \{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{|\mathcal{P}|-1}\}$ and $\mathcal{P}_y = \{\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_{|\mathcal{P}|-1}\}$, in our objective function $\mathcal{L}_{\text{rec}}$, by minimising the sum of squared Euclidean distances between paired points. The Hungarian algorithm is employed to find the optimal assignment efficiently. The goal is to find a bijective mapping $\sigma : \{0, 1, \dots, |\mathcal{P}| - 1\} \to \{0, 1, \dots, |\mathcal{P}| - 1\}$ that minimises the total cost:

$$\min_{\sigma} \sum_{i=1}^{|\mathcal{P}|} \|\mathbf{x}_i - \mathbf{y}_{\sigma(i)}\|^2.$$

This problem can be formulated as a linear assignment problem where the cost matrix $\Delta = [\delta_{gh}] \in \mathbb{R}^{|\mathcal{P}| \times |\mathcal{P}|}$ is defined by:

$$\delta_{gh} = \|\mathbf{x}_g - \mathbf{y}_h\|^2.$$

The Hungarian algorithm solves the assignment problem in $O(|\mathcal{P}|^3)$ time [2] and consists of the following steps:

1. **Construct the Cost Matrix:** Compute the distance matrix Δ where each entry δ_{gh} represents the squared distance between points $\mathbf{x}_g$ and $\mathbf{y}_h$.
2. **Row Reduction:** For each row of Δ, subtract the smallest element from every element in that row.

$$\Delta'_{gh} = \Delta_{gh} - \min_h \Delta_{gh}, \quad \forall g.$$

3. **Column Reduction:** For each column of the reduced matrix Δ', subtract the smallest element from every element in that column.

$$\Delta''_{gh} = \Delta'_{gh} - \min_g \Delta'_{gh}, \quad \forall h.$$

4. **Cover All Zeros with Minimum Number of Rows and Columns:** Determine the minimum number of rows and columns required to cover all zero entries in Δ''. We can achieve this with the following method (note that there are many alternative iterative approaches):

4.1 Iterate through each row, and **star** the first zero that doesn't share a column with an existing starred zero.
4.2 **Cover all columns** containing a starred zero. If all zeros are covered, go to step 5.
4.3 Find any uncovered zero in Δ'' and **mark** it.
4.3.a If the zero shares a row with an existing starred zero, **cover the row and uncover the column** previously covering the starred zero. Repeat from step 4.3 with the new coverage.
4.3.b Otherwise, we create a path. Starting from the uncovered zero, alternate between travelling to a **starred zero in the same column** and to a **marked zero in the same row**. Stop when there are no available zeros to travel to. Among the traversed zeros, **un-star all starred zeros**, and **star all marked zeros**. Repeat from step 4.2 with the new starred zeros.

5. **Check for Optimality:**
   - If the number of covering lines equals $|\mathcal{P}|$, an optimal assignment is defined by the matrix coordinates of the starred zeros, for which there should be exactly $|\mathcal{P}|$ of.
   - Otherwise, proceed to adjust the matrix and repeat the process.
6. **Adjust the Matrix:**
   (a) Find the smallest uncovered element m in Δ''.
   (b) Subtract m from all uncovered elements.
   (c) Add m to all elements covered by both a horizontal and a vertical line.

   Return to step 4 until the number of covering lines equals $|\mathcal{P}|$.

References

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Georg Minderer, Matthias andHeigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*. International Conference on Learning Representations, 2021. 1

[2] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972. 3

[3] Chuan Guo, Shihao Zou, Xinxin Zuo, Sen Wang, Wei Ji, Xingyu Li, and Li Cheng. Generating diverse and natural 3d human motions from text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5152–5161, 2022. 1

[4] Jihoon Kim, Jiseob Kim, and Sungjoon Choi. FLAME: Free-form language-based motion synthesis & editing. In *AAAI*, pages 8255–8263, 2023. 1

[5] Clinton Mo, Kun Hu, Chengjiang Long, Dong Yuan, and Zhiyong Wang. Motion keyframe interpolation for any human skeleton via temporally consistent point cloud sampling and reconstruction. In *ECCV*, pages 159–175. Springer, 2024. 2, 3