Motion Matching for Responsive Animation for Digital Humans

Longteng Duan* Guo Han* Boxiang Rong* Hang Yin*
ETH Zürich
{loduan, guohan, borong, hanyin}@student.ethz.ch

Abstract

Motion matching is an approach to generate character animation in a controlled manner by blending and transitioning between pre-recorded animation sequences multiple times per second. It is widely used in the game industry to generate more natural and dynamic animations that respond to the character's environment and input in real-time. However, despite its popularity, there is no official open-source implementation available and the primary inputs are limited, only keyboard or game controller. We believe that these methods are not immersive enough for the players. In this project, we implemented a functional motion matching pipeline and introduced multiple alternative input methods, including drawn trajectories and real-time captured human poses, to enhance the immersive experience for video game players. Meanwhile, we did ablation study to verify the effectiveness of several components in our pipeline. You can find our project in this repo and our demo videos on youtube.

1. Introduction

Motion matching serves as a straightforward yet powerful tool for character animation that align with a user's command input, and is a widely-used motion generation technique in the game industry. Motivated by enabling users to have an immersive experience when interacting with the digital character, we want to go beyond normal keyboard control and explore more interactive methods between digital characters' animation and human actions. To initiate our exploration here, we begin with motion matching. Our target is to enhance the immersive user experience within this framework by investigating diverse input possibilities.

To be more specific, this project aims to develop a motion matching pipeline from scratch using the provided code base. The initial focus is on implementing a responsive motion matching pipeline that effectively utilizes basic user input, i.e. using keyboard, for controlling the digital character's movements. Further exploration will involve incorporating additional input, like drawn trajectories and human pose, to enhance the user's immersive experience.

For the pipeline implementation, despite motion matching being a well-established pipeline in the industry, there is currently no official open-source implementation available. Thus, implementing the entire pipeline involves addressing numerous challenges and fine-tuning parameters through trial and error.

For input diversity exploration, in addition to keyboard control, we aspire to incorporate higher-level interactive input to further enhance the user experience. Here, we focus on the user's own movement to create a mapping between real-world human motion and digital character motion.

Finally, we integrate all our functionalities and create a captivating video game-like demonstration.

In conclusion, our contributions include:

- Developed a traditional motion matching pipeline from scratch within the provided code base, that can do stand-still, dance, walk, sprint, jump, creep, etc. for responsive animation of a general digital human skeleton.
- Explored various input possibilities to control the skeleton human, including keyboard, drawn trajectory and real-time captured poses.
- Created video game-like using experience.

2. Related Work

2.1. Motions in Video Games

In the game industry, numerous techniques are employed for motion generation. In [20], the authors introduced the utilization of motion capture in video games. This technique involves capturing the movements of real individuals and employing the recorded data to animate digital characters. Typically, motion capture entails placing sensors on an actor's body and recording their movements while they perform various actions. Subsequently, the data is processed by software to generate a digital representation of the actor's

*These authors contributed equally to this work

motions. Presently, motion capture remains extensively employed in the game industry [12, 16, 21].

Motion capture is a powerful technique; however, its drawback lies in the fact that it only captures fixed motions, resulting in limited scalability. Another technique, known as procedural animation, can also be employed in video games [6, 12]. This approach utilizes algorithms to generate animations dynamically, without relying on prerecorded motion capture data. Procedural animation enables the creation of more dynamic and diverse animations. In contrast to motion capture, there exist various methods for generating procedural animations [6, 8, 12, 18].

Due to the complex and stochastic nature of human motion, motion capture currently remains superior to other methods for generating motions in video games [12], despite its high memory and time requirements.

2.2. State Machine

In general, a state machine, specifically a finite state machine, is a computational model used to control the behavior of a system [24]. It divides the system's behavior into different states and defines the transition conditions and actions between states. Compared to a naive system that reacts to inputs instantly, a state machine is able to handle collisions between the current state and inputs and transition to a new state in a controllable way [22].

In the game industry, state machines are commonly used to control the behavior and actions of characters. Each state represents a different behavior of the character, such as standing, walking, running, attacking, etc. By defining transition conditions between states, character behavior and reactions can be implemented and switched between different states. Additionally, state machines can also be used to manage the states and progress of game levels, with each state representing a different state of the level, such as preparation, gameplay, victory, failure, etc. [13].

Using state machines clarifies the relationships between different states and behaviors, making it easier to extend new logic. However, state machines also have their limitations. Programmers have to predefine all the possible states and transitions in advance. As the variety and number of game characters increase, the complexity of the state machine can exponentially increase.

2.3. Motion Matching

Motion matching is a cutting-edge technique widely used in the game industry for generating character motion. The essence of motion matching is a simple yet powerful idea: conducting a search within a motion capture dataset to find the best match that aligns with the current context.

Since its initial introduction in [7], the motion matching algorithm has undergone significant development, emerging as a mature pipeline applicable to various usage scenarios. Notably, it has found success in close character interactions for fighting sports [11]. Moreover, motion synthesis has extended beyond basic movements like walking or running, encompassing complex actions such as parkour or climbing [5].

To address issues of poor scalability and large memory requirements inherent in the traditional motion matching pipeline, an elegant solution called learned motion matching [15] has been proposed. This innovative approach combines the strengths of motion matching and neural networks by replacing the main components of the motion matching pipeline — projection, stepping, and decompression — with three MLPs. This strategy effectively reduces memory usage and improves scalability without compromising its original performance.

Despite the emergence of generative motion synthesis models [17, 23], motion matching continues to be a popular choice in the industry, thanks to its notable advantages, including flexibility, predictability, low processing time, and high visual quality [15].

3. Method

3.1. LAFAN1 Motion Capture Dataset

The LAFAN1 Dataset from Ubisoft [12] was used as the motion capture dataset in this project. The dataset consists of a collection of 77 motion sequences captured from 5 subjects, which was categorized into 15 themes such as walking, dancing, aiming, etc. There are 496,672 motion frames in total , played at a frame rate of 30 frames per second. In addition to the high quality of the motion capture data, it is worth noting a few undocumented details for the sake of completeness.

3.1.1 Euler Angle Order

In contrast to the distinctive Euler angle order of $Z \rightarrow X \rightarrow Y$ found in typical BVH files [1], the BVH files included in the LAFAN1 dataset follow a different but more canonical order of $Z \rightarrow Y \rightarrow X$. This divergence must be considered when parsing and loading the motion data to avoid erroneous poses. In order to handle both conventional BVH files and those from LAFAN1, the order should be recorded during parsing. The order is indicated by the labels of $*$*rotation* where $*$=$\{$X,Y,Z$\}$ following the `CHANNELS` keyword. For example, when loading LAFAN1, the joints' relative rotation quaternion were calculated as

$$q_j = q_z(\psi) * q_y(\theta) * q_x(\phi)$$

where $q_{\{x,y,z\}}(\cdot)$ denotes the elementary rotation around the corresponding axis.

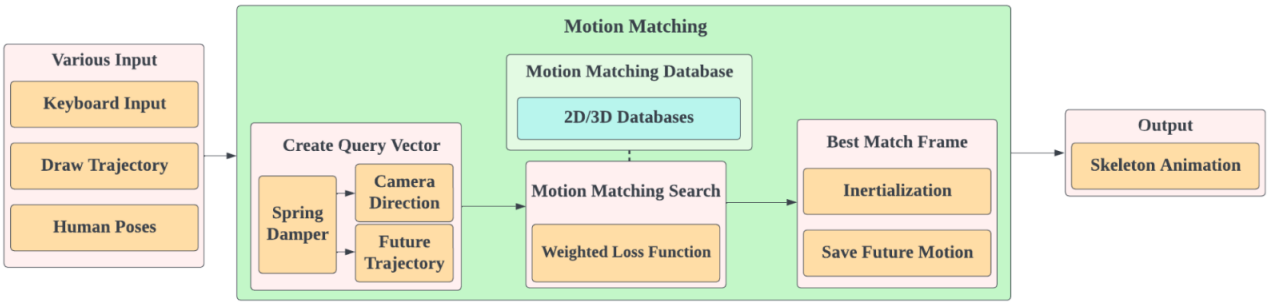

Figure 1. **Pipeline of Our Motion Matching System.** Our system takes three kinds of input, keyboard, drawing trajectories, and human poses. When generating a query feature vector, we use the spring damper method to predict the future trajectory. During the search process, we emphasize more on fitting the future trajectory. Once find the best match, we do inertialization for smooth transformation between motions. The resulted pose information is then applied on the digital character as output.

3.1.2 Character Root Orientation

Another notable characteristic of the LAFAN dataset is that it uses $\hat{y} = [0\,1\,0]$ as the forward direction of the character, as opposed to the x-axis convention. This is particularly important when extracting feature vectors within the motion matching pipeline. Furthermore, note that the codebase used in this project adopts a y-axis-upward world frame convention. If not adapted, the x-axis of the root frame will eventually point vertically, resulting in unstable forward direction projected onto the ground.

3.2. Basic Motion Matching Pipeline

3.2.1 Overview

Our pipeline is illustrated in Fig. 1. The animation system based on motion matching comprises three main components: projection (motion matching search), stepping (incrementing the frame index), and decompression (pose lookup).

The system takes the current animation context and user control query as input and generates continuous, natural animation output. Initially, the raw user inputs are converted into query features. During the projection phase, a best-fit feature vector is obtained by performing a nearest neighbor search in the matching database. The frame index corresponding to the best fit replaces the current frame index, serving as the starting point for the subsequent animation sequence playback.

At each step, the index is advanced, and the associated pose is retrieved from the animation database. This pose is then applied to the controlled digital character, ensuring smooth and synchronized animation.

3.2.2 Matching Database Construction

The matching database $\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{n-1}]$ contains feature vectors of all frames in provided motion clips and is used for best matched motion search.

The feature vector $\mathbf{x}$ is defined in a concise but representative way, describing the features to be matched. In more detail, according to [15], $\mathbf{x} = \left\{\mathbf{t}^t, \mathbf{t}^d, \mathbf{f}^t, \dot{\mathbf{f}}^t, \dot{\mathbf{h}}^t\right\} \in \mathbb{R}^{27}$.

All the features need to be converted to the character's local coordinate frame. The derivation of local coordinate frame is detailed in Eq.(1). Here, $\mathbf{v}_{\text{facing}}, \mathbf{v}_{\text{up}}, \mathbf{v}_{\text{side}}$ are the coordinate axes of the character's local frame, which are then concatenated into $\mathbf{R}_{\text{local}} \in \mathbb{R}^{3\times3}$. $\mathbf{R}_{\text{root}}$ represents the character's root orientation, with $\mathbf{R}_{\text{root}}(:,1)$ denoting its second column.

$$\begin{aligned}
\mathbf{v}_{\text{up}} &= [0 \quad 1 \quad 0]^\intercal \\
\mathbf{v}_{\text{facing}} &= -\frac{\mathbf{R}_{\text{root}}(:,1) - \langle \mathbf{R}_{\text{root}}(:,1), \mathbf{v}_{\text{up}} \rangle \cdot \mathbf{v}_{\text{up}}}{\left\| \mathbf{R}_{\text{root}}(:,1) - \langle \mathbf{R}_{\text{root}}(:,1), \mathbf{v}_{\text{up}} \rangle \cdot \mathbf{v}_{\text{up}} \right\|} \\
\mathbf{v}_{\text{side}} &= \mathbf{v}_{\text{facing}} \times \mathbf{v}_{\text{up}} \\
\mathbf{R}_{\text{local}} &= [\mathbf{v}_{\text{facing}} \quad \mathbf{v}_{\text{up}} \quad \mathbf{v}_{\text{side}}]
\end{aligned} \tag{1}$$

$\mathbf{t}^t \in \mathbb{R}^6$ and $\mathbf{t}^d \in \mathbb{R}^6$ encode trajectory information. They store the 2D future trajectory positions projected on the ground and future trajectory orientations for the future 20, 40, 60 frames respectively. They are calculated as Eq.(2) shows, where m represents $\{20, 40, 60\}$. Below, $\mathbf{t}.x$ and $\mathbf{t}.z$ denote the x and z coordinates of the vector $\mathbf{t}$, respectively.

$$\begin{aligned}
\mathbf{t}_{\text{m}}^{t_{\text{raw}}} &= \mathbf{R}_{\text{local}}^T \cdot \overrightarrow{(\mathbf{p}_{\text{rootPosNow}}, \mathbf{p}_{\text{rootPosAfter-m}})} \\
\mathbf{t}_{\text{m}}^{t} &= \left[\mathbf{t}_{\text{m}}^{t_{\text{raw}}}.x, \mathbf{t}_{\text{m}}^{t_{\text{raw}}}.z\right] \\
\mathbf{t}_{\text{m}}^{d_{\text{raw}}} &= \mathbf{R}_{\text{local}}^T \cdot \mathbf{v}_{\text{rootVelAfter-m}} \\
\mathbf{t}_{\text{m}}^{d} &= \frac{\left[\mathbf{t}_{\text{m}}^{d_{\text{raw}}}.x, \mathbf{t}_{\text{m}}^{d_{\text{raw}}}.z\right]}{\left\| \left[\mathbf{t}_{\text{m}}^{d_{\text{raw}}}.x, \mathbf{t}_{\text{m}}^{d_{\text{raw}}}.z\right] \right\|}
\end{aligned} \tag{2}$$

The other features describe current animation context and state. $\mathbf{f}^t \in \mathbb{R}^6$ and $\dot{\mathbf{f}}^t \in \mathbb{R}^6$ are two foot joint positions and velocities of the current frame respectively, while $\dot{\mathbf{h}}^t \in \mathbb{R}^3$ is the hip joint velocity. Their derivations are shown in Eq.(3)

After organizing all feature vectors $\{\mathbf{x}_i\}_{i=0}^{N-1}$ into the matching database, each feature is normalized for magnitude integrity.

$$\begin{aligned}\mathbf{f}^t_{\text{left/right}} &= \mathbf{R}^T_{\text{local}} \cdot \overrightarrow{(\mathbf{p}_{\text{rootPosNow}}, \mathbf{p}_{\text{footPosNow}})} \\ \dot{\mathbf{f}}^t_{\text{left/right}} &= \mathbf{R}^T_{\text{local}} \cdot \mathbf{v}_{\text{footVelNow}} \\ \dot{\mathbf{h}}^t &= \mathbf{R}^T_{\text{local}} \cdot \mathbf{v}_{\text{hipVelNow}}\end{aligned} \tag{3}$$

Extension for 3D Motion Matching. Before in [15], all trajectory features were limited to 2D, which resulted in the inability to account for motion occurring along the vertical axis, such as jumping and creeping. To address this limitation in our application, we have enhanced the trajectory features to include 3D information. Consequently, the updated feature vector, denoted as $\mathbf{x}$, now consists of $\left\{\mathbf{t}^t, \mathbf{t}^d, \mathbf{f}^t, \dot{\mathbf{f}}^t, \dot{\mathbf{h}}^t\right\} \in \mathbb{R}^{33}$ with $\mathbf{t}^t \in \mathbb{R}^9$ and $\mathbf{t}^d \in \mathbb{R}^9$. To implement this enhancement, we have included the y component when calculating trajectory features.

3.2.3 Trajectory Prediction

Every time we receive a new input, we want the character to react smoothly to the change. Simple algorithms, such as interpolating between the current position and target positions, can generate smooth trajectories by adding extra parameters to control the interpolation rate. However, they fail in cases of fast changes, as the velocity on these trajectories is discontinued [14].

Therefore, we utilize a spring-damper system to compute the character's future trajectories. First, let's consider Hooke's Law as shown in Eq.(4).

$$F = -kx = ma \Rightarrow a + \frac{k}{m}x = 0 \tag{4}$$

$$x(t) = x_0 cos(\omega t) + \frac{v_0}{\omega} sin(\omega t),\ \omega = \sqrt{\frac{k}{m}} \tag{5}$$

Here, a is the second derivative of x. By solving this equation, we got solution in Eq.(5). Suppose the solution is the moving function of a game character on one dimension, we will observe the character doing simple harmonic motion around a point. To make it stop at the target point, we have to add an extra force opposite to the velocity to make it stop after a certain number of oscillations. Then Eq. (6) shows the Hooke's Law for damped spring. And $-cv$ is the extra force, where c is a parameter controlling the dampe level.

$$F = -kx - cv = ma \Rightarrow a + \frac{c}{m}v + \frac{k}{m}x = 0 \tag{6}$$

Solving Eq.(6), we get solution Eq.(7).

$$x(t) = e^{st}, s = -\omega(\zeta \pm \sqrt{\zeta^2 - 1}) \tag{7}$$

$$\text{where } \omega = \sqrt{\frac{k}{m}},\ \zeta = \frac{c}{2\sqrt{km}} \tag{8}$$

When $0 \leq \zeta \leq 1$, the system is a underdamped spring; $\zeta = 1$ is the Critical Spring Damper and $\zeta > 1$ is the overdamped spring [14]. In games, we generally use critical spring damper. The simplified solution is shown by Eq.(9), which will stop at the target place without oscillating.

$$x(t) = (x_0 + (v_0 + \omega x_0)t)e^{-\omega t} \tag{9}$$

By further modify x and v in Eq.(6) into $x_{goal} - x_0$ and $v_{goal} - v_0$, and solve again, we can get more general expressions Eq.(10).

$$\begin{aligned} x(t) &= (j_0 + j_1 t)e^{-yt} + c \\ c &= x_{goal} + \frac{d}{v_{goal}} \\ y &= \frac{d}{2} \\ j_0 &= x_0 - c \\ j_1 &= v_0 + j_0 y \end{aligned} \tag{10}$$

d is a parameter to control the decaying rate. Instead of target position, we can only get target velocity from controller. Therefore, x stands for character velocity and v stands for acceleration. Using Eq.(11), we finally compute future trajectory positions [14].

$$x(t) = \int (j_0 e^{-yt} + j_1 t e^{-yt} + c)dt \tag{11}$$

3.2.4 Best Motion Search - Weighted Loss

In [15], the best motion search is finding the index k in the motion matching database that minimizes the squared Euclidean distance to the query vector.

$$k^* = \arg\min_k \|\mathbf{x} - \mathbf{x}_k\|^2 \tag{12}$$

In [15], the author claimed that the features in the query vector do not need to be weighted. However, in our experiments, we found that when the features are not weighted, the matched animation does not agree with the user input. To mitigate this issue, we place more weight on the predicted trajectory during the search process because it is generated based on the user input. In the search process, we divide the feature vector into $\mathbf{x}_{trajectory} = \left\{\mathbf{t}^t, \mathbf{t}^d\right\}$ and $\mathbf{x}_{feature} = \left\{\mathbf{f}^t, \dot{\mathbf{f}}^t, \dot{\mathbf{h}}^t\right\}$. And we search for

$$\begin{aligned} k^* = \arg\min_k\ & 0.8 \left\|\mathbf{x}_{trajectory} - \mathbf{x}_{k\_trajectory}\right\|^2 \\ & + 0.2 \left\|\mathbf{x}_{feature} - \mathbf{x}_{k\_feature}\right\|^2 \end{aligned} \tag{13}$$

3.2.5 Inertialization

The inertialization technique [4] can be understood as a quintic interpolation with specific constraints which aims to interpolate motion between two poses, resembling physical inertial.

Scalar Case Without loss of generality, the ending value x_1 at time $t = t_1$ can be set to 0. Moreover $x_0 \geqslant 0$ as it corresponds to the norm of the difference vector between positions. Given inputs x_0, x_1, t_1, v_0 where v_0 is the estimated initial velocity, we are seeking the interpolation without overshoot:

$$\begin{aligned} &\tilde{x}(t) = c_5 t^5 + c_4 t^4 + c_3 t^3 + c_2 t^2 + v_0 t + x_0 \quad t \in [0, t_1] \\ &\text{s.t.} \quad 0 < x(t) < x_0 \quad \forall t \in (0, t_1) \end{aligned} \tag{14}$$

The overshoot-free constraint leads to clamping of v_0

$$v_0 \leftarrow \min\{v_0, 0\} \tag{15}$$

Assume the steady state at $t = t_1$:

$$\tilde{x}(t), \tilde{x}'(t), \tilde{x}''(t)|_{t=t_1} = 0 \tag{16}$$

yielding a solution of $c_{5:3}$ in 14 parametrized by c_2.

$$\begin{aligned} c_5 &= -\frac{c_2 {t_1}^2 + 3v_0 t_1 + 6x_0}{{t_1}^5} \\ c_4 &= \frac{3c_2 {t_1}^2 + 8v_0 t_1 + 15x_0}{{t_1}^4} \\ c_3 &= -\frac{3c_2 {t_1}^2 + 6v_0 t_1 + 10x_0}{{t_1}^3} \end{aligned} \tag{17}$$

Notice that $c_2 = 1/2a_0$ where a_0 is the initial acceleration in [4]. In order to find c_2, we may set $\tilde{x}''' = 0$ at $t = t_1$. Substituting $c_{5:3}$ in 14 with 17 yields:

$$c_2 = -\frac{10x_0 + 4t_1 v_0}{{t_1}^2} \tag{18}$$

Next, c_2 and t_1 were tuned to guarantee the overshoot-free behavior. A second order constraint was imposed:

$$c_2 \leftarrow \min\{c_2, 0\} \tag{19}$$

Finally, the interpolation time t_1 is verified so that no overshoot below 0 may occur. The concavity was realized by placing the inflection point outside the interval. The result can be derived as:

$$t_1 \leftarrow \min\{t_1, \frac{5x_0}{|v_0|}\} \tag{20}$$

which is indeed the result in [4]. As Eq.20 merely depends on x_0, v_0, it can be imposed before proceeding with the calculation.

Euclidean Vector Case Full dimensional quintic interpolation in Euclidean space can be complicated [10]. As a reasonable approximation, only the norm of the difference between states was inertialized, i.e. $x_0 = \|\mathbf{x}_0 - \mathbf{x}_1\|$ while the unit direction $\bar{\mathbf{x}}_{01} = (\mathbf{x}_0 - \mathbf{x}_1)/\|\mathbf{x}_0 - \mathbf{x}_1\|$ remained constant during interpolation:

$$\tilde{\mathbf{x}}(t) = \mathbf{x_1} + \tilde{x}(t) \cdot \bar{\mathbf{x}}_{01} \tag{21}$$

The initial velocity v_0 can be calculated by finite difference:

$$v_0 \approx \frac{x0 - (\mathbf{x}_{-1} - \mathbf{x}_1) \cdot \bar{\mathbf{x}}_{01}}{\Delta t} \tag{22}$$

Quaternion Case The direction and norm between quaternions was defined by axis-angle representation. Since q and $-q$ represent the same rotation [3], the shortest rotation must be selected:

$$\begin{aligned} &\tilde{q}(t) = q\left(\cos\left(\tilde{\alpha}(t)/2\right), \sin\left(\tilde{\alpha}(t)/2\right)\mathbf{v}_{01}\right) * q_1 \\ &\text{where} \quad \alpha_0 = \text{Angle}(q_0 * q_1^{-1}),\ \mathbf{v}_{01} = \text{Axis}(q_0 * q_1^{-1}) \\ &\text{and} \quad \text{Angle}(q) := 2\cos^{-1}\left(\text{Re}(\bar{q})\right), \text{Axis}(q) := \frac{\text{Im}(\bar{q})}{\|\text{Im}(\bar{q})\|} \\ &\bar{q} = \text{Reduce}(q) := \begin{cases} q & \text{if } \text{Re}(q) \geqslant 0 \\ -q & \text{otherwise} \end{cases} \end{aligned} \tag{23}$$

The initial velocity v_0 can be estimated by the finite difference of the twist component around v_{01} [25]:

$$v_0 \approx \frac{\alpha_0 - 2\tan^{-1}\left(\frac{\text{Im}(q_{-1} * q_1^{-1}) \cdot \mathbf{v}_{01}}{\text{Re}(q_{-1} * q_1^{-1})}\right)}{\Delta t} \tag{24}$$

3.2.6 Accelerated Search with AABB Tree

As shown in [15], the time-consuming nearest neighbor search can be accelerated with simple two-layer Axis-Aligned Bounding Box (AABB), assuming similarities of neighboring frames.

Fitting AABB Fitting an AABB to a set of geometry objects set S including other AABBs is relatively simple. The lower/upper bound lb, ub are defined as $lb_i = \inf_{x \in S} x_i, ub_i = \sup_{x \in S} x_i$. As stated in [15], a 2-layer hierarchy comprising 16 and 64 consecutive frames is sufficient.

Search the AABB Tree The distance between the query point $\hat{x}$ and an AABB provides a lower bound estimate for that between $\hat{x}$ and any object within that AABB:

$$\min_{x \in \text{AABB}} \|x - \hat{x}\| \geqslant \text{dist}_{\text{AABB}}(x) \tag{25}$$

Therefore whenever the distance against an AABB to be searched is greater than the current lowest distance, no further searching into this AABB will be needed. It can be shown that $\mathrm{dist}(\cdot)$ can be calculated in an efficient manner thanks to the decoupled box constraints:

$$\mathrm{dist}_{\mathrm{AABB}}(x) = \sum_{i}^{D} \left(\max\{lb_i - x_i, 0, x_i - ub_i\}\right)^2 \quad (26)$$

3.3. Advanced Motions

3.3.1 3D Motion

In Section 3.2.2, we discussed the weakness of the feature vector mentioned in [15] and introduced an updated feature vector that includes the y component in the future trajectory. However, we discovered that using only the updated feature vector for motion matching resulted in improper matching of the original 2D motions. To address this issue, we divided the database into 2D and 3D motions. Whenever the character performs a jump or creep, we switch from the 2D database to the 3D database and conduct motion matching within the 3D database. Once the motion is completed, we switch back to the 2D database.

When creating the trajectory for jumping, the character's facing direction remains unchanged, and we only modify the y component of the character velocity and goal velocity. Firstly, we assign the character an upward y component velocity, denoted as v^y. Consequently, we set the y component of the goal velocity to $-v^y$. In 2D motions, we exclusively utilize spring damper in the x and z directions. However, in 3D motions, we additionally employ spring damper in the y direction to generate a smooth jumping trajectory.

When creeping, the character's facing direction is controlled by the user's input direction. When creating the trajectory for creeping, the goal velocity is always set to 0.8, which closely approximates the average creeping speed in the database. It is important to note that, for creeping, we do not assign v^y to the current speed and goal speed. Instead, we set the height of the trajectory to 0.4, which is near the average hip height of creeping in the database.

3.3.2 Fixed Animation

The fixed animation strategy is implemented to handle actions that pose challenges in generating smooth and natural results through motion matching techniques. This approach is particularly useful for stand-still and dance animations.

In the case of stand-still animation, when the projected future velocity falls below a certain threshold and no input is detected, motion matching is halted, and a pre-selected stand-still animation is played. Once the animation clip reaches its conclusion, the character maintains the position from the last frame of the clip.

For the dancing animation, the initial frame is chosen based on the motion matching result. Subsequently, motion matching is temporarily paused, and the animation clip is played starting from the matched frame. Upon completion of the clip, another round of motion matching occurs, and this process iterates as needed.

3.4. Input Diversity Exploration

3.4.1 Keyboard Input

The keyboard serves as the primary control medium for human-computer interaction, making it an ideal starting point for our exploration of input diversity.

In our system, we assign specific functions to various keys to control the movement and actions of the digital character. The up and down arrow keys are utilized to adjust the target speed value. The "W, S, A, D" keys are responsible for controlling the character's movement in different directions. Pressing the "Shift" key enables the character to sprint, while pressing "J" triggers a jump action. The "P" key initiates a dance animation, and the "C" key prompts the character to creep.

These interaction commands are implemented within the "keyPressed" and "keyReleased" functions, allowing for real-time response.

3.4.2 Drawn Trajectory Control

Keyboard is limited for direction control, which only has 8 fixed velociy directions. And the players have to keep pressing keyboard to keep character moving. Sometimes we may just want the character to follow specific route and set our fingers free.

In the "draw Trajectory" mode, we can use mouse to paint on the ground. The painted lines are represented by sequences of dots, stored in "hitPoints" variable. Once paint a trajectory, keyboard control will be disabled, while you can still press "shift" to run. Then the character will follow a straight line to the nearest dot on the trajectory, and start following the whole trajectory. Once reaching the end, it will go straight back to the start point and do another round.

3.4.3 Human Pose Control

To provide users with a more intuitive and immersive experience, we have also implemented human pose control, which takes real-time human poses as input and enables smooth control of the character.

We utilized FastPose [9] to capture human poses. For the 13 motions, namely *creep forward*, *creep left*, *creep right*, *dance*, *jump*, *punch*, *run forward*, *run left*, *run right*, *stand still*, *walk forward*, *walk left*, and *walk right*, we prerecorded 13 videos as the reference for the corresponding poses. Subsequently, for each video, we iterated through its

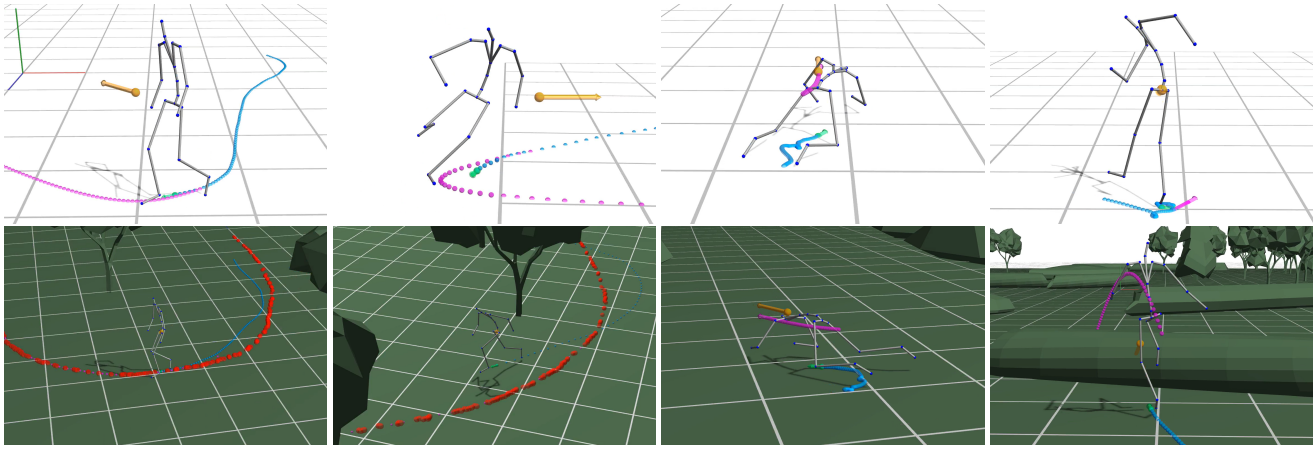

Figure 2. **Screenshots of Our Demos** [2]. **First row (left to right)**: walk, run, creep, dance; **Second row (left to right)**: walk, run with drawn trajectory, creep, jump in forest terrain. In the pictures, yellow arrow represents expected direction from input, blue dots are the history trajectory and purple dots are the predicted future trajectory.

frames, and for each frame *i*, we concatenated the pose from frame *i* to *i+10* to create a labeled data point representing the respective motion. This processing step also needs to be performed for newly acquired data.

We employ a Python script to translate real-time human poses into keyboard input for program control. When a new pose is received, we apply a KNN classifier to assign a label to the pose. To prevent jittering, we maintain a record of recent human pose labels and only utilize the most frequently occurring label from the last five poses to trigger the corresponding keyboard input.

4. Experiments

4.1. Demo

Our demos [2] feature diverse terrains, including basic and forest environments. They showcase motion matching based on selected action themes from LAFAN1 [12] data, including Obstacles, Walk, Dance, Ground, Run, Jumps, and Sprint. Fig.2 displays representative screenshots of the demos. In our implemented application, the digital character's skeleton can be controlled via keyboard, drawn trajectory, and pose control inputs, enabling actions like standstill, dance, walk, sprint, jump, and creep.

4.2. Ablation Study

4.2.1 Spring Damper

To smoothly transfer a variable from current value to target value, the simplest way is to interpolate in between. By controlling the interpolation rate, we can make it converge faster or slower. However, when new input comes in, target value changes instantly, which results in the discontinuity of first derivative [14], and applying interpolation here is inappropriate. On the other, spring damper uses *cos* based functions, which make it infinitely differentiable. So we get smooth trajectories, in a physical-alike manner, which helps to match more natural motions. The effectiveness of spring damper is shown vividly in Fig.3

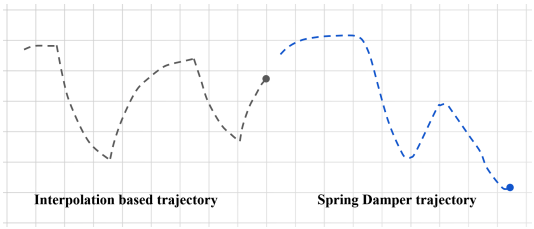

Figure 3. **Trajectory Comparison.** Spring damper generates smoother and more physical-alike trajectories, than interpolation does.

4.2.2 Weighted Loss

We compared motion matching before and after adding weights to the loss function. With more weights put on future trajectory, the character is able to focus on the red line and walk in a straight line as shown in Fig.4. While for unweighted loss, character is more likely to do matching based on behavior similarity, and ignore the moving direction.

4.2.3 Inertialization

Fig.5 demonstrates the difference between with and without inertialization. Without inertialization, pose switches are abrupt, resulting in odd behaviors. In contrast, inertialization generates smooth transformations by interpolating between current and matched poses, providing a seamless motion transition.

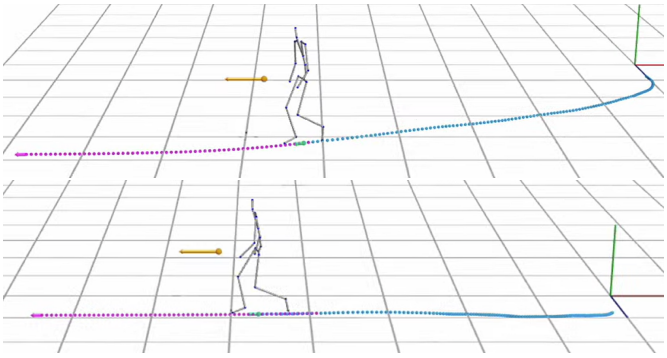

Figure 4. **Original Loss (*Upper*) and weighted Loss (*Lower*).** Character with weighted loss walk straightly to the left, while character with original loss cannot follow strictly to our control given left command.

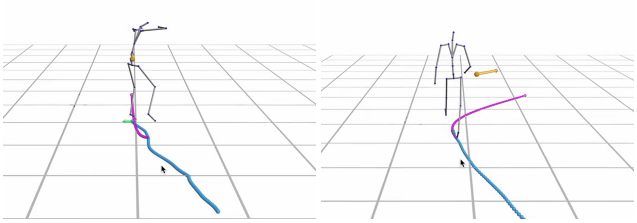

Figure 5. **Without Inertialization (*Left*) and with Inertialization (*Right*).** Character with Inertialization makes smoother transformation between motions as shown in the smoother history trajectory colored as blue, which also results in matching more natural behaviors.

4.3. Failure Cases

Including noisy data in the matching database can lead to motion matching failures. Specifically, when attempting the creeping motion and incorporating additional creeping data from the LAFAN1 dataset, the character initially creeps correctly. However, after a few frames, the character's creeping movement becomes uncontrollable, even when continuously pressing the 'C' key. To regain control over the character's creeping, the 'C' key needs to be released and pressed again.

Additionally, the pose classifier's failure can result in inaccurate motion matching. For example, distinguishing between "run right" and "walk right" actions poses a challenge to the kNN classifier, causing undesired jittering between the two motions. Besides, the absence of training data for action transitions also leads to unstable classification results during pose transitions, such as from 'dance' to 'jump'. Although attempts were made to improve the classifier's performance using a basic MLP, the limited training data and incomplete settings resulted in overfitting, making it ineffective for test videos.

5. Conclusion

In this project, we implemented a motion matching pipeline in C++ by building upon the provided codebase. The pipeline effectively integrates inputs from keyboard, drawn trajectory, and human pose, and produces responsive and smooth animation.

In the current stage, we formulate the interaction model between the human pose and digital character in the application as a straightforward classification problem. The available poses for controlling humans are predefined and limited. An optimal setup would involve mapping the actions performed by the human to the character. However, achieving this requires advanced techniques such as precise human pose detection and retargeting, which go beyond the scope of this project. Besides, the motion matching method has certain limitations. Firstly, it cannot generate motions that do not exist in the database. Additionally, parameter tuning is necessary to achieve smooth matching, and simple feature vectors may not be sufficient for complex behaviors. To address the limitations mentioned earlier, a promising solution is to combine the emerging generative models with the traditional motion matching pipeline, such as that described in [19]. This integration allows us to harness the strengths of both approaches, enhancing motion generation and expanding the capabilities of the motion matching process.

6. Contributions of Team Members

The workload of this project was distributed evenly among us. The table below highlights the assigned tasks and responsibilities of all team members.

Task	**Members**
1. Motion Matching: Database Construction	Guo Han
2. Motion Matching: Trajectory Prediction	Boxiang Rong
3. Motion Matching: Inertialization, AABB Search	Hang Yin
4. Motion Matching: Various Actions	Longteng Duan, Guo Han, Boxiang Rong
5. Motion Matching: Codebase Adaption and General Pipeline	All group members
6. Trajectory Control	Longteng Duan, Boxiang Rong
7. Human Pose Control	Longteng Duan, Guo Han, Boxiang Rong
8. Terrain Construction	Boxiang Rong

Table 1. Work split of team members.

References

[1] Biovision bvh. Lecture Note. https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html. 2

[2] Motion matching demos. https://youtube.com/playlist?list=PLUffCQyBEYtYXr-pVqqUgSG1Ncxp4UzAb. 7

[3] *Appendix D: Quaternions for Engineers*, pages 381–392. John Wiley Sons, Ltd, 2009. 5

[4] David Bollo. High performance animation in gears of war 4. In *ACM SIGGRAPH 2017 Talks*, SIGGRAPH '17, New York, NY, USA, 2017. Association for Computing Machinery. 5

[5] Michael Buttner. Machine learning for motion synthesis and character control in games. https://i3dsymposium.org/2019/keynotes/I3D2019_keynote_MichaelButtner.pdf, 2019. 2

[6] Alex Christo. Procedural creature generation and animation for games. 2022. 2

[7] Simon Clavet. Motion matching and the road to next-gen animation. https://www.gdcvault.com/play/1023280/Motion-Matching-and-The-Road, 2016. 2

[8] César Roberto de Souza, Adrien Gaidon, Yohann Cabon, Naila Murray, and Antonio Manuel López. Generating human action videos by coupling 3d game engines and probabilistic graphical models. *International Journal of Computer Vision*, 128(5):1505–1536, 2020. 2

[9] DrNoodle. Fastpose. https://drnoodle.github.io/fastpose_html/, 2022. [Accessed: 14-Jun-2023]. 6

[10] Tamar Flash and Neville Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. In *Journal of Neuroscience*, 1985. 5

[11] Geoff Harrower. Real player motion tech in 'ea sports ufc 3'. gdcvault.com/play/1025228/Real-Player-Motion-Tech-in, 2018. 2

[12] Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. Robust motion in-betweening. 39(4), 2020. 2, 7

[13] Joe Hocking. *Unity in Action: Multiplatform Game Development in C#*. Manning Publications, 2018. 2

[14] Daniel Holden. Spring-it-on: The game developer's spring-roll-call. https://theorangeduck.com/page/spring-roll-call. 4, 7

[15] Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. Learned motion matching. *ACM Trans. Graph.*, 39(4), aug 2020. 2, 3, 4, 5, 6

[16] Midori Kitagawa and Brian Windsor. *MoCap for artists: workflow and techniques for motion capture*. CRC Press, 2020. 2

[17] Peizhuo Li, Kfir Aberman, Zihan Zhang, Rana Hanocka, and Olga Sorkine-Hornung. GANimator. *ACM Transactions on Graphics*, 41(4):1–12, jul 2022. 2

[18] Quanzhou Li, Jingbo Wang, Chen Change Loy, and Bo Dai. Task-oriented human-object interactions generation with implicit neural representations. *arXiv preprint arXiv:2303.13129*, 2023. 2

[19] Weiyu Li, Xuelin Chen, Peizhuo Li, Olga Sorkine-Hornung, and Baoquan Chen. Example-based motion synthesis via generative motion matching, 2023. 8

[20] Alberto Menache. *Understanding motion capture for computer animation and video games*. Morgan kaufmann, 2000. 1

[21] Alberto Menache. *Understanding motion capture for computer animation*. Elsevier, 2011. 2

[22] Robert Nystrom. *Game Programming Patterns*. Genever Benning, 2014. 2

[23] Sigal Raab, Inbal Leibovitch, Guy Tevet, Moab Arar, Amit H. Bermano, and Daniel Cohen-Or. Single motion diffusion, 2023. 2

[24] Mark Shead. State machines in computer science. https://blog.markshead.com/869/state-machines-computer-science/. 2

[25] Ken Shoemake. *Fiber Bundle Twist Reduction*, page 230–236. Academic Press Professional, Inc., USA, 1994. 5