Learned Motion Matching

DANIEL HOLDEN, Ubisoft La Forge, Ubisoft, Canada
OUSSAMA KANOUN, Ubisoft La Forge, Ubisoft, Canada
MAKSYM PEREPICHKA, Concordia University, Canada
TIBERIU POPA, Concordia University, Canada

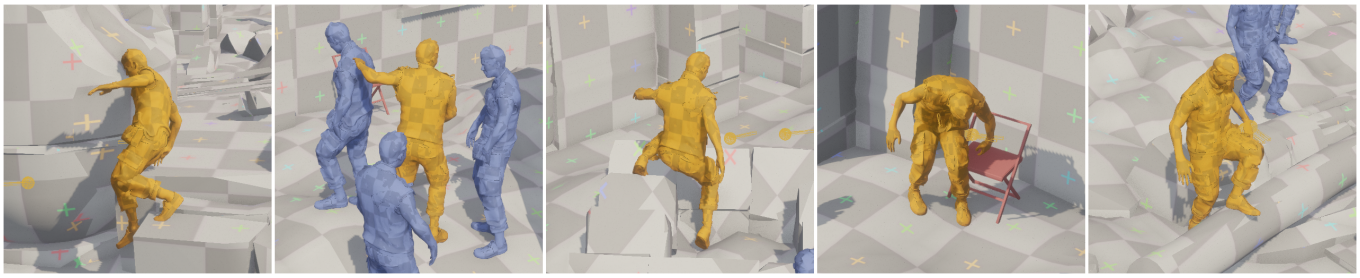

Fig. 1. Our method applied to various situations including navigating rough terrain, interaction with other characters, and using scene props.

In this paper we present a learned alternative to the Motion Matching algorithm which retains the positive properties of Motion Matching but additionally achieves the scalability of neural-network-based generative models. Although neural-network-based generative models for character animation are capable of learning expressive, compact controllers from vast amounts of animation data, methods such as Motion Matching still remain a popular choice in the games industry due to their flexibility, predictability, low pre-processing time, and visual quality - all properties which can sometimes be difficult to achieve with neural-network-based methods. Yet, unlike neural networks, the memory usage of such methods generally scales linearly with the amount of data used, resulting in a constant trade-off between the diversity of animation which can be produced and real world production budgets. In this work we combine the benefits of both approaches and, by breaking down the Motion Matching algorithm into its individual steps, show how learned, scalable alternatives can be used to replace each operation in turn. Our final model has no need to store animation data or additional matching meta-data in memory, meaning it scales as well as existing generative models. At the same time, we preserve the behavior of Motion Matching, retaining the quality, control, and quick iteration time which are so important in the industry.

CCS Concepts: • **Computing methodologies → Motion capture**.

Authors' addresses: Daniel Holden, Ubisoft La Forge, Ubisoft, 5505 St Laurent Blvd, Montreal, QC, H2T 1S6, Canada, daniel.holden@ubisoft.com; Oussama Kanoun, Ubisoft La Forge, Ubisoft, 5505 St Laurent Blvd, Montreal, QC, H2T 1S6, Canada, oussama.kanoun@ubisoft.com; Maksym Perepichka, maksym@perepichka.com, Concordia University, 1493 Saint-Catherine Street West, Montreal, Quebec, Canada, H3G 2W1; Tiberiu Popa, tiberiu.popa@concordia.ca, Concordia University, 1493 Saint-Catherine Street West, Montreal, Quebec, Canada, H3G 2W1.

0730-0301/2020/7-ART1 $15.00
https://doi.org/10.1145/3386569.3392440

Additional Key Words and Phrases: Motion Matching, Generative Models, Neural Networks, Character Animation, Animation,

ACM Reference Format:
Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. 2020. Learned Motion Matching. *ACM Trans. Graph.* 39, 4, Article 1 (July 2020), 13 pages. https://doi.org/10.1145/3386569.3392440

1 INTRODUCTION

In interactive applications such as video games, demand for larger, more immersive and dynamic worlds has steadily made it more difficult to produce characters which can respond realistically and naturally in the exponentially growing number of different situations that are presented to them. Meanwhile, the amount of data required has also slowly grown, and AAA video games now often contain tens of thousands of unique animations that all must be triggered in the correct context [Holden 2018].

Introduced by Clavet and Büttner [2015], *Motion Matching* is a method of searching a large database of animations for the animation which best fits the given context. This method has quickly been adopted by many studios due to its simplicity, flexibility, controllability, and the quality of the motion it produces [Büttner 2019; Clavet 2016; Harrower 2018; Hussain 2019; Zinno 2019]. Rather than specifying the fine-grained animation logic via a state-graph, Motion Matching allows animators to specify the properties of the animation which should be produced, and the best fitting match is selected automatically via a nearest neighbor search. When combined with large amounts of data, Motion Matching proves a simple and effective way of dealing with the vast number of possible transitions and interactions that are required by a modern AAA video game. Additionally, since Motion Matching plays back the animation data stored in the database as-is, with only simple blending and post-processing such as inverse kinematics applied, quality is generally preserved, animators retain a level of control, and the behaviour can be tracked and debugged with appropriate tools. Finally, since it has minimal training/pre-processing time, adjustments can often

be made in real-time, resulting in quick iteration time. The primary limitation of Motion Matching is that the memory usage (and runtime performance to some degree) scale linearly with the amount of data that is provided and the number of features to be matched. This results in a constant balance between the expressiveness of the system, the quality of the results, and the real-world memory and performance budgets. This leaves developers unable to combine it with powerful data processing methods such as automatic data augmentation.

Meanwhile, the academic community has seen an increasing interest in neural-network-based generative models of motion due to their low memory usage, scalability in terms of data, and fast runtime evaluation. Recent methods have shown that neural-network-based models can be effectively applied to generate realistic motion in a number of difficult cases including navigation over rough terrain [Holden et al. 2017], quadruped motion [Zhang et al. 2018], and interactions with the environment or other characters [Lee et al. 2018; Starke et al. 2019]. Yet, such models are often difficult to control, have unpredictable behaviour, long training times, and can produce animation of lower quality than that of the original training set [Büttner 2019; Zinno 2019].

In this paper we present a method which retains the positive aspects and behaviour of Motion Matching, but with the scalability of neural-network-based models. It works by breaking down the Motion Matching algorithm into individual components which are then replaced with learned, scalable alternatives. More specifically, we break the algorithm down into three specialized neural networks that can be used in unison, or in various specific combinations depending on the exact runtime, memory, and quality requirements of the controller.

In summary, our contribution is a learned alternative to the Motion Matching algorithm which replaces the three key stages of the algorithm with specialized neural networks, resulting in state-of-the-art animation generation results in terms of animation quality, runtime performance, and memory usage.

2 RELATED WORK

In this section we discuss previous work including research on data-driven animation synthesis, generative models for character animation, and motion-matching-based methods.

2.1 Data-Driven Animation Synthesis

There is a long history of data-driven animation synthesis in the animation community with a large variety of tools being used including graphs, linear methods, kernel-based methods, and most recently neural networks.

Graphs are a commonly used structure for controlling and generating animation. For unstructured data, transitions can be inserted at similar frames and the resulting graph structure searched to find animations which achieve particular goals using algorithms such as A* [Arikan and Forsyth 2002; Kovar et al. 2002; Lee et al. 2002; Min and Chai 2012; Safonova and Hodgins 2007]. Due to the use of a "matching cost", Motion Matching could be viewed as a special case of a graph-based search algorithm where the goal cost and pose cost are combined, transitions are made possible at regular intervals, and a greedy algorithm is used in traversal. Due to their simplicity and flexibility, many extensions to motion graphs are proposed, such as motion grammars [Hyun et al. 2016] which can be used to enforce rules in the graph traversal, and parametric motion graphs [Heck and Gleicher 2007; Shin and Oh 2006] which allow blending at nodes and transitions. To accelerate the search of motion graphs, various forms of pre-computation have been proposed including usage of a lookup table, functional approximation, or reinforcement learning (RL) [Lee and Lee 2004; Lo and Zwicker 2008; Treuille et al. 2007]. Notably, a graph like structure was extended to the continuous domain by Lee et al. [2010], who employed RL to decide how to best interpolate the ten nearest neighbors, and pull the synthesized animation toward valid configurations which achieved the desired goals.

Although graph-based methods are simple and powerful, the automatic graph construction process is difficult to control and maintain, and as such researchers have often looked towards statistical methods for managing animation data instead. Linear methods such as Principal Component Analysis (PCA) and local PCA have been used to synthesise animation from low dimensional signals [Chai and Hodgins 2005; Tautges et al. 2011], while kernel-based methods have been applied successfully to build character controllers in many non-linear contexts [Grochow et al. 2004; Levine et al. 2012; Mukai 2011; Mukai and Kuriyama 2005; Park et al. 2002; Rose et al. 1998; Wang et al. 2008]. More recently, neural-network-based methods have grown in popularity. Auto-regression uses the current frame and user controls to predict the next frame [Fragkiadaki et al. 2015; Harvey and Pal 2018; Henter et al. 2019; Lee et al. 2018; Li et al. 2017; Park et al. 2019; Pavllo et al. 2019, 2018; Taylor and Hinton 2009; Wang et al. 2018], while offline methods use convolutions which move over the temporal dimension [Holden et al. 2016, 2015; Li et al. 2019].

Interactive generative models based on neural networks often exhibit a "dying-out" effect when generating long sequences of motion [Fragkiadaki et al. 2015]. Researchers have therefore been interested in ways to overcome this effect, including the introduction of a Phase or Gating Network [Holden et al. 2017; Zhang et al. 2018], yet such additions increase the complexity and resource requirements of the neural networks involved. In addition, such methods are difficult to extend to interactions with other characters, or with the environment. Lee et al. [2018] use a careful encoding of the goal state which updates a timer until completion each time step along with the network recurrence and pose state, and a smart method of data augmentation, to produce a character which can accurately follow user goals such as positioning and performing specific actions at specific times. Starke et al. [2019] propose an extension to the gating model, a specific environmental encoder network, as well as ego-centric and and goal-centric encodings to achieve a controller which can interact naturally with the environment. However, such methods can be difficult to train and control, and can often smooth out the appearance of animation or produce bad behaviour when extrapolating beyond the domain of the training data.

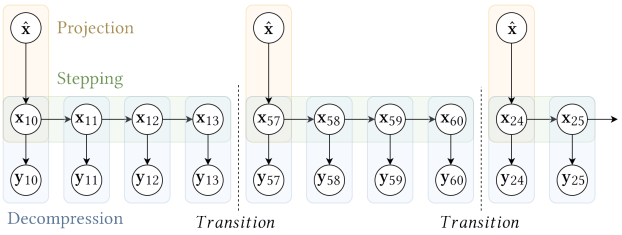

Fig. 2. Overview of the basic Motion Matching algorithm.

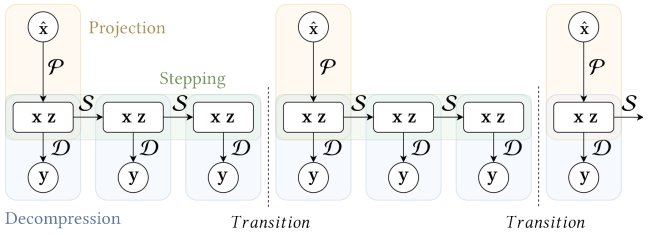

Fig. 3. Overview of the Learned Motion Matching algorithm.

2.2 Motion Matching

Motion Matching was first presented by Büttner and Clavet [2015] as a greedy approximation of the Motion Fields algorithm [Lee et al. 2010]. This system, which was developed for the game *For Honor*, was later presented by Clavet [2016] who framed it as a search over a database of animation to find the frame which best matches the current pose and user trajectory. Additional methodologies for data capture and control were presented by Zadziuk [2016]. Later, Harrower [2018] showed the same methods can be used for close character interactions, while Holden [2018] showed that Motion Matching is a special case of using machine learning for animation synthesis where a Nearest Neighbor Regression is used to map from user controls to animation data. More recently, Büttner [2019] and Zinno [2019] presented additional applications of Motion Matching including parkour and soccer. They both compared it to neural-network-based generative models, presenting the benefits in terms of quality and flexibility, while also addressing the core limitation of memory usage. In addition, Büttner [2019] presented a method using neural networks to accelerate the nearest neighbor search of Motion Matching by mapping the high dimensional feature vector into an optimized, smaller, quantized space [Jégou et al. 2011]. In academia, similar matching-based approaches have been used to produce kinematic responses to physical interactions [Zordan et al. 2005], while more recently, Bergamin et al. [2019] and Hong et al. [2019] have both used it as a lightweight and simple kinematic controller to guide a physically-based animation system.

3 BASIC MOTION MATCHING

While there are many variations of the Motion Matching algorithm, in this section we present the details of our own state-of-the-art implementation used in several AAA game productions. For a visual description of this algorithm please see Fig 2.

At a high level our algorithm works as follows: every N frames we search a database of animation for a frame which best matches some set of features which describe the current context and desired user properties of the animation we wish to produce. If a frame with a lower cost than the current frame is found, a transition is inserted, and the new animation is blended in.

More formally, we can start by defining a feature vector $\mathbf{x}$, which describes the features we wish to match at each frame. As shown in Section 5, many different behaviours can be described by choosing different feature vectors, but in the case of a locomotion controller we can define it in a similar way to Clavet [2016] where $\mathbf{x} = \{\mathbf{t}^t\ \mathbf{t}^d\ \mathbf{f}^t\ \dot{\mathbf{f}}^t\ \dot{\mathbf{h}}^t\} \in \mathbb{R}^{27}$, and $\mathbf{t}^t \in \mathbb{R}^6$ are the 2D future trajectory positions projected on the ground, 20, 40, and 60 frames in the future (at 60Hz) local to the character, $\mathbf{t}^d \in \mathbb{R}^6$ are the future trajectory facing directions 20, 40, and 60 frames in the future local to the character, $\mathbf{f}^t \in \mathbb{R}^6$ are the two foot joint positions local to the character, $\dot{\mathbf{f}}^t \in \mathbb{R}^6$ are the two foot joint velocities local to the character, and $\dot{\mathbf{h}}^t \in \mathbb{R}^3$ is the hip joint velocity local to the character.

Since features may be of vastly different magnitudes, it is important to normalize them. Here, we scale each feature (e.g. left foot position) by its standard deviation in the data-set. This scaling can then be tweaked further by a user weighting to adjust its importance in the search, however we found that in most cases a default weighting of 1 is sufficient.

Next, we define a pose vector $\mathbf{y}$, which contains all the pose information for a single frame of animation. In our case we can define $\mathbf{y} = \{\mathbf{y}^t\ \mathbf{y}^r\ \dot{\mathbf{y}}^t\ \dot{\mathbf{y}}^r\ \dot{\mathbf{r}}^t\ \dot{\mathbf{r}}^r\ \mathbf{o}^*\}$ where $\mathbf{y}^t\ \mathbf{y}^r$ are the joint local translations and rotations, $\dot{\mathbf{y}}^t\ \dot{\mathbf{y}}^r$ are the joint local translational and rotational velocities, $\dot{\mathbf{r}}^t\ \dot{\mathbf{r}}^r$ are the character root translational and rotational velocity, local to the character forward facing direction, and $\mathbf{o}^*$ are all the other additional task specific outputs, such as foot contact information, the position or trajectory of other characters in the scene, or the future positions of some joints of the character. In general we represent rotational velocities using the scaled angle axis representation, and joint rotations as quaternions, but convert to the 2-axis rotation matrix representation used in Zhang et al. [2018] when given as input to or output from a neural network. Although we describe our method using a pose representation common to video games, at a high level our algorithm does not make any assumptions about the pose representation and we found it worked equally well with character space encodings typical in other neural-network-based methods such as Holden et al. [2017].

Feature and pose vectors are computed for each frame i and concatenated into large matrices: $\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{n-1}]$ and $\mathbf{Y} = [\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_{n-1}]$, called the *Matching Database*, and *Animation Database* respectively. See Appendix A for more details.

At runtime, every N frames, or when the user input changes significantly, a query vector $\hat{\mathbf{x}}$ is constructed which contains the desired feature vector. Pose-based features can typically be extracted from the entry in the Matching Database $\mathbf{X}$ corresponding to the current frame i, while other user controls can be constructed via other means - for example, to control the features corresponding to the future trajectory position and direction we use a spring-damper-based system which generates a desired velocity and direction based on the current state of the joystick [Kermse 2004].

The task is to then find the entry in the Matching Database which minimizes the squared euclidean distance to the query vector.

$$k^* = \arg\min_k \|\hat{\mathbf{x}} - \mathbf{x}_k\|^2 \quad (1)$$

Once found, if the current frame i does not match the nearest frame, $i \neq k^*$, then animation playback continues from this point $i := k^*$, and a transition is inserted using inertialization blending [Bollo 2016, 2017]. Each frame we increment the frame variable $i := i + 1$ and the pose $\mathbf{y}_i$ is looked up in the Animation Database.

To support binary features such as gaits or other tags there are two options. Either these can be added to the feature vector $\mathbf{x}$ as one-hot encoded vectors, or alternatively, ranges in the Animation and Matching Databases can be pre-computed, and the search limited to these ranges when certain gaits or tags are required.

Since a brute force search over every entry of $\mathbf{X}$ is too slow, generally an acceleration structure is required such as a KD-Tree [Clavet 2016], a voxel-based lookup table (as in Büttner [2018]), or clustering [Yi and Jee 2019]. We found a custom two-layer bounding volume hierarchy consisting of axis-aligned bounding boxes fitted to groups of 16 frames and 64 frames was particularly simple and effective. See Appendix B for more details.

Finally, PCA can be applied to the Feature Database $\mathbf{X}$ and other feature vectors such as $\hat{\mathbf{x}}$ to reduce their dimensionality. This non-essential step does not change the overall behaviour of the algorithm (as, providing enough dimensions are kept, distances are preserved under PCA) so we omit it from the rest of our explanation, but note that it can be a useful way to reduce memory usage and increase performance at the cost of some search accuracy when a large number of redundant or correlated features are used.

4 LEARNED MOTION MATCHING

By examining Fig 2 we can observe that the Motion Matching algorithm consists of three key stages: *Projection:* where a nearest neighbor search is used to find the feature vector in the Matching Database which best matches the query vector, *Stepping:* where we advance forward the index in the Matching Database, and *Decompression:* where we look up the associated pose in the Animation Database which corresponds to our current index in the Matching Database. These stages are simply repeated every N frames and a blend is inserted to remove any discontinuity.

In terms of scale, the core issue with this system is the reliance on $\mathbf{X}$ and $\mathbf{Y}$, which incur large memory overheads and grow as we add additional animations, pose features, or matching features.

In the rest of this section we show how we fix this issue by creating learned alternatives to each of the three key stages and remove the need to store any databases in memory. First, we remove the reliance on $\mathbf{Y}$ by training a decoder network called the *Decompressor* which takes as input the feature vector $\mathbf{x}$, as well as additional latent variables $\mathbf{z}$, found via an encoder network called the *Compressor*, and outputs a pose vector $\mathbf{y}$. Next, we remove the reliance on $\mathbf{X}$ by training two networks that work together called the *Stepper*, and the *Projector*. The *Stepper* learns the dynamics of the system, advancing the matching and latent feature vectors forward in time by outputting a delta which is added to $\mathbf{x}_i$ and $\mathbf{z}_i$ to produce $\mathbf{x}_{i+1}$ and $\mathbf{z}_{i+1}$, while the *Projector* emulates the nearest neighbor search, taking some query vector $\hat{\mathbf{x}}$ as input, and mapping it to the feature vector and latent variables of the nearest match in the database $\mathbf{x}_{k^*}$ and $\mathbf{z}_{k^*}$.

Algorithm 1: Our training algorithm for Decompressor $\mathcal{D}$.

Function `TrainDecompressor`$(\mathbf{X}, \mathbf{Y}, \boldsymbol{\theta}_C, \boldsymbol{\theta}_{\mathcal{D}})$**:**

- `/* Compute forward kinematics */`
- $\mathbf{Q} \leftarrow \text{ForwardKinematics}(\mathbf{Y})$
- `/* Generate latent variables Z */`
- $\mathbf{Z} \leftarrow C([\mathbf{Y}\ \mathbf{Q}]^T; \boldsymbol{\theta}_C)$
- `/* Reconstruct pose Ỹ */`
- $\tilde{\mathbf{Y}} \leftarrow \mathcal{D}([\mathbf{X}\ \mathbf{Z}]^T; \boldsymbol{\theta}_{\mathcal{D}})$
- `/* Recompute forward kinematics */`
- $\tilde{\mathbf{Q}} \leftarrow \text{ForwardKinematics}(\tilde{\mathbf{Y}})$
- `/* Compute latent regularization losses */`
- $\mathcal{L}_{lreg} \leftarrow w_{lreg} \| \mathbf{Z} \|_2^2$
- $\mathcal{L}_{sreg} \leftarrow w_{sreg} \| \mathbf{Z} \|_1$
- $\mathcal{L}_{vreg} \leftarrow w_{vreg} \left\| \frac{\mathbf{Z}_0 - \mathbf{Z}_1}{\delta t} \right\|_1$
- `/* Local & character space losses */`
- $\mathcal{L}_{loc} \leftarrow w_{loc} \left\| \mathbf{Y} \ominus \tilde{\mathbf{Y}} \right\|_1$
- $\mathcal{L}_{chr} \leftarrow w_{chr} \left\| \mathbf{Q} \ominus \tilde{\mathbf{Q}} \right\|_1$
- `/* Local & character space velocity losses */`
- $\mathcal{L}_{lvel} \leftarrow w_{lvel} \left\| \frac{\mathbf{Y}_0 \ominus \mathbf{Y}_1}{\delta t} - \frac{\tilde{\mathbf{Y}}_0 \ominus \tilde{\mathbf{Y}}_1}{\delta t} \right\|_1$
- $\mathcal{L}_{cvel} \leftarrow w_{cvel} \left\| \frac{\mathbf{Q}_0 \ominus \mathbf{Q}_1}{\delta t} - \frac{\tilde{\mathbf{Q}}_0 \ominus \tilde{\mathbf{Q}}_1}{\delta t} \right\|_1$
- `/* Update network parameters */`
- $\boldsymbol{\theta}_C\ \boldsymbol{\theta}_{\mathcal{D}} \leftarrow \text{RAdam}(\boldsymbol{\theta}_C\ \boldsymbol{\theta}_{\mathcal{D}}, \nabla \sum_* \mathcal{L}_*)$

end

The result is the *Learned Motion Matching* algorithm shown in Fig 3. We use the Projector network on user input $\hat{\mathbf{x}}$ to project it onto the nearest feature vector in the matching database and associated latent variables. The Stepper is then used to move this set of feature vector and latent variables forward in time, and the Decompressor is used to generate the full character pose. As in basic Motion Matching, this process is simply repeated every N frames, and a blend is inserted to remove the discontinuity.

4.1 Decompressor

The goal of the *Decompressor* is to avoid storing $\mathbf{Y}$ in memory by instead taking the feature vector at a particular frame $\mathbf{x}_i$ and producing the corresponding pose $\mathbf{y}_i$ directly. While the feature vector often includes key information about the corresponding pose such as the foot positions and velocities (making this mapping partially possible), it usually does not contain enough information to fully infer it. For this reason we introduce additional latent variables $\mathbf{z}_i$, which we discover via an autoencoder-like structure. Using a network called the *Compressor*, we map a pose $\mathbf{y}_i$ to a low dimensional representation $\mathbf{z}_i$ which is then concatenated to $\mathbf{x}_i$, and given as input to the *Decompressor* which attempts to reconstruct the original pose $\mathbf{y}_i$. In this way we learn what additional information is missing from the feature vector $\mathbf{x}$, and encode it in $\mathbf{z}$.

A key aspect of the *Decompressor* is the loss function. If we use a naïve mean squared error loss we observe jittery, low quality motion (see Section 6.3). Instead, we design a loss function intended to minimize the visually perceived error which, as in Pavllo et al. [2019; 2018], uses forward kinematics to measure the error in character space as well as velocity-based losses which ensure the output pose changes smoothly in time. We also add some additional regularization losses on $\mathbf{Z}$ to induce sparsity and smoothness.

For details on the training procedure please see Algorithm 1. Here, given 2 frames from the Animation Database $\mathbf{Y}$ and Matching Database $\mathbf{X}$, we use the Compressor C to find latent variables $\mathbf{Z}$, and the Decompressor $\mathcal{D}$ to try and reconstruct the original pose. While we present the procedure for a single training sample (a pair of two frames), we apply it to each element in a mini-batch and average the result when updating network parameters $\boldsymbol{\theta}_C, \boldsymbol{\theta}_{\mathcal{D}}$. The operator $\ominus$ computes the difference between two poses where all pose differences are defined as basic subtractions with the exception of the rotational difference of $\mathbf{y}^r$, which is computed by first converting to rotation matrix form and then subtracting. In addition, in the velocity losses $\mathcal{L}_{lvel}$ and $\mathcal{L}_{cvel}$ we avoid computing the difference in pose velocities $\dot{\mathbf{y}}^r$ and $\dot{\mathbf{y}}^t$ as these accelerations are too noisy to be useful. Weights $\mathbf{w}_*$ are set to give roughly equal weight to all pose-based losses and a small weighting to regularization losses.

We found providing the Compressor with both local and character space inputs improved the accuracy as it was able to copy features directly to the latent space if it found them useful. For example, character space foot joint velocities are a useful feature for predicting joint contact information.

Once trained, the Decompressor is already useful without the other networks. By making the dimensionality of $\mathbf{z}$ small, computing $\mathbf{z}_i$ for each frame i, and storing $\mathbf{Z} = [\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_{n-1}]$ instead of $\mathbf{Y}$, we can achieve significant memory savings without affecting at all the behaviour of the Motion Matching algorithm. The Compressor and Decompressor can therefore be used as a fairly effective general purpose compression method for animation data even when no matching features are present. For more details see Table 3.

While the Decompressor removes the need to store $\mathbf{Y}$ in memory, we must still store $\mathbf{X}$ and $\mathbf{Z}$, both of which can be fairly large when many matching features or latent variables are used. In Section 4.2 and Section 4.3 we describe how we also remove the need to store these two additional databases.

4.2 Stepper

Given some initial matching and latent feature vectors $\mathbf{x}_i$ $\mathbf{z}_i$, each frame we must advance forward in time to retrieve the next consecutive vectors $\mathbf{x}_{i+1}$ and $\mathbf{z}_{i+1}$. In basic Motion Matching this is trivial - we simply increment the index i and look-up the new row of $\mathbf{X}$ or $\mathbf{Z}$. However, if we wish to avoid storing $\mathbf{X}$ and $\mathbf{Z}$ in memory this simple advancement is not possible.

Instead, we introduce the *Stepper*, a network trained to take as input matching and latent feature vectors at a given frame $\mathbf{x}_i$ $\mathbf{z}_i$, and output a delta which can be added to produce the feature vectors at the next frame $\mathbf{x}_{i+1}$ $\mathbf{z}_{i+1}$.

For details on the training procedure please see Algorithm 2. Here, we train the network in an auto-regressive fashion, and given a short window of s feature vectors $\mathbf{X}$ and latent variables $\mathbf{Z}$, we repeatedly predict the next feature and latent variables and feed them in at the next frame. While we present the procedure for a single training sample, we apply it to each element in the mini-batch and average the result when updating $\boldsymbol{\theta}_{\mathcal{S}}$. Weights $\mathbf{w}_*$ are set to give roughly equal weight to all losses.

Algorithm 2: Our training algorithm for Stepper $\mathcal{S}$.

Function `TrainStepper`($\mathbf{X}$, $\mathbf{Z}$, s, $\boldsymbol{\theta}_{\mathcal{S}}$)**:**

- `/* Set initial states */`
- $\tilde{\mathbf{X}}_0,\ \tilde{\mathbf{Z}}_0 \leftarrow \mathbf{X}_0,\ \mathbf{Z}_0$
- `/* Predict` $\tilde{\mathbf{X}}$ `and` $\tilde{\mathbf{Z}}$ `over a window of` s `frames */`
- **for** $i \leftarrow 1$ **to** s **do**
  - `/* Predict deltas for` $\tilde{\mathbf{X}}$ `and` $\tilde{\mathbf{Z}}$ `*/`
  - $\delta\tilde{\mathbf{x}},\ \delta\tilde{\mathbf{z}} \leftarrow \mathcal{S}([\tilde{\mathbf{X}}_{i-1}\ \tilde{\mathbf{Z}}_{i-1}]^T; \boldsymbol{\theta}_{\mathcal{S}})$
  - $\tilde{\mathbf{X}}_i \leftarrow \tilde{\mathbf{X}}_{i-1} + \delta\tilde{\mathbf{x}}$
  - $\tilde{\mathbf{Z}}_i \leftarrow \tilde{\mathbf{Z}}_{i-1} + \delta\tilde{\mathbf{z}}$
- **end**
- `/* Compute losses */`
- $\mathcal{L}_{xval} \leftarrow w_{xval} \left\| \mathbf{X} - \tilde{\mathbf{X}} \right\|_1$
- $\mathcal{L}_{zval} \leftarrow w_{zval} \left\| \mathbf{Z} - \tilde{\mathbf{Z}} \right\|_1$
- $\mathcal{L}_{xvel} \leftarrow w_{xvel} \left\| \frac{\mathbf{X}_{0\to s-1} - \mathbf{X}_{1\to s}}{\delta t} - \frac{\tilde{\mathbf{X}}_{0\to s-1} - \tilde{\mathbf{X}}_{1\to s}}{\delta t} \right\|_1$
- $\mathcal{L}_{zvel} \leftarrow w_{zvel} \left\| \frac{\mathbf{Z}_{0\to s-1} - \mathbf{Z}_{1\to s}}{\delta t} - \frac{\tilde{\mathbf{Z}}_{0\to s-1} - \tilde{\mathbf{Z}}_{1\to s}}{\delta t} \right\|_1$
- `/* Update network parameters */`
- $\boldsymbol{\theta}_{\mathcal{S}} \leftarrow \text{RAdam}(\boldsymbol{\theta}_{\mathcal{S}}, \nabla \sum_* \mathcal{L}_*)$

end

Once trained, the Stepper can be used as a replacement for the *Stepping* part of the pipeline, and produce a stream of matching and latent feature vectors without reliance on $\mathbf{X}$ or $\mathbf{Z}$. While an initial starting state may be found using the Compressor, the nearest neighbor search still requires $\mathbf{X}$ and $\mathbf{Z}$ to be kept in memory. In Section 4.3 we describe how this limitation is removed using a network trained to map from some user query $\hat{\mathbf{x}}$ to $\mathbf{x}$ and $\mathbf{z}$ directly.

4.3 Projector

Although the *Stepper* allows us to advance forward feature vectors in time, the matching database $\mathbf{X}$ is still required by the nearest neighbor search that is performed on the query vector $\hat{\mathbf{x}}$.

To finally remove the need to store $\mathbf{X}$ and $\mathbf{Z}$ in memory we introduce the *Projector*, a network trained to emulate the nearest neighbor search behaviour and produce feature vectors matching the nearest entry $\mathbf{x}_{k^*}$ from $\hat{\mathbf{x}}$.

For details on the training procedure please see Algorithm 3. Here, given a feature vector from the Matching Database $\mathbf{x}$, we sample a noise magnitude n^σ and use it to scale a Gaussian noise vector $\mathbf{n}$. We add this to $\mathbf{x}$ to produce $\hat{\mathbf{x}}$ and find the nearest neighbor k^*. The Projector is then trained to output the associated feature vector and latent variables $\mathbf{x}_{k^*}$ and $\mathbf{z}_{k^*}$. While we present the procedure for a single training sample here, we apply it to each element in the mini-batch and average the result when updating $\boldsymbol{\theta}_{\mathcal{P}}$. By sampling

Algorithm 3: Our training algorithm for Projector $\mathcal{P}$.

Function `TrainProjector(`$\mathbf{x}, \mathbf{X}, \mathbf{Z}, \theta_{\mathcal{P}}$`):`

```
/* Sample uniform noise magnitude n^σ          */
n^σ ~ U(0, 1)
/* Sample gaussian noise vector n              */
n ~ N(0, 1)
/* Add noise to feature vector                 */
x̂ ← x + n^σ n
/* Find nearest neighbor                       */
k* = Nearest(x̂, X)
/* Project feature vector                      */
x̃, z̃ ← P(x̂; θ_P)
/* Compute losses                              */
L_xval ← w_xval ‖ x_k* − x̃ ‖_1
L_zval ← w_zval ‖ z_k* − z̃ ‖_1
L_dist ← w_dist ‖ ‖ x̂ − x_k* ‖_2^2 − ‖ x̂ − x̃ ‖_2^2 ‖_1
/* Update network parameters                   */
θ_P ← RAdam(θ_P, ∇ Σ_* L_*)
```

end

Table 1. Network architecture details such as the number of layers (including input and output layers), number of units in each of the hidden layers, and the activation functions used for each network. Note: the Compressor is generally not required at runtime, and so is larger with a more expensive activation function.

Network	#Layers	#Units	Activation
$\mathcal{C}$	5	512	ELU
$\mathcal{D}$	3	512	ReLU
$\mathcal{S}$	4	512	ReLU
$\mathcal{P}$	6	512	ReLU

different noise magnitudes, we make the Projector robust to perturbations of different sizes. Weights $\mathbf{w}_*$ are set to give roughly equal weight to all losses.

Once trained, the Projector completes our Learned Motion Matching pipeline. Instead of a nearest neighbor search, every N frames we pass the user query $\hat{\mathbf{x}}$ through the Projector $\mathcal{P}$. Then, each frame, we advance forward the found matching and latent feature vectors using the Stepper $\mathcal{S}$, and decode them to produce a pose using the Decompressor $\mathcal{D}$.

4.4 Training

All networks are trained in PyTorch using the RAdam optimizer [Liu et al. 2019], with a batch size of 32 and a learning rate of 0.001 which is decayed by a factor of 0.99 every 1000 iterations. The Stepper network is trained with a window size s of $2N$ frames, where N is the number of frames in-between each search (typically ~10). We use a latent variable dimensionality of $\mathbf{z} \in \mathbb{R}^{32}$. For all networks, training is performed for up to 500,000 iterations single-threaded on an Intel Xeon 3.5Ghz 12 Core CPU. We found the small network sizes made training on the CPU almost always more efficient than training on the GPU. Although results can be obtained in a few hours, training overnight is required for optimal results (see Table 3). Note that while the Decompressor must always be trained first to allow for the latent variables $\mathbf{Z}$ to be computed, the Stepper and Projector can afterwards be trained in parallel. For specific details on the network architectures used please see Table 1.

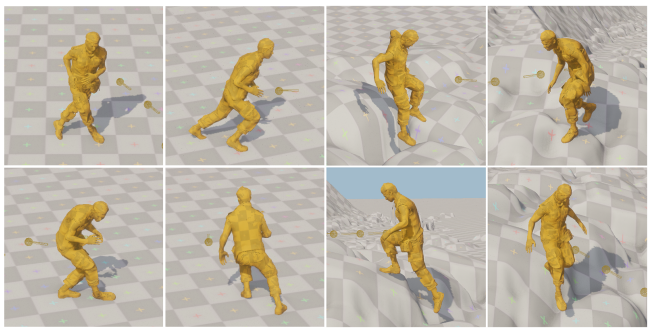

Fig. 4. Our method applied to a typical character locomotion scenario and a character traversing rough terrain.

Fig. 5. Our method applied to a character getting up and sitting down on a chair and two characters interacting.

Fig. 6. Our method applied to quadruped dog and bear characters.

5 RESULTS

In this section we present the results of our method applied in various Motion Matching scenarios, an overview of which is shown in Fig 1 and numerical details of which are provided in Table 2. For more in-depth results please see the supplementary video.

Table 2. Details of the different scenarios shown in our results.

Scenario	#Joints	#Frames	#Features	Frame Rate
Locomotion	28	89480	27	60Hz
Terrain	28	170534	35	60Hz
Chair	28	22398	19	60Hz
Interaction	28	38976	32	60Hz
Dog	58	124418	39	60Hz
Bear	75	694272	55	60Hz
User-Study	54	23382	27	30Hz

In Fig 4 we show our method applied to character locomotion. Here we produce high quality, natural and responsive motion without the need for phase or other additional variables. In this case the matching features are the foot positions and velocities relative to the root, the hip velocity, the future trajectory positions and directions at 20, 40, and 60 frames in the future (at 60Hz), as well as the gait of the character (walking, running, or crouching). In addition to the pose, we output from the Decompressor binary contact information and apply inverse kinematics to remove foot sliding. We extend this demo to locomotion over rough terrain by including additional features such as the height of the terrain under the toes 0, 15, 30, and 45 frames in the future. This height is recorded relative to the current height of the character's hips. At runtime, we therefore need to output from the Decompressor the future toe positions 15, 30, and 45 frames in the future so we can find the height below them, while at training time we need to find the height of the terrain in the motion capture data. For more information on how we do this please see Appendix D.

In Fig 5 we show our method applied to sitting and getting up from a chair. Here we use the standard locomotion controller shown previously, but in addition regularly try to match from a set of sitting down and getting up animations using the relative chair position and direction. If a match is found with a cost below some threshold we switch to this controller and then regularly re-search to see if an even better match can be found. Trajectory warping and inverse kinematics are also applied such that the character arrives exactly at the chair location without foot sliding. For get-up animations we similarly match the position and direction of the character at the end of the animation with the newly placed chair. We also show our method applied to character interactions. Here, the player-controlled character matches the best fitting interaction using features which include the other character's position, facing direction, and velocity. The non-player-controlled character matches the trajectory that was present in synchronized animation played by player-controlled character. This synchronized trajectory must be output by the Decompressor of the player-controlled character, as otherwise it would not be available at runtime.

In Fig 6 we show our method applied to a dog character trained on a large database of raw unstructured motion capture data. Here we match the position and velocity of all four paws as well as the future trajectory and gait. Our method naturally extends to quadrupeds, and produces high quality movement without any loss of fidelity or need for phase information or specific gating networks. In addition, we show our method using the same set of features, but applied to a bear. By using an automatic data-augmentation technique known as carpet unrolling [Miller et al. 2015] on around one minute of key-framed animation data (3581 frames), we can produce a wide variety of turns as well as locomotion over rough terrain. The result is an extensive database consisting of ~700000 frames of animation. This large, augmented database is very inefficient when combined with basic Motion Matching but is easily accommodated by our method.

Fig. 7. Comparison between our method and basic Motion Matching. Left: Our Method (LMM), Right: basic Motion Matching (MM).

6 EVALUATION

In this section we evaluate our method, first comparing our results to basic Motion Matching and other state-of-the-art neural-network-based models, and then measuring the performance characteristics. Finally, we perform a user study using an implementation of our method in a AAA production where Motion Matching is used for third-person player locomotion.

6.1 Comparison

In this section we compare our method against basic Motion Matching and other state-of-the-art neural-network-based models including Phase-Functioned Neural Networks (PFNN) and Mode-Adaptive Neural Networks (MANN). In all comparisons we train on the same data, use the same input features, and apply the same pre-processing and post-processing steps (such as foot sliding clean-up), with the exception of animation blending which is not required by the PFNN or MANN.

In Fig 7 we see a side-by-side comparison between our method and basic Motion Matching. Here, when a search is triggered we give both systems the same input and evaluate them in parallel until the next search is triggered. It is difficult to tell the difference between our results and that of basic Motion Matching even though we achieve large memory gains.

In Fig 8 we compare our method against a PFNN trained on the rough terrain data set. While the PFNN produces motion which adapts to the terrain, we find our method retains more of the detail found in the original data and overall looks smoother and more diverse. We also compare our method against a MANN trained on our quadruped data set. Similarly, while the MANN produces natural motion, it does not retain the same level of detail or quality as our method which more closely resembles the original training data. For a better visual comparison please see the supplementary video.

Fig. 8. Comparison between our method and other neural-network-based models Left: Our Method (LMM), Top Right: Phase-Functioned Neural Networks (PFNN), Bottom Right: Mode-Adaptive Neural Networks (MANN).

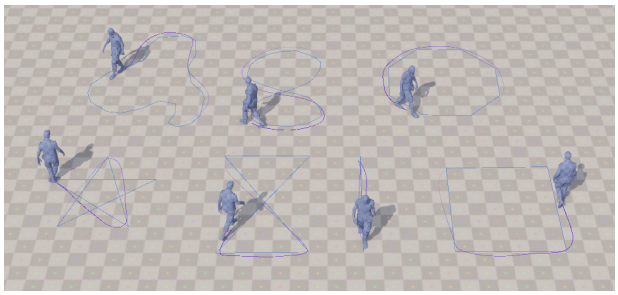

Fig. 9. Evaluation of how closely our method follows pre-defined paths.

Fig. 10. Experiment showing the impact of inverse kinematics on our method. Left: Rough terrain traversal data used but inverse kinematics disabled. Right: Inverse kinematics enabled, but only flat locomotion data used.

6.2 Performance

In this section we evaluate the performance and memory usage of our method. For full details please see Table 3. All performance measures including neural network inference were made single-threaded on an Intel Xeon 3.5Ghz 12 Core CPU using a custom, optimized neural network inference library. In Animation Databases $\mathbf{Y}$, the velocity information is removed and instead computed on-the-fly via finite difference, while joint translations that do not change are only stored once to reduce overall memory usage.

One of the key aspects of our method is its performance in terms of evaluation time and memory usage. We find our method has faster evaluation time than other state-of-the-art neural-network-based models, while using less memory. When compared to basic Motion Matching our method has a slower evaluation time, but vastly improved memory usage. In addition, if faster computation time is required, or the animation database is small (such as in the *Chair* example), usage of the Decompressor by itself provides a compromise between the two. Although we do not present it here, we found neural network weights could be effectively compressed as 16-bit integers without significant loss of precision or runtime performance, potentially reducing the memory usage of Learned Motion Matching further by a factor of two.

We note that while we compare our method to uncompressed animation data, state-of-the-art animation compression technology generally achieves ~5× compression ratios for databases stored in the way described above, with decompression times of ~20μs per frame [Frechette 2019]. Additionally, while our method requires blending, other auto-regressive neural-network-based models do not. In our experiments blending generally took ~20μs per frame. Even taking these difference into account the overall characteristics of our algorithm compared to others do not change.

Finally, important to note is that although training times for our method are long compared to basic Motion Matching (which requires no training), our framework still allows artists and designers to iterate quickly as they can use basic Motion Matching until they are happy with their results, and then simply switch to Learned Motion Matching as a post-process.

6.3 Experiments

In this section we present some additional experiments designed to evaluate our method. For better visual explanation please see additional supplementary material.

In Fig 9 we show our approach evaluated in a path following setup. Here, we see the characters are able to fairly closely follow the user defined paths. It is worth noting that our feature weights are configured for animation quality rather than path following precision, and by reducing the weight for the foot position and velocity features a more accurate path following can be achieved [Clavet 2016].

We evaluate the numerical precision of our Decompressor by measuring the distribution of positional errors of joints compared to the ground truth. We found our method produced visually smooth animation with an average positional error of 1.4cm, and standard deviation of 1.1cm (evaluated for the *Terrain* scenario). We also found that when using a standard mean squared error loss function the

Table 3. Performance and memory comparison between basic Motion Matching (MM), Motion Matching with the Decompressor (DMM), Our Method - Learned Motion Matching (LMM), Phase-Functioned Neural Networks [Holden et al. 2017] (PFNN), and Mode Adaptive Neural Networks [Zhang et al. 2018] (MANN). Missing entries are assumed to be zero. A few important things to note: (1) The Stepper and Projector are trained in parallel, meaning the total training time is only affected by the longer of the two. (2) Projection is performed only every $N = 10$ frames, reducing the average impact on frame time. (3) The PFNN and MANN are both trained using the GPU as otherwise training would take too long, while all other networks are trained using the CPU.

		Memory (MB)							**Performance (μs)**				**Training (hours)**			
Scenario	**Method**	**X**	**Y**	**Z**	$\mathcal{D}$	$\mathcal{S}$	$\mathcal{P}$	**Total**	$\mathcal{D}$	$\mathcal{S}$	$\mathcal{P}$	**Frame**	$\mathcal{D}$	$\mathcal{S}$	$\mathcal{P}$	**Total**
Locomotion	MM	9.4	42.7					52.1			90	9				
	DMM	9.4		11.1	0.9			21.4	85		90	94	8.7			8.7
	LMM				0.9	1.2	3.2	5.3	85	100	127	197	8.7	6.1	3.3	14.7
Terrain	MM	23.3	81.4					104.7			213	21				
	DMM	9.4		21.3	1.0			31.7	81		213	102	8.8			8.8
	LMM				1.0	1.3	3.2	5.5	81	106	129	200	8.8	6.5	5.4	15.3
	PFNN					9.3		9.3		1370		1370		21.5		21.5
Chair	MM	1.6	10.7					12.3			62	6				
	DMM	1.6		2.8	1.9			6.3	80		62	86	8.6			8.6
	LMM				1.9	2.4	6.4	10.7	80	93	111	184	8.6	6.2	2.8	14.8
Interaction	MM	4.8	18.6					23.4			140	14				
	DMM	4.8		4.8	1.9			11.4	82		140	96	8.4			8.4
	LMM				1.9	2.5	6.5	10.9	82	102	131	197	8.4	6.3	2.8	14.7
Dog	MM	18.9	117.6					136.5			111	11				
	DMM	18.9		15.5	1.9			36.3	131		111	142	16.9			16.9
	LMM				1.9	1.3	3.3	6.5	131	118	139	262	16.9	6.1	5.2	23.0
	MANN					16.9		16.9		2440		2440		5.9		5.9
Bear	MM	149.1	846.5					995.6			946	94				
	DMM	149.1		86.7	2.5			238.3	93		946	187	20.4			20.4
	LMM				2.5	1.3	3.3	7.1	93	110	137	340	20.4	6.5	8.9	29.3
User-Study	MM	2.4	21.0					23.4			110	11				
	DMM	2.4		2.9	1.7			7.0	83		110	94	16.6			16.6
	LMM				1.7	1.2	3.2	6.1	83	62	98	155	16.6	6.3	6.4	23.0

Decompressor produced noisy, jittery motion compared to our loss function. For visual comparison please see additional supplementary material.

Finally, in Fig 10 we show our method applied to locomotion over rough terrain in two separate ablations. Firstly, where inverse kinematics is disabled, and secondly, where inverse kinematics is enabled, but where the system has only been training on flat locomotion data. We observe that without inverse kinematics enabled our method exhibits some foot sliding and penetration into the terrain, while when excluding data of traversal over rough terrain the motion looks unnatural and unbalanced.

6.4 User Study

We performed a user study to evaluate how our method compares to basic Motion Matching in terms of perceived differences, motion quality, and responsiveness. To ensure the study was performed in realistic conditions, we implemented Learned Motion Matching in a AAA production where Motion Matching is used for third-person control of the player character. We selected 34 subjects, with the goal of covering a wide spread of expertise and experience. Subjects included 4 women and 30 men, with ages ranging from 21 to 47, of which 73% considered themselves gamers. In addition, 17 of our subjects had relevant professions in the industry including animators, animation directors, motion capture artists, and animation or gameplay programmers. Participants were not told the technology behind any of the systems tested but received a full explanation of the study and the questions they were going to be asked ahead of time. The study was separated into two stages, described below.

In the first stage, subjects were provided with a character controller, referred to as the *reference* system, and were allowed to interactively control the character for up to five minutes until they felt familiar enough with the system to proceed. They were then presented with two new character controllers denoted as A and B, and were allowed to interactively control the character as before for up to 10 minutes, as well as given freedom to switch between systems A and B at any time using a key press. Subjects were then asked a series of questions with responses on a Likert scale. Firstly, they were asked which of A or B they believed was the original reference system they had played first (see Fig 11). Secondly, they were asked which system they preferred in terms of animation quality (see Fig 12). And thirdly, which they preferred in terms of responsiveness (see Fig 13).

In this case, the original reference system presented to the subjects was basic Motion Matching (MM), while systems A and B, presented afterwards, were, for each user individually, randomly

selected between basic Motion Matching (MM) and Learned Motion Matching (LMM). We found that although there was a trend toward correctly identifying the reference system, many subjects struggled to identify the original reference system and were either unsure or answered incorrectly. In terms of quality and responsiveness, although we again found a slight preference toward Motion Matching in terms of responsiveness, there was no preference in terms of quality, and in general we found subjects had a large spread of different opinions.

In the second part of the study, we allowed subjects once again to control the reference system for several minutes. After they felt comfortable we presented them with a sequence of 10 unknown systems, randomly selected between MM and LMM. Subjects were then asked to identify the system as either the reference system or not the reference system. The results, reported in Fig 14, show that in this setup subjects found it very difficult to classify the reference system, including those who confidently and correctly identified the reference system in the first part of the study. Additionally, we did not find any correlation between a subject's experience and the accuracy of their answers, and some of the most experienced subjects in terms of content evaluation (such as animation directors) were unable to identify which was the reference system in either stage of the study.

Finally, we asked each participant if they had comments on the experiment and systems they tried. All subjects emphasised how difficult they found it to tell any difference between the two systems, and some subjects noted they were forcing themselves to focus on tiny details which they would not do in a usual gaming context. Those with more experience sometimes preferred the motion quality of the learned system stating that it had "smoother transitions", while the system using basic Motion Matching was said to feel more "dynamic", "snappier", and "executed plant-and-turn animations better".

The subjects who scored highest in the second test spent significantly longer than other subjects switching back and forth between systems A and B during the first test, enough to discover slight differences in the character pose either during a looping animation or during animation transitions. Then, when allowed to play with the reference system again before the second test they were able to identify which was which.

When asked if one animation system would be acceptable as a replacement for the other in-game, 32 subjects answered positively emphasizing the close similarity of the two, while two subjects answered negatively. Among the latter, one strongly preferred the smoothness of the learned system while the other preferred the snappiness of basic Motion Matching.

7 DISCUSSION

In this section we discuss some of the decisions behind our method including why we use multiple networks and the generalization behaviour of our system.

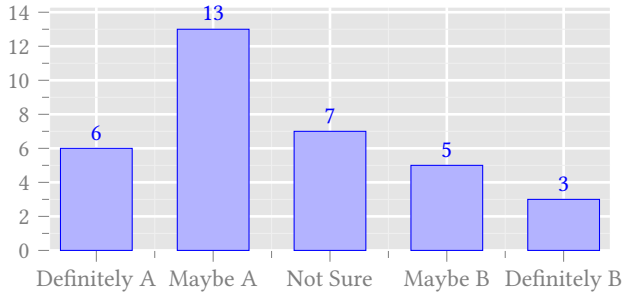

Fig. 11. Discrimination Test. Distribution of answers to the question "Which system do you think is the reference system?" with the reference system being Motion Matching (MM) and, for simplicity of presentation, system A being the correct answer in this case.

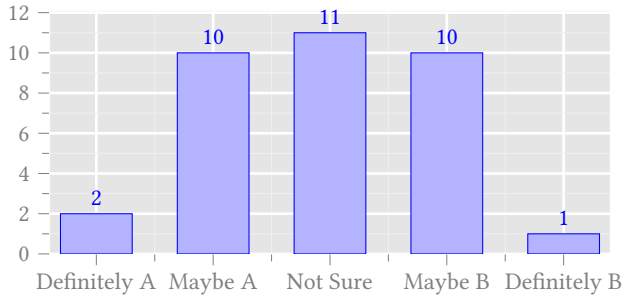

Fig. 12. Quality Evaluation. Distribution of answers to the question "Which system has better animation quality?" with A: Motion Matching (MM) and B: Learned Motion Matching (LMM).

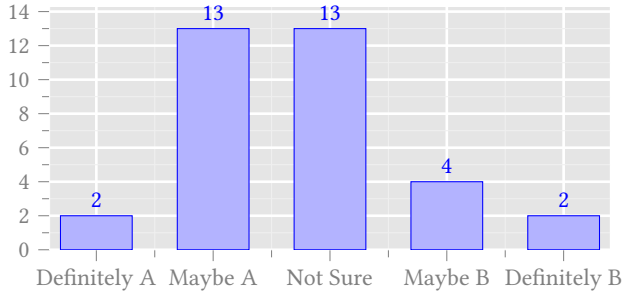

Fig. 13. Responsiveness Evaluation. Distribution of answers to the question "Which system has better responsiveness?" with A: Motion Matching (MM) and B: Learned Motion Matching (LMM).

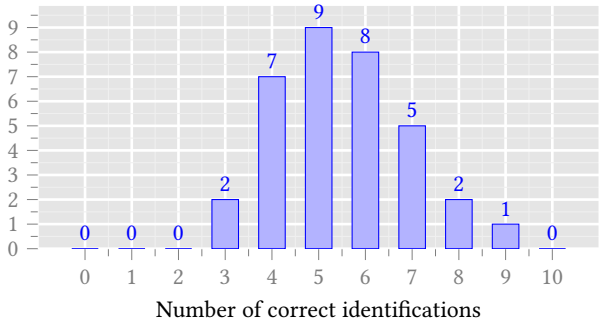

Fig. 14. Classification Test. Number of correct identifications of 10 randomly selected systems between Motion Matching and Learned Motion Matching.

7.1 Multiple Networks

A natural question about our system is, if it's possible to use fewer networks, and what are the benefits of using three separate networks? We found using multiple networks had a number of interesting advantages: firstly, it was possible to train, debug, and toggle each network in isolation and verify that the behaviour it exhibited matched what was desired. Secondly, we found that using the Decompressor alone was useful in many cases and provided a good compromise when not a lot of animation data is used. Thirdly, we found having the Decompressor and Projector work on a per-pose basis, with all dynamics handled by the Stepper, produced a system that was more stable, easier to understand, and simpler to use. It is, for example, a fairly easy extension to make our system work with variable time steps by appropriately scaling the output of the Stepper and training on a variety of time steps. On the other hand, making this kind of change in a large, end-to-end model such as Starke et al. [2019] would likely require significant investment and re-training. Finally, there is a theoretical contribution: each network in our system plays specific roles which might be applicable in other styles of neural-network-based model. For example, the Projector has two key roles: firstly, it maps the user controls onto the training data distribution which is useful when the user provides invalid requests the system has not been trained on; secondly, it regularly "resets" the system, ensuring the recurrence does not "die out", get stuck in small loops, or drift away from the data manifold, as is common in traditional recurrent models [Fragkiadaki et al. 2015].

7.2 Generalization

Unlike other neural-network-based models we actively encourage our system *not* to generalize as the Projector emulates the nearest neighbor search. This has some obvious downsides - primarily that our model does not interpolate to produce new animations not present in the training data. However, it also has some advantages: by closely fitting the training data and avoiding extrapolation we ensure the state of the system is always close to a state it has seen during training. And, while the lack of interpolation means our model requires a more thorough data coverage, automatic data augmentation (where the results can be verified by animators) can be used to increase coverage instead. Similarly, by adjusting the weights of the matching features, animators can change the projection behaviour of the nearest neighbor search - giving even finer control over how the system behaves when there is no clear, obvious match to select. Finally, although our networks are not trained to generalize we found they were still robust to small differences in the input. Most likely this is because their limited size prevents them from over-fitting aggressively.

8 LIMITATIONS AND FUTURE WORK

During our research we found that the final accuracy of the Projector had the highest impact on the overall quality of our results. It is also the largest network, using the most memory, and whilst we found increasing the number of hidden layers improves the quality of the results, it can also start to drastically increase the runtime evaluation time beyond production budgets. We believe this limitation is the reason for some of the perceived additional smoothness reported by our user study subjects. As a future step, it would be interesting to explore ways of increasing the accuracy without sacrificing runtime performance such as by using memory layers [Lample et al. 2019]. Similarly, it would be interesting to find a way to train our system end-to-end, rather than the two-stage process we present in this paper.

9 CONCLUSION

In this paper we presented Learned Motion Matching, an alternative to Motion Matching consisting of three unique neural networks each trained to emulate some particular component of the algorithm. We showed that with our method we can combine the scalability of existing neural-network-based methods with the flexibility of Motion Matching, achieving state-of-the-art results in terms of quality, performance, and memory usage in multiple domains.

ACKNOWLEDGMENTS

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC). We also would like to thank Mathieu Gendron-Denis for building our test environment, the Ubisoft Shanghai Wildlife Team for supplying the bear data, and everyone who took part in the User Study.

REFERENCES

Okan Arikan and D. A. Forsyth. 2002. Interactive Motion Generation from Examples. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (San Antonio, Texas) *(SIGGRAPH '02)*. ACM, New York, NY, USA, 483–490. https://doi.org/10.1145/566570.566606

S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D.S. Seljebotn, and K. Smith. 2011. Cython: The Best of Both Worlds. *Computing in Science Engineering* 13, 2 (2011), 31 –39. https://doi.org/10.1109/MCSE.2010.118

Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. DReCon: Data-driven Responsive Control of Physics-based Characters. *ACM Trans. Graph.* 38, 6, Article 206 (Nov. 2019), 11 pages. https://doi.org/10.1145/3355089.3356536

David Bollo. 2016. Inertialization: High-Performance Animation Transitions in 'Gears of War'. In *Proc. of GDC 2018.*

David Bollo. 2017. High Performance Animation in Gears of War 4. In *ACM SIGGRAPH 2017 Talks* (Los Angeles, California) *(SIGGRAPH '17)*. ACM, New York, NY, USA, Article 22, 2 pages. https://doi.org/10.1145/3084363.3085069

Michael Büttner. 2019. Machine Learning for Motion Synthesis and Character Control in Games. In *Proc. of i3D 2019.*

Michael Büttner and Simon Clavet. 2015. Motion Matching - The Road to Next Gen Animation. In *Proc. of Nucl.ai 2015.* https://www.youtube.com/watch?v=z_wpgHFSWss&t=658s

Jinxiang Chai and Jessica K. Hodgins. 2005. Performance Animation from Low-dimensional Control Signals. In *ACM SIGGRAPH 2005 Papers* (Los Angeles, California) *(SIGGRAPH '05)*. ACM, New York, NY, USA, 686–696. https://doi.org/10.1145/1186822.1073248

Simon Clavet. 2016. Motion Matching and The Road to Next-Gen Animation. In *Proc. of GDC 2016.*

Katerina Fragkiadaki, Sergey Levine, and Jitendra Malik. 2015. Recurrent Network Models for Kinematic Tracking. *CoRR* abs/1508.00271 (2015). arXiv:1508.00271 http://arxiv.org/abs/1508.00271

Nicholas Frechette. 2019. Animation Compression Library. (2019). https://nfrechette.github.io/

Keith Grochow, Steven L. Martin, Aaron Hertzmann, Aaron Hertzmann, and Zoran Popović. 2004. Style-based Inverse Kinematics. In *ACM SIGGRAPH 2004 Papers* (Los Angeles, California) *(SIGGRAPH '04)*. ACM, New York, NY, USA, 522–531. https://doi.org/10.1145/1186562.1015755

Geoff Harrower. 2018. Real Player Motion Tech in 'EA Sports UFC 3'. In *Proc. of GDC 2018.*

Félix G Harvey and Christopher Pal. 2018. Recurrent transition networks for character locomotion. In *SIGGRAPH Asia 2018 Technical Briefs*. ACM, 4.

Rachel Heck and Michael Gleicher. 2007. Parametric Motion Graphs. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (Seattle, Washington) *(I3D '07)*. ACM, New York, NY, USA, 129–136. https://doi.org/10.1145/1230100.1230123

Gustav Eje Henter, Simon Alexanderson, and Jonas Beskow. 2019. MoGlow: Probabilistic and controllable motion synthesis using normalising flows. *CoRR* abs/1905.06598 (2019). arXiv:1905.06598 http://arxiv.org/abs/1905.06598

Daniel Holden. 2018. Character Control with Neural Networks and Machine Learning. In *Proc. of GDC 2018.*

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned Neural Networks for Character Control. *ACM Trans. Graph.* 36, 4, Article 42 (July 2017), 13 pages. https://doi.org/10.1145/3072959.3073663

Daniel Holden, Jun Saito, and Taku Komura. 2016. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Trans. Graph.* 35, 4, Article 138 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925975

Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. 2015. Learning Motion Manifolds with Convolutional Autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs* (Kobe, Japan) *(SA '15)*. ACM, New York, NY, USA, Article 18, 4 pages. https://doi.org/10.1145/2820903.2820918

Seokpyo Hong, Daseong Han, Kyungmin Cho, Joseph S. Shin, and Junyong Noh. 2019. Physics-based Full-body Soccer Motion Control for Dribbling and Shooting. *ACM Trans. Graph.* 38, 4, Article 74 (July 2019), 12 pages. https://doi.org/10.1145/3306346.3322963

David Hunt, Richard Lico, and Michael Buttner. 2018. Topics in Real-time Animation. In *ACM SIGGRAPH 2018 Courses* (Vancouver, British Columbia, Canada) *(SIGGRAPH '18)*. ACM, New York, NY, USA, Article 17, 1 pages. https://doi.org/10.1145/3214834.3214882

Tamoor Hussain. 2019. The Last Of Us 2 Has An Awesome Improvement You Might Not Have Noticed. (2019). https://www.gamespot.com/articles/the-last-of-us-2-has-an-awesome-improvement-you-mi/1100-6470118/?utm_source=reddit.com

Kyunglyul Hyun, Kyungho Lee, and Jehee Lee. 2016. Motion Grammars for Character Animation. In *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics* (Lisbon, Portugal) *(EG '16)*. Eurographics Association, Goslar Germany, Germany, 103–113. https://doi.org/10.1111/cgf.12815

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE transactions on pattern analysis and machine intelligence* 33 (01 2011), 117–28. https://doi.org/10.1109/TPAMI.2010.57

Andrew Kermse. 2004. Game Programming Gems 4. (2004), 95–101.

Lucas Kovar, Michael Gleicher, and Frédéric Pighin. 2002. Motion Graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (San Antonio, Texas) *(SIGGRAPH '02)*. ACM, New York, NY, USA, 473–482. https://doi.org/10.1145/566570.566605

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2019. Large Memory Layers with Product Keys. *CoRR* abs/1907.05242 (2019). arXiv:1907.05242 http://arxiv.org/abs/1907.05242

Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. 2002. Interactive Control of Avatars Animated with Human Motion Data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (San Antonio, Texas) *(SIGGRAPH '02)*. ACM, New York, NY, USA, 491–500. https://doi.org/10.1145/566570.566607

Jehee Lee and Kang Hoon Lee. 2004. Precomputing Avatar Behavior from Human Motion Data. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Grenoble, France) *(SCA '04)*. Eurographics Association, Goslar Germany, Germany, 79–87. https://doi.org/10.1145/1028523.1028535

Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018. Interactive Character Animation by Learning Multi-objective Control. *ACM Trans. Graph.* 37, 6, Article 180 (Dec. 2018), 10 pages. https://doi.org/10.1145/3272127.3275071

Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. 2010. Motion Fields for Interactive Character Locomotion. In *ACM SIGGRAPH Asia 2010 Papers* (Seoul, South Korea) *(SIGGRAPH ASIA '10)*. ACM, New York, NY, USA, Article 138, 8 pages. https://doi.org/10.1145/1866158.1866160

Sergey Levine, Jack M. Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. 2012. Continuous Character Control with Low-dimensional Embeddings. *ACM Trans. Graph.* 31, 4, Article 28 (July 2012), 10 pages. https://doi.org/10.1145/2185520.2185524

Yanran Li, Zhao Wang, Xiaosong Yang, Meili Wang, Sebastian Iulian Poiana, Ehtzaz Chaudhry, and Jianjun Zhang. 2019. Efficient convolutional hierarchical autoencoder for human motion prediction. *The Visual Computer* 35, 6 (01 Jun 2019), 1143–1156. https://doi.org/10.1007/s00371-019-01692-9

Zimo Li, Yi Zhou, Shuangjiu Xiao, Chong He, and Hao Li. 2017. Auto-Conditioned LSTM Network for Extended Complex Human Motion Synthesis. *CoRR* abs/1707.05363 (2017). arXiv:1707.05363 http://arxiv.org/abs/1707.05363

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2019. On the Variance of the Adaptive Learning Rate and Beyond. arXiv:cs.LG/1908.03265

Wan-Yen Lo and Matthias Zwicker. 2008. Real-time Planning for Parameterized Human Motion. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Dublin, Ireland) *(SCA '08)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 29–38. http://dl.acm.org/citation.cfm?id=1632592.1632598

Mark Miller, Daniel Holden, Rami Al-Ashqar, Christophe Dubach, Kenny Mitchell, and Taku Komura. 2015. Carpet Unrolling for Character Control on Uneven Terrain. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games* (Paris, France) *(MIG '15)*. Association for Computing Machinery, New York, NY, USA, 193–198. https://doi.org/10.1145/2822013.2822031

Jianyuan Min and Jinxiang Chai. 2012. Motion Graphs++: A Compact Generative Model for Semantic Motion Analysis and Synthesis. *ACM Trans. Graph.* 31, 6, Article 153 (Nov. 2012), 12 pages. https://doi.org/10.1145/2366145.2366172

Tomohiko Mukai. 2011. Motion Rings for Interactive Gait Synthesis. In *Symposium on Interactive 3D Graphics and Games* (San Francisco, California) *(I3D '11)*. ACM, New York, NY, USA, 125–132. https://doi.org/10.1145/1944745.1944767

Tomohiko Mukai and Shigeru Kuriyama. 2005. Geostatistical Motion Interpolation. In *ACM SIGGRAPH 2005 Papers* (Los Angeles, California) *(SIGGRAPH '05)*. ACM, New York, NY, USA, 1062–1070. https://doi.org/10.1145/1186822.1073313

Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. 2019. Learning Predict-and-Simulate Policies from Unorganized Human Motion Data. *ACM Trans. Graph.* 38, 6, Article Article 205 (Nov. 2019), 11 pages. https://doi.org/10.1145/3355089.3356501

Sang Il Park, Hyun Joon Shin, and Sung Yong Shin. 2002. On-line Locomotion Generation Based on Motion Blending. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Antonio, Texas) *(SCA '02)*. ACM, New York, NY, USA, 105–111. https://doi.org/10.1145/545261.545279

Dario Pavllo, Christoph Feichtenhofer, Michael Auli, and David Grangier. 2019. Modeling Human Motion with Quaternion-based Neural Networks. *CoRR* abs/1901.07677 (2019). arXiv:1901.07677 http://arxiv.org/abs/1901.07677

Dario Pavllo, David Grangier, and Michael Auli. 2018. QuaterNet: A Quaternion-based Recurrent Model for Human Motion. *CoRR* abs/1805.06485 (2018). arXiv:1805.06485 http://arxiv.org/abs/1805.06485

Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. 1998. Verbs and Adverbs: Multidimensional Motion Interpolation. *IEEE Comput. Graph. Appl.* 18, 5 (Sept. 1998), 32–40. https://doi.org/10.1109/38.708559

Alla Safonova and Jessica K. Hodgins. 2007. Construction and Optimal Search of Interpolated Motion Graphs. In *ACM SIGGRAPH 2007 Papers* (San Diego, California) *(SIGGRAPH '07)*. ACM, New York, NY, USA, Article 106. https://doi.org/10.1145/1275808.1276510

Hyun Joon Shin and Hyun Seok Oh. 2006. Fat Graphs: Constructing an Interactive Character with Continuous Controls. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Vienna, Austria) *(SCA '06)*. Eurographics Association, Goslar, DEU, 291–298.

Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. 2019. Neural State Machine for Character-scene Interactions. *ACM Trans. Graph.* 38, 6, Article 209 (Nov. 2019), 14 pages. https://doi.org/10.1145/3355089.3356505

Jochen Tautges, Arno Zinke, Björn Krüger, Jan Baumann, Andreas Weber, Thomas Helten, Meinard Müller, Hans-Peter Seidel, and Bernd Eberhardt. 2011. Motion Reconstruction Using Sparse Accelerometer Data. *ACM Trans. Graph.* 30, 3, Article 18 (May 2011), 12 pages. https://doi.org/10.1145/1966394.1966397

Graham W. Taylor and Geoffrey E. Hinton. 2009. Factored Conditional Restricted Boltzmann Machines for Modeling Motion Style. In *Proceedings of the 26th Annual International Conference on Machine Learning* (Montreal, Quebec, Canada) *(ICML '09)*. ACM, New York, NY, USA, 1025–1032. https://doi.org/10.1145/1553374.1553505

Adrien Treuille, Yongjoon Lee, and Zoran Popović. 2007. Near-Optimal Character Animation with Continuous Control. In *ACM SIGGRAPH 2007 Papers* (San Diego, California) *(SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 7–es. https://doi.org/10.1145/1275808.1276386

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. 2008. Gaussian Process Dynamical Models for Human Motion. *IEEE Trans. Pattern Anal. Mach. Intell.* 30, 2 (Feb. 2008), 283–298. https://doi.org/10.1109/TPAMI.2007.1167

Zhiyong Wang, Jinxiang Chai, and Shihong Xia. 2018. Combining Recurrent Neural Networks and Adversarial Training for Human Motion Synthesis and Control. *CoRR* abs/1806.08666 (2018). arXiv:1806.08666 http://arxiv.org/abs/1806.08666

Gwonjin Yi and Junghoon Jee. 2019. Search Space Reduction In Motion Matching by Trajectory Clustering. In *SIGGRAPH Asia 2019 Posters* (Brisbane, QLD, Australia) *(SA '19)*. ACM, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.1145/3355056.3364558

Kristjan Zadziuk. 2016. Motion Matching, The Future of Games Animation... Today. In *Proc. of GDC 2016.*

He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-adaptive Neural Networks for Quadruped Motion Control. *ACM Trans. Graph.* 37, 4, Article 145 (July 2018), 11 pages. https://doi.org/10.1145/3197517.3201366

Fabio Zinno. 2019. ML Tutorial Day: From Motion Matching to Motion Synthesis, and All the Hurdles In Between. In *Proc. of GDC 2019.*

Victor Brian Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. 2005. Dynamic Response for Motion Capture Animation. *ACM Trans. Graph.* 24, 3 (July 2005), 697–701. https://doi.org/10.1145/1073204.1073249

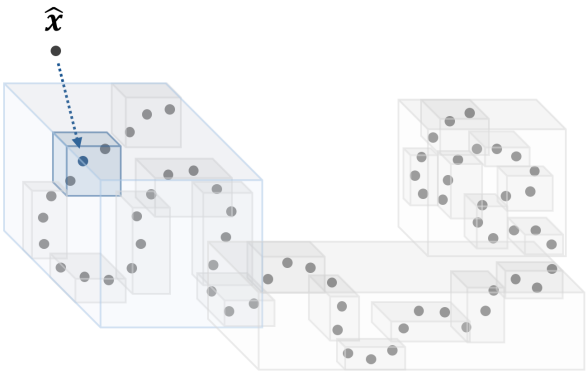

Fig. 15. Visual description of our search acceleration structure. We fit axis-aligned bounding boxes to groups of consecutive frames in the Matching Database $\mathbf{X}$. Search is first performed against these bounding boxes before continuing on to what is inside.

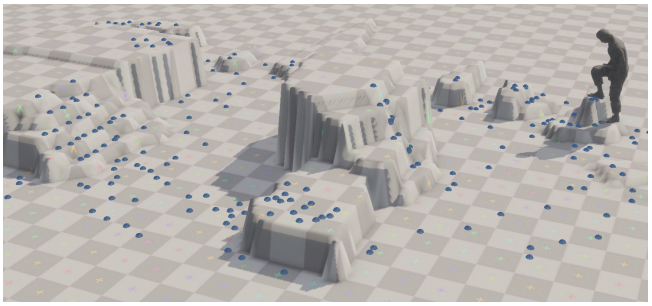

Fig. 16. Visual example of the height map function we extract from our animation data with contact points shown in blue.

A CLIP STRUCTURE

For simplicity of presentation, in Section 3 we assume $\mathbf{X}$ and $\mathbf{Y}$ consist of a single large continuous animation, but in reality animation databases consists of multiple individual clips. In our case, we pack all animation and matching data into single large matrices $\mathbf{X}$ and $\mathbf{Y}$, but store alongside it `(start, end, anim_index)` tuples describing the individual clips within. Generally, some care must be taken when using this format such as when computing velocities via finite difference or constructing windows for training. We also use this additional information to trigger a search when playback comes to the end of a clip, and disallow the nearest neighbor search from returning a frame close to the end of a clip as this can result in re-triggering the search too frequently.

B SEARCH ACCELERATION

Rather than a KD-Tree or clustering-based approach we use a simple axis-aligned bounding-box (AABB) based method to accelerate the nearest neighbor search. We fit axis-aligned bounding boxes to groups of 16 and 64 frames consecutively in $\mathbf{X}$ (see Fig 15 for a visual description).

For each AABB we find the distance from the query point to the nearest point inside the AABB. If this distance is larger than the smallest distance so far then no point inside the AABB will have a smaller distance than our current best, and therefore there is no need to check inside.

We found that axis-aligned bounding boxes had a number of interesting advantages for this task. Firstly, as we iterate over the database in order, we have excellent cache performance and avoid the random access that can occur using structures such as KD-Trees. Secondly, the squared distance to an AABB can be computed as a sum of the squared distance along each dimension individually. This allows for an essential form of early-out in the search, as the accumulated distance to an AABB along just a few dimensions will often quickly exceed the distance to the best match found so far. Finally, axis-aligned bounding boxes are simple to use and require a minimal amount of memory overhead. In our experiments we found two levels of hierarchy with the sizes described above enough to greatly accelerate the search. To accelerate training as well as runtime we implement this same algorithm in both C++ and Cython [Behnel et al. 2011].

C FORWARD KINEMATICS

The Forward Kinematics function used in Algorithm 1 can be defined recursively as follows:

$$\text{ForwardKinematics}(\mathbf{Y}) = \{\mathbf{q}^t\ \mathbf{q}^r\ \dot{\mathbf{q}}^t\ \dot{\mathbf{q}}^r\}, \tag{2}$$

$$\mathbf{q}_j^t = \mathbf{q}_p^t + \mathbf{q}_p^r \otimes \mathbf{y}_j^t, \tag{3}$$

$$\mathbf{q}_j^r = \mathbf{q}_p^r \otimes \mathbf{y}_j^r, \tag{4}$$

$$\dot{\mathbf{q}}_j^t = \dot{\mathbf{q}}_p^t + \mathbf{q}_p^r \otimes \dot{\mathbf{y}}_j^t + \dot{\mathbf{q}}_j^r \times (\mathbf{q}_p^r \otimes \mathbf{y}_j^t), \tag{5}$$

$$\dot{\mathbf{q}}_j^r = \dot{\mathbf{q}}_p^r + \mathbf{q}_p^r \otimes \dot{\mathbf{y}}_j^r, \tag{6}$$

where j is the joint index, p is the joint index of the parent joint, and $\otimes$ represents quaternion-quaternion multiplication or quaternion-vector product when multiplying by a vector. For joints with no parent such as the root joint, the local rotation and translation of that joint are taken as-is without transformation.

D TERRAIN FITTING

Because our method avoids extrapolation, and the input features we use to match terrain heights are more sparse than in Holden et al. [2017] we can use a simpler terrain fitting procedure without worrying about over-fitting. In our case we extract all the foot contact points, and fit a nearest neighbor regression which maps from the xy position, to the height z. Given a new 2D position we can then get the new height using this regression function. This produces a "stepped" style terrain as shown in Fig 16.