DATA-DRIVEN SYNTHESIS ON MOTION FIELDS

DUARTE DAVID

Based on Lee, Y., et al ,“Motion Fields for Interactive Character Animation”

Motivation

Motions Graphs were great!

… but they only “played” predefined clips of animation.

We want something more flexible!

That works in a continuous setting!

Outline

- What are Motion Fields?
- Crash Course on Reinforcement learning
- Guiding the motion synthesis with the motion field
- Further Remarks
- Results and discussion

Motion Space

AKA “Phase Space” or “State Space”

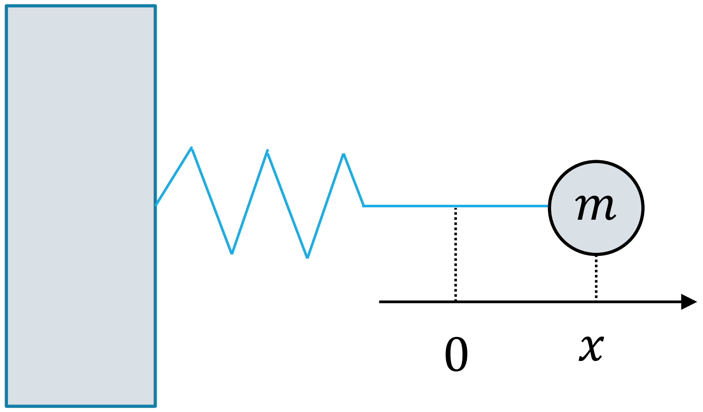
A point in the motion space represents the state (pose + velocity) of the character.

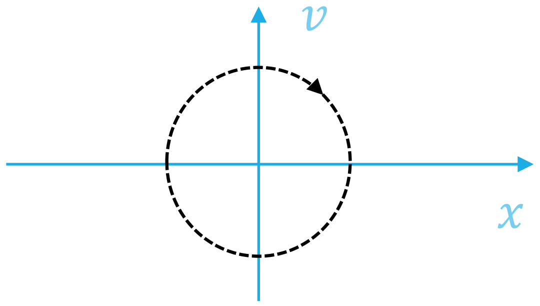
Motion Space

Simple Example: Spring

$$x''(t) = -\frac{k}{m}x(t)$$

Point in phase space: $(x, v), x' = v$





Motion Space

$$x = (x_{root}, p_0, p_1, \dots, p_n)$$

$$\mathrm{v} = (v_{root}, q_0, q_1, \dots, q_n)$$

$$m = (x, v)$$

We can define “addition” and “subtraction” of poses:

$$x' \oplus x = (x'_{root} + x_{root}, p_0 p'_0, p_1 p'_1, \dots, p_n p'_n)$$
$$x' \ominus x = \left(x'_{root} - x_{root}, p'{}_0 p_0^{-1}, p'_1 p_1^{-1}, \dots, p'_n p_n^{-1}\right)$$

v is computed via finite differences:

- $m_i = (x_i, x_{i+1} \ominus x_i)$, for the i-th frame in the database
- $y_i = x_{i+2} \ominus x_{i+1}$

Motion Space

Distance metric:

$$d(m, m') = \sqrt{\begin{array}{rr} \beta_{\text{root}} \| v_{\text{root}} - v'_{\text{root}} \|^2 & + \\ \beta_0 \| q_0(\hat{u}) - q'_0(\hat{u}) \|^2 & + \\ \sum_{i=1}^{n} \beta_i \| p_i(\hat{u}) - p'_i(\hat{u}) \|^2 & + \\ \sum_{i=1}^{n} \beta_i \| (q_i p_i)(\hat{u}) - (q'_i p'_i)(\hat{u}) \|^2 & + \end{array}}$$

β are adjustable parameters

$\beta_{root} = 0.5, \beta_0 = 0.5, \beta_i$ is set to the length of bone *i*

Motion Field

$A(m)$ = set of actions

$(a_1, a_2, \dots, a_k) = \ a \in A(m)$

$\sum_{i=1}^{k} a_i = 1, \ a_i \geq 0$

The vector a describes the influence of each of the k nearest neighbors

(It's a convex combination)

Motion Synthesis

$$I(m, a) = (x \oplus v', y')$$

$$v' = \sum_{i=1}^{k}(a_i v_i)$$

$$y' = \sum_{i=1}^{k}(a_i y_i)$$

Motion Synthesis

$$I(m, a) = (x + v', y')$$

$$v' = (1 - \delta) \sum_{i=1}^{k} (a_i v_i) + \delta((\bar{x} \oplus \bar{v}) \ominus x)$$

$$y' = (1 - \delta) \sum_{i=1}^{k} (a_i y_i) + \delta \bar{y}$$

We add a tug that pulls toward closest state $\overline{m}$

Motion Synthesis

How do we choose the action a?

Should we set $a_i = w_i$?

(Where w_i are the similarity weights, $w_i = \frac{1}{\eta} \frac{1}{d(m,m_i)^2}$)

$$\eta = \sum_i \frac{1}{d(m, m_i)^2}$$

Reinforcement Learning

We have **states**…

And our motion field describes the **actions…**

That cause our state to **transition** to another state…

While fulfilling a certain goal (i.e.: maximizing a **reward**)

We want to find the best action (**policy**)

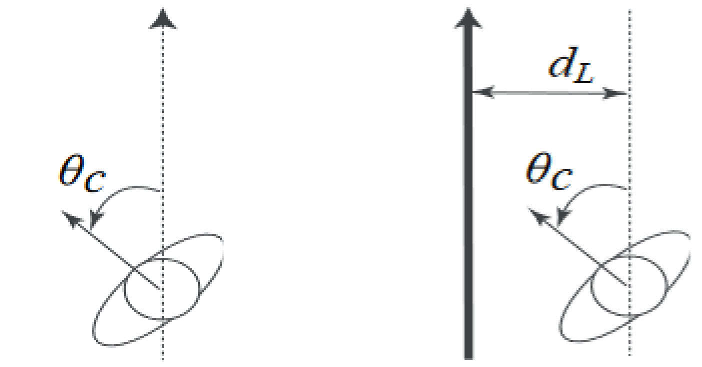
These 4 key items define a **Markov Decision Process**

Markov Decision Process

States:

- m is not enough!
- We need to represent how well the state is fulfilling our task
- We consider the state $s = (m, \theta_t)$,

θ_t is the vector of **task parameters**



(a) Direction (b) Line Following

Markov Decision Process

Actions:

- We discretize the set of possible actions
- Instead of considering all possible actions, we simply consider ***k*** different actions (if we have k nearest neighbors):

$$a^n = \frac{(w_1,\ldots,w_{n-1},1,w_{n+1},\ldots,w_k)}{\sum_{i=1,i\neq n}^{k} w_i \; + \; 1}$$

Markov Decision Process

Transitions:

$$s_{i+1} = I_s(s_i, a_i)$$

$$I_s(s, a) = (m', \theta') \\ = (I(m, a), \theta(m'))$$

Note: The indices here represent the "timestep" in which we generate the state, not their position in the original animation here.

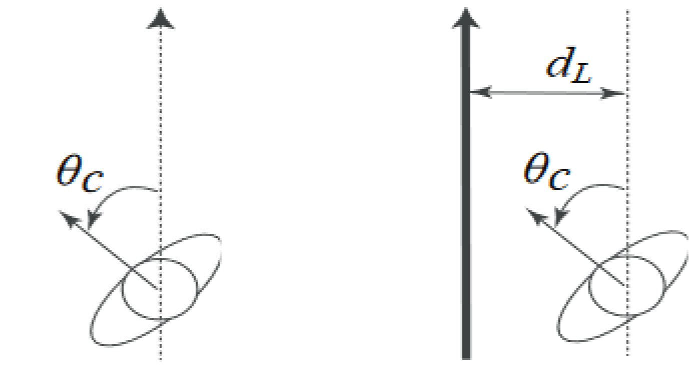
Markov Decision Process

Reward Function:

- $R(s, a)$

Examples:

$$R_{direction}(s, a) = -|\theta_c|$$
$$R_{line}(s, a) = -|\theta_c| - 0.05|d_L|$$



(a) Direction (b) Line Following

Reinforcement Learning

Greedy Policy:

$$\pi(s) = argmax_{a \in A(m)} R(s, a)$$

Reinforcement Learning

Lookahead Policy:

$$\pi(s) = argmax_{a \in A(m)} [R(s, a) + \max_{\{a_t\}} \sum_{t=1}^{\infty} \gamma^t R(s, a)]$$

γ is the discount factor.

Reinforcement Learning

$$V(s) = \max_{\{a_t\}} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)$$

$$V(s) = R\big(s, \pi(s)\big) + \gamma V\Big(I_s\big(s, \pi(s)\big)\Big)$$

$$\pi(s) = argmax_{a \in A(m)}\Big[R(s,a) + \gamma V\big(I_s(s,a)\big)\Big]$$

V is evaluated at points of the form (m_i, θ_j), where m_i are the motion states associated with the data, and θ_j are sample points in the task parameters (i.e.: a regular grid)

Value Iteration Algortihm:

- Initialize V to zero
- While (V changes)
  - Compute $\pi(s)$
  - Update V

$V\big(I_s(s,a)\big)$ is computed via interpolation

Motion Synthesis (again)

Given a current task state s

Compute k states $s_i = I_s(s, a^i)$, a^i is the action that favors the *i*-th nearest neighbor.

For each action, compute the reward + value of next state:

$$R(s, a^i) + \gamma V(s_i)$$

Choose the action that maximizes that sum.

Going Further

- Foot skate clean up
- Temporal Compression
- Response to perturbation

Foot-skate cleanup

For every motion state m_i , store if the left foot is in contact ($l_{contact}(m_i) = 1$) or not ($l_{contact}(m_i) = 0$)

At any given motion state, we have that

$$l_{\text{contact}}(m) = \sum_{m_i \in \mathcal{N}(m)} w_i l_{\text{contact}}(m_i)$$

If $l_{contact}(m_i) > 0.5$, we hold the foot in place and pose the leg bones with an IK solution. Otherwise, we blend out of the IK solution in 0.2 seconds.

We do the same for the right foot.

Temporal Compression

If we have too many task parameters, the size of the value function may quickly grow!

Solution: states that follow each other in the motion database should be similar enough. Compute and store V only for every N-th frame (anchor states), interpolate in time for the other frames.

$$V_{m_i}(\theta_T) = \frac{N-i}{N} V_{m_0}(\theta_T) + \frac{i}{N} V_{m_N}(\theta_T)$$

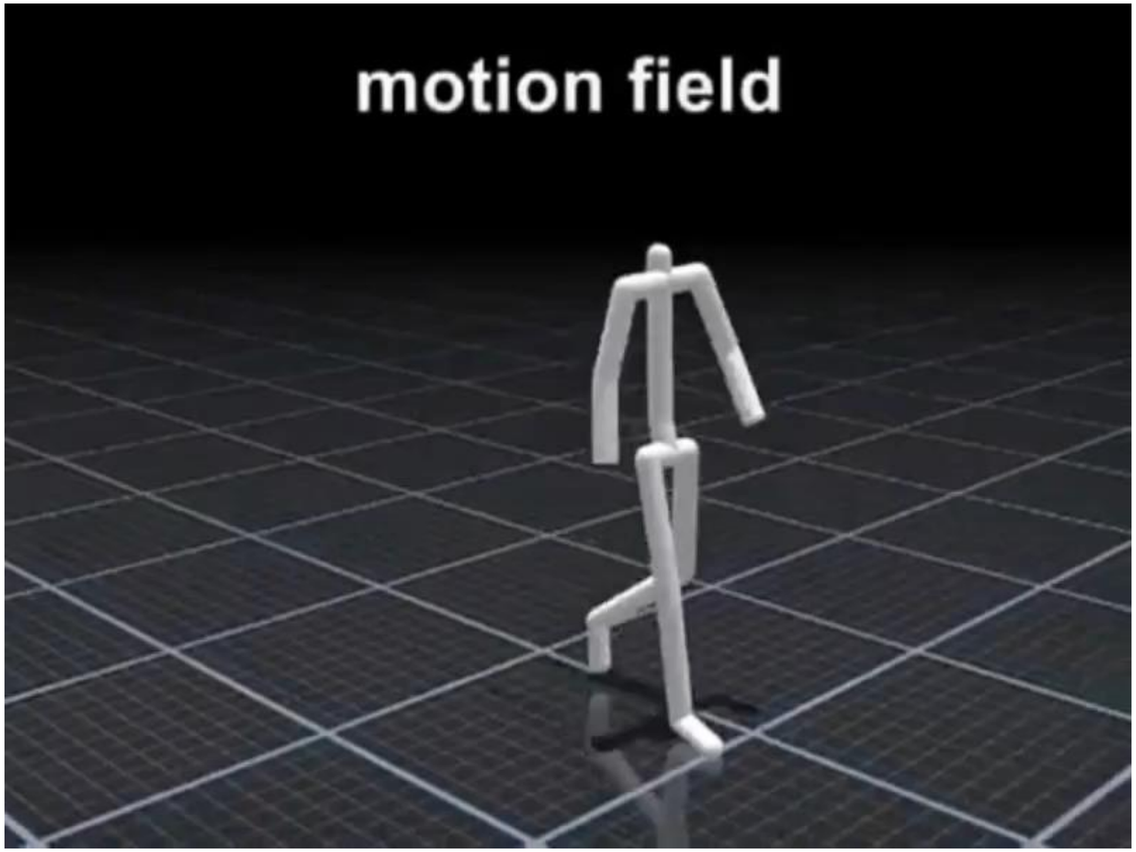
Response to Perturbation

At any given point in time, blend into a dynamics simulation:

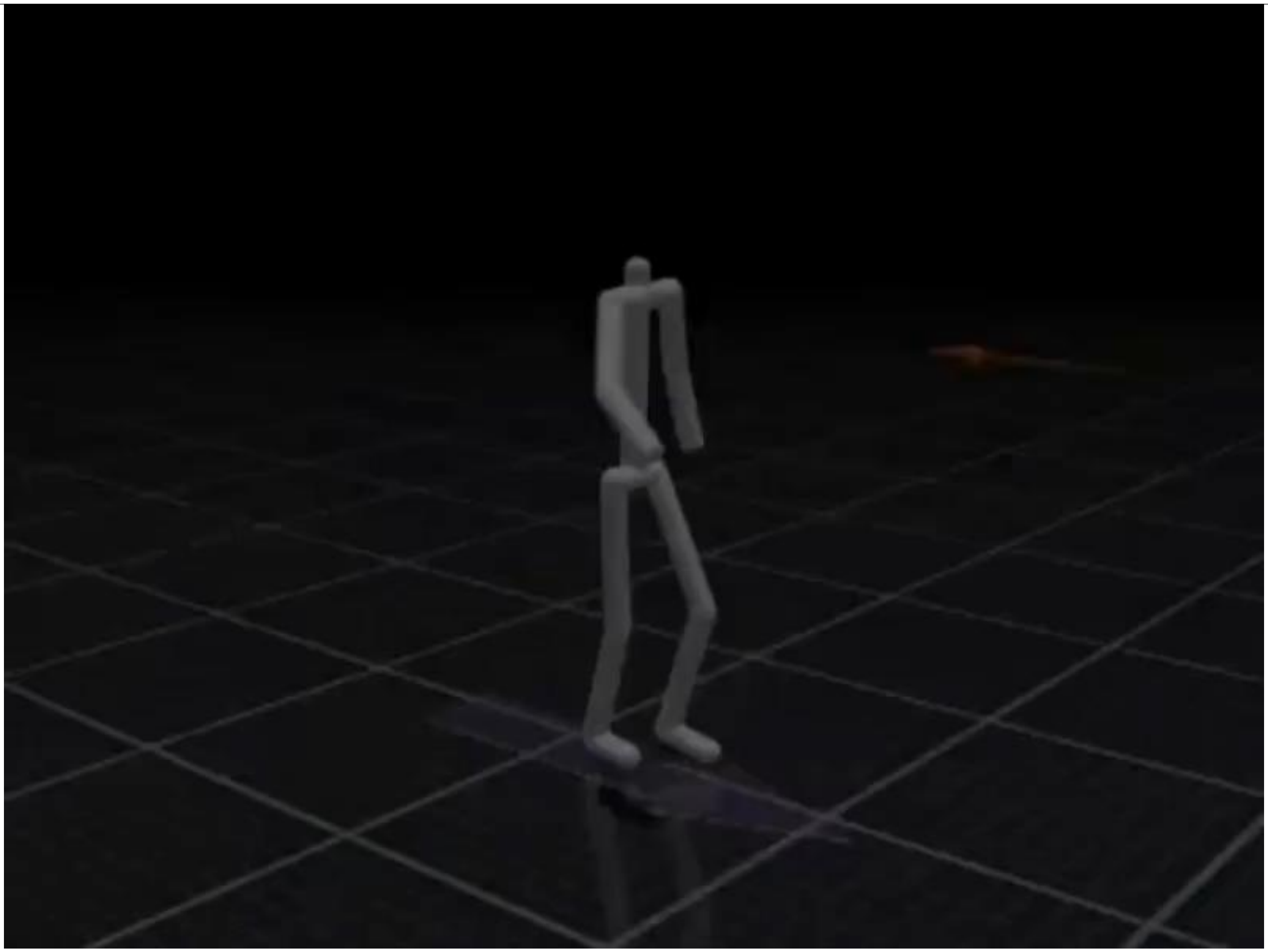
$$\mathcal{I}_D(x, v, a) = \frac{i}{k}\mathcal{I}(x, v, a) + \frac{k-i}{k}D(x, 0, i)$$

When the blending stops, go back to the normal integration function.

Results

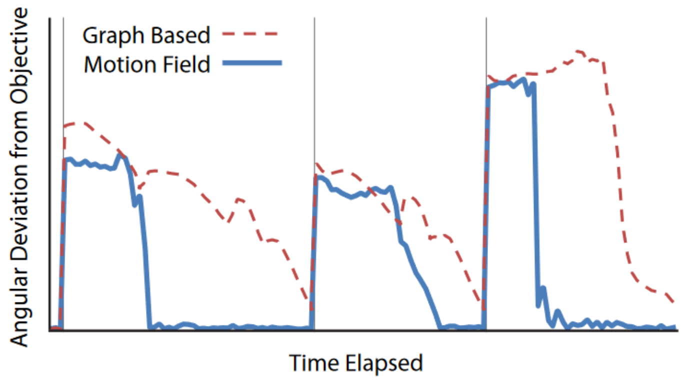


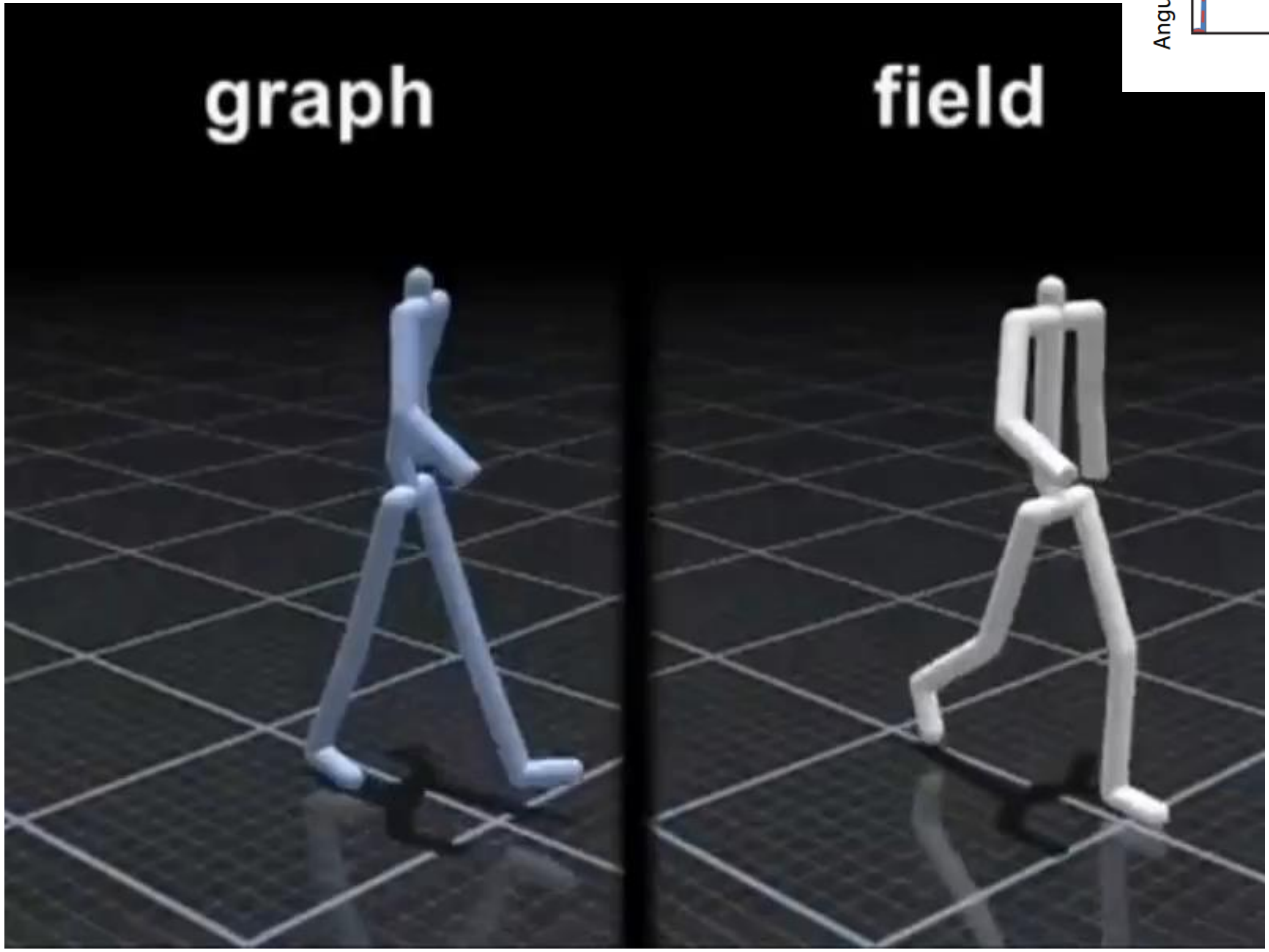
Results Perturbation response



Results

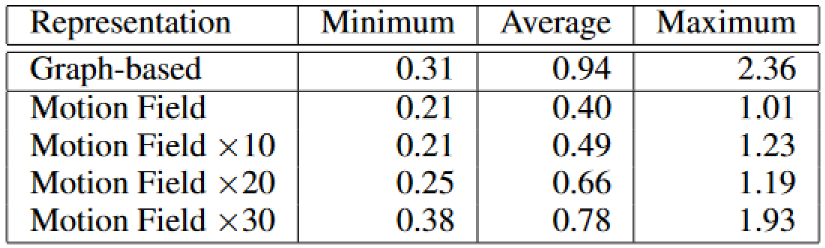
Comparison to motion graphs:





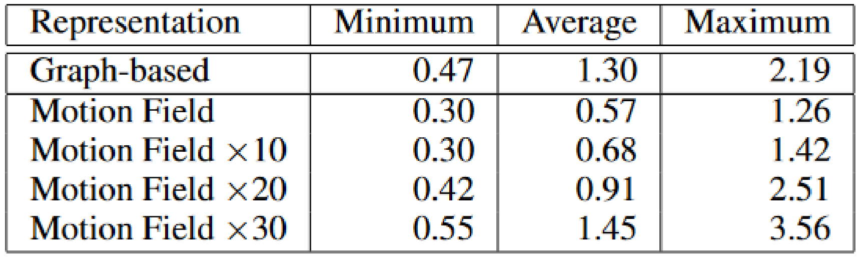
Results

Temporal Compression



Representation	Minimum	Average	Maximum
Graph-based	0.31	0.94	2.36
Motion Field	0.21	0.40	1.01
Motion Field $\times 10$	0.21	0.49	1.23
Motion Field $\times 20$	0.25	0.66	1.19
Motion Field $\times 30$	0.38	0.78	1.93

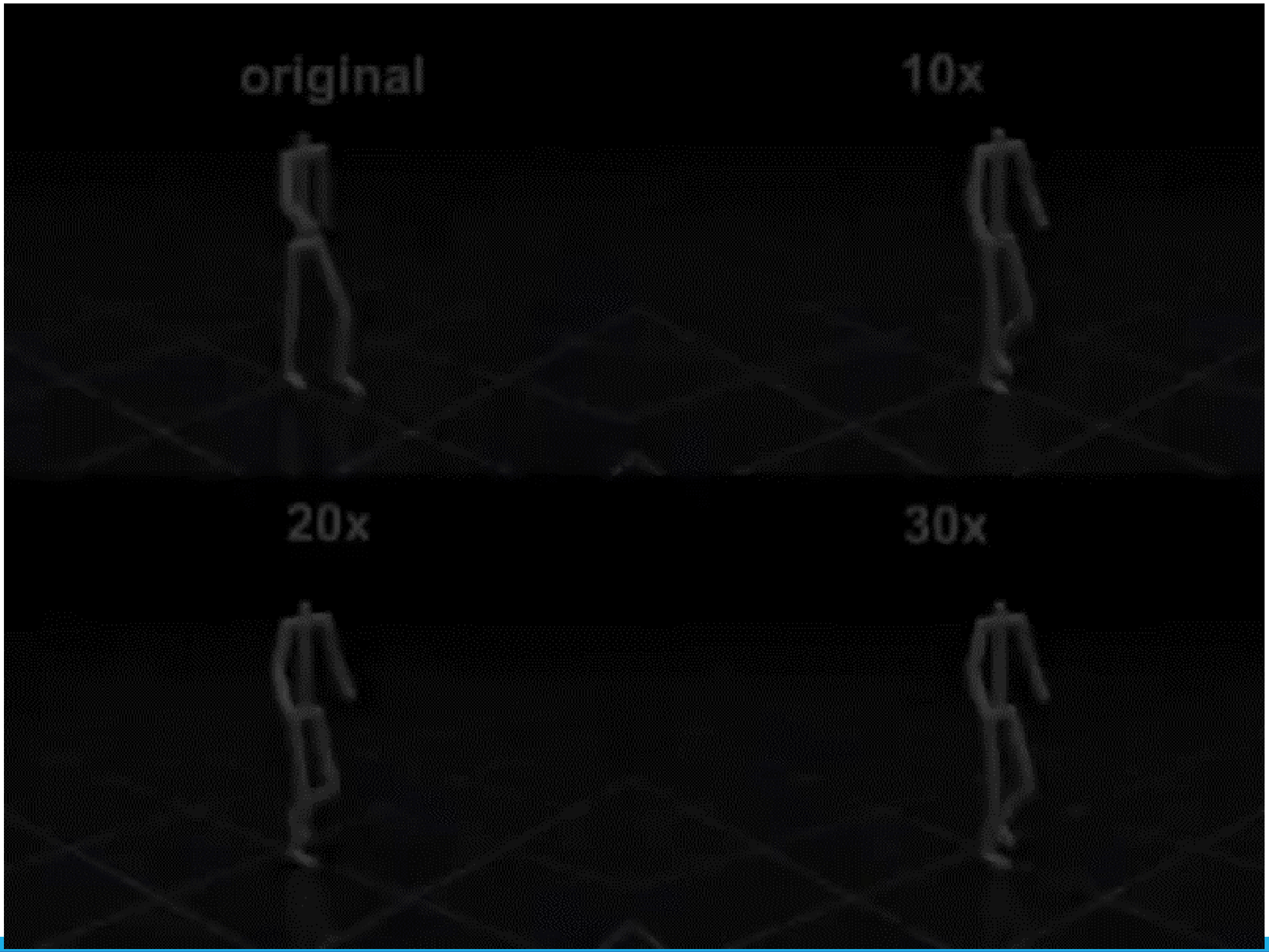
Table 1: *Response times in seconds for the direction task until converging within 5 degrees of desired direction. Motion Field $\times 10$ denotes ten-fold temporally compressed value function on the motion field ($N = 10$). Motion Field $\times 20$ and $\times 30$ are defined similarly. A motion field with thirty-fold temporal compression has agility similar to graph-based control, while even a twenty-fold compression is significantly more responsive than the graph-based alternative.*



Representation	Minimum	Average	Maximum
Graph-based	0.47	1.30	2.19
Motion Field	0.30	0.57	1.26
Motion Field $\times 10$	0.30	0.68	1.42
Motion Field $\times 20$	0.42	0.91	2.51
Motion Field $\times 30$	0.55	1.45	3.56

Table 2: *Response times in seconds for the line following task until converging within 5 degrees of desired direction and 0.1 meters from the desired tracking line. In this two-dimensional control example, the twenty-fold compression is still more responsive than the graph-based control.*

Results Temporal Compression



In practice…

Is this used?

YES!

Example: Ubisoft uses a derivative of this method (highly simplified)

in their most recent videogames!

https://www.gdcvault.com/play/1023280/Motion-Matching-and-The-Road

https://www.gdcvault.com/play/1023478/Animation-Bootcamp-Motion-Matching-The

UBISOFT
TORONTO
@KrisZadziuk
- Motion Matching -
The Future of Games Animation... Today
- Kristjan Zadziuk - Animation Director - Ubisoft Toronto -

The end

Limitations?

The character can’t stray too far from data.
It’s harder to analyze and edit motion fields.
ANN search is too costly.
K-NN may not provide enough variety.

How can we improve?

Motion Matching heavily simplifies this method, and it has in turn inspired more complex ideas for control of interactive characters, like Phase Functioned Neural Networks.
Using different features for the motion space and better distance metrics is crucial.

Questions?

Extra

$$\text{Slerp}(p_0, p_1; t) = \frac{\sin[(1-t)\Omega]}{\sin\Omega} p_0 + \frac{\sin[t\Omega]}{\sin\Omega} p_1$$

$$\begin{aligned}
\text{Slerp}(q_0, q_1, t) &= q_0 (q_0^{-1} q_1)^t \\
&= q_1 (q_1^{-1} q_0)^{1-t} \\
&= (q_0 q_1^{-1})^{1-t} q_1 \\
&= (q_1 q_0^{-1})^t q_0
\end{aligned}$$

https://en.wikipedia.org/wiki/Slerp

Extra

$$M \triangleq \sum_{i=1}^{n} w_i \, \mathbf{q}_i \, \mathbf{q}_i^T$$

The averaged quaternion is given by the eigenvector of M corresponding to the largest eigenvalue

Markley, F.L., Cheng, Y., Crassidis, J.L., Oshman, Y., Quaternion Averaging

See also:

PARK, S. I., SHIN, H. J.,ANDSHIN, S. Y. 2002. On-line locomotion generation based on motion blending. Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer Animation,

Extra

For more on reinforcement learning:

Ernst, D., Geurts, P., Wehenkel, L., and Littman, L., 2005, Tree-Based Batch Mode Reinforcement Learning, jornal of Machine Learning Research 6