Motion Matching —— 概念与发展

本文档已上传SVN: /TD文档/说明讲解/${this}.pptx

Motion Matching —— 概念与发展

Table of Contents

- 基于状态机的动画
- Motion Matching的引出与基本概念
- Motion Matching的要点与对比
- Motion Matching的发展
- 小结

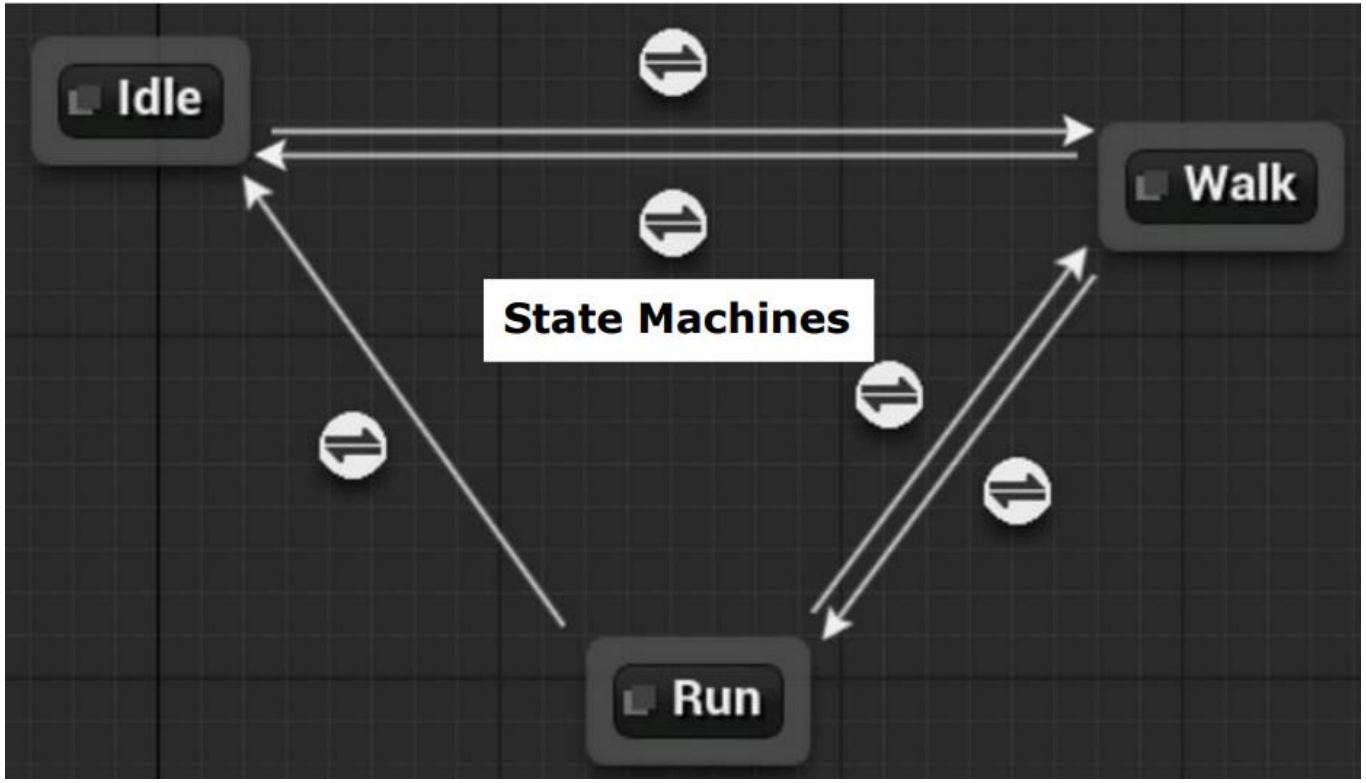
Motion Matching —— 概念与发展

Table of Contents

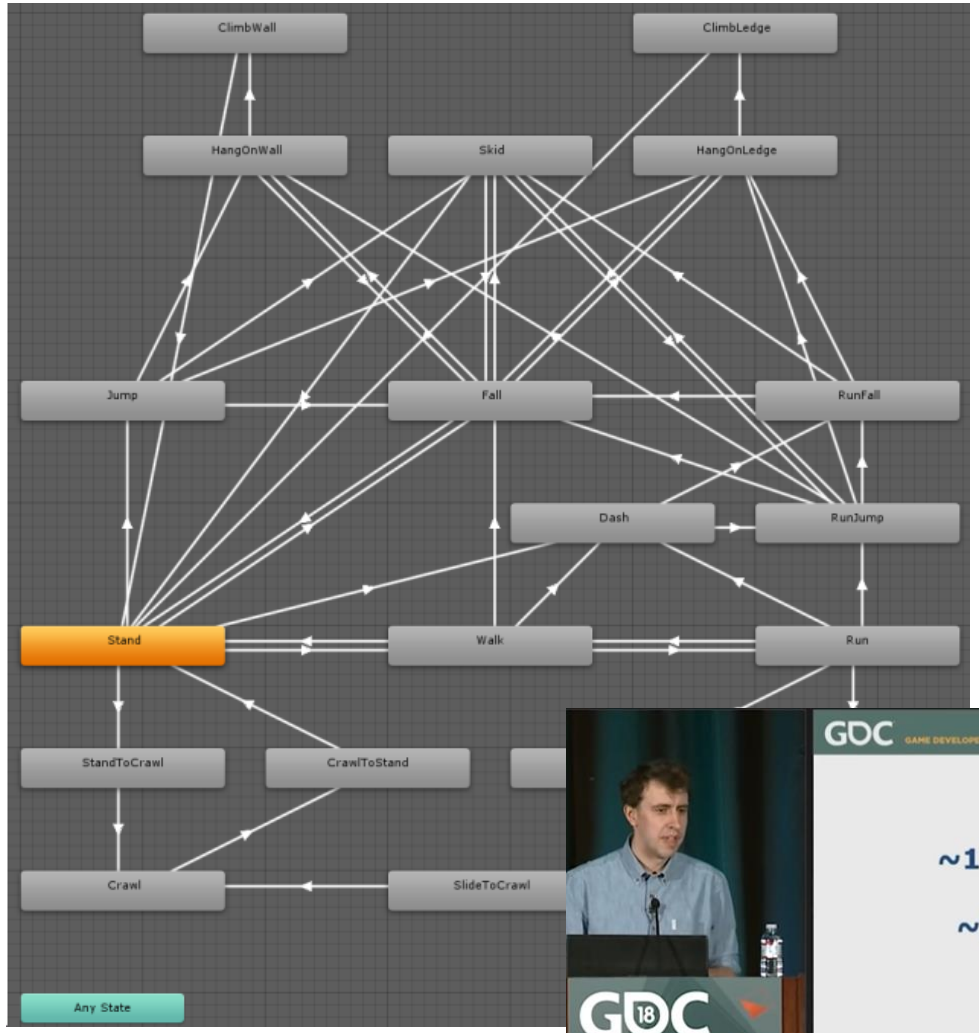
- 基于状态机的动画
- Motion Matching的引出与基本概念
- Motion Matching的要点与对比
- Motion Matching的发展
- 小结

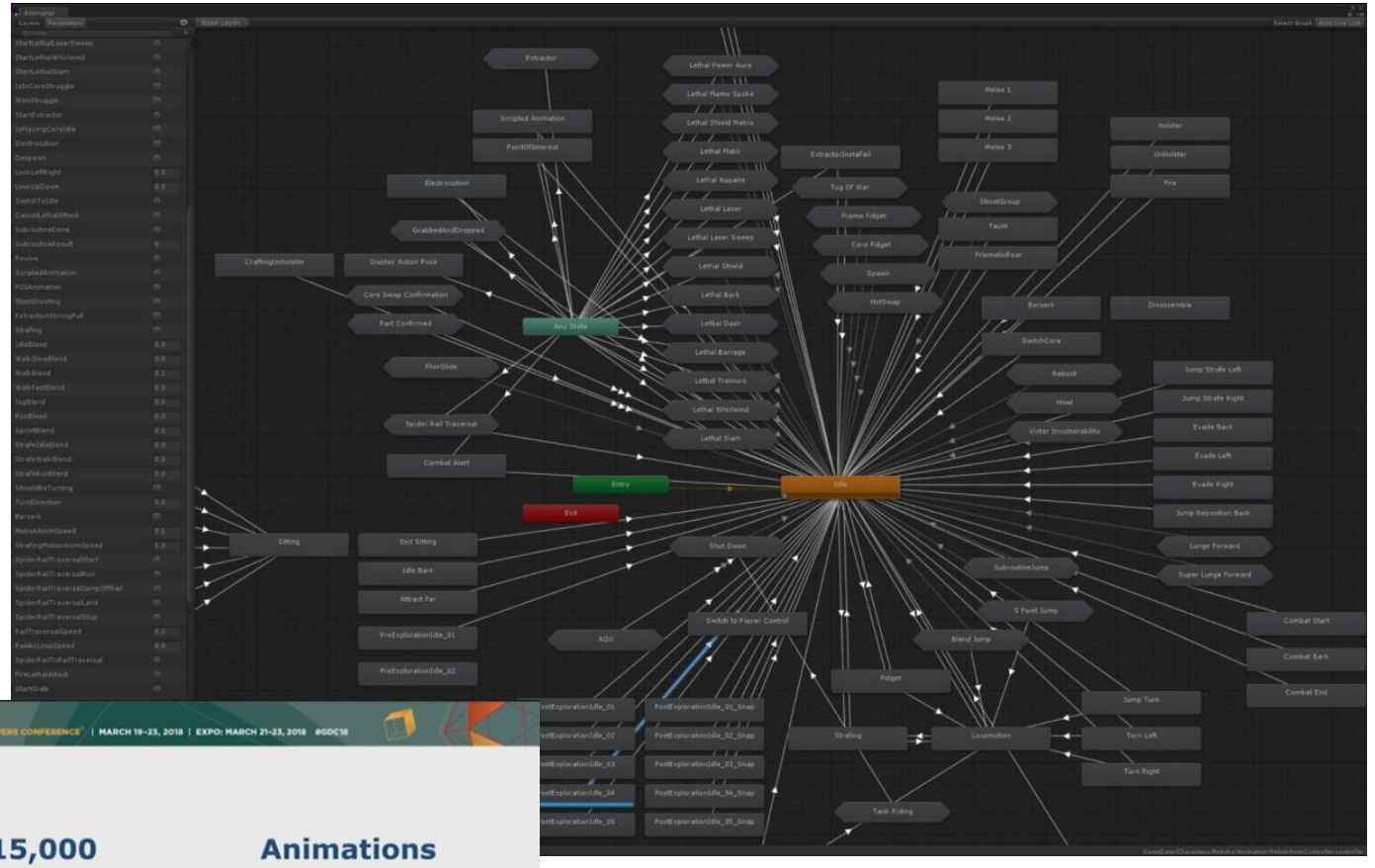
State Machine Based Animation

```
// start
if (!walking && wantToWalk)
{
    PlayAnim (StartAnim);
    walking = true;
}

// walk loop
if (IsPlaying(StartAnim) && IsAtEndOfAnim())
{
    PlayAnim (WalkLoopAnim);
}

// stop
if (walking && !wantToWalk)
{
    PlayAnim (StopAnim);
    walking = false;
}
```



State Machine Based Animation



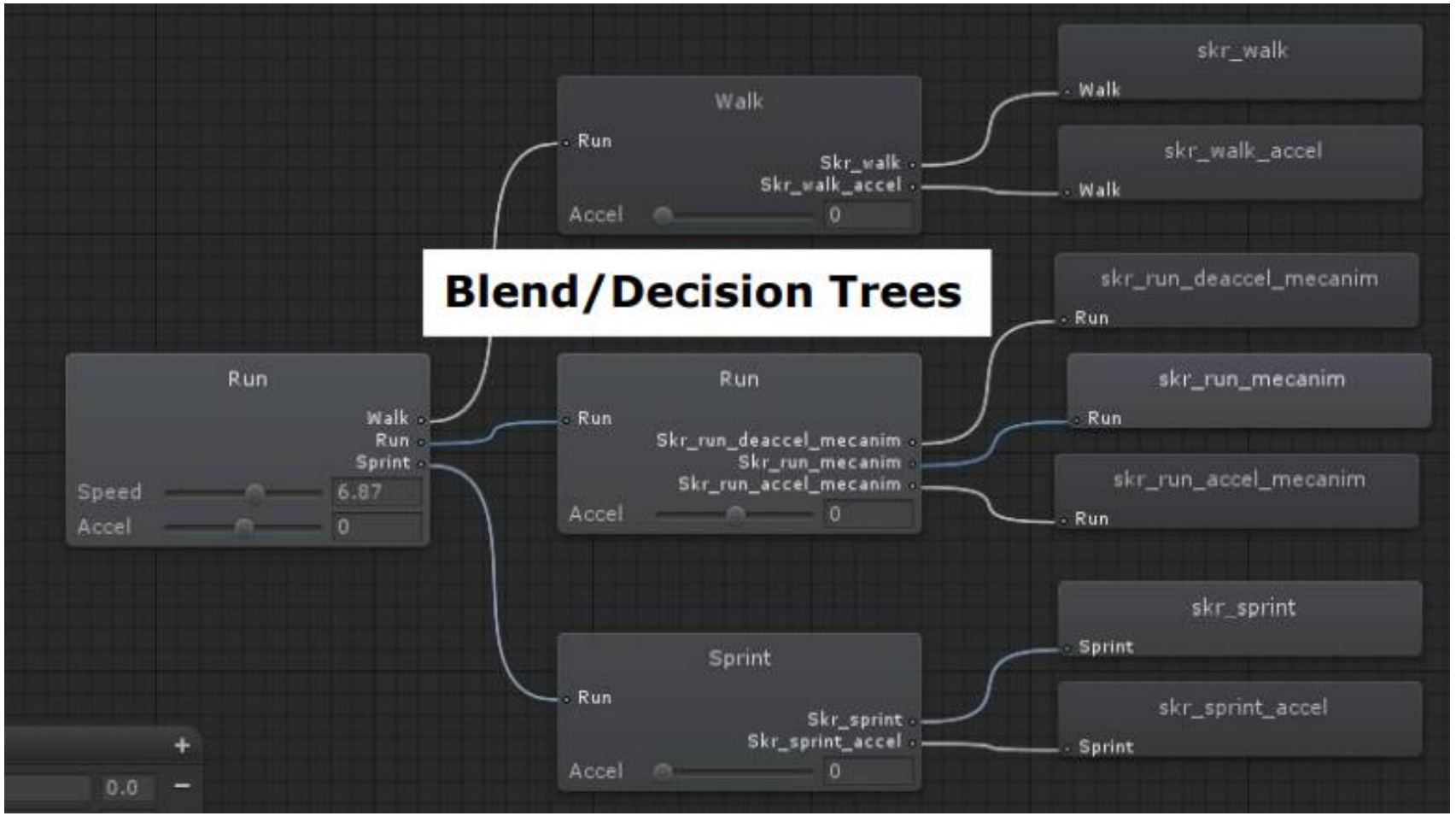


State Machine Based Animation

- 如何进行动作衔接?

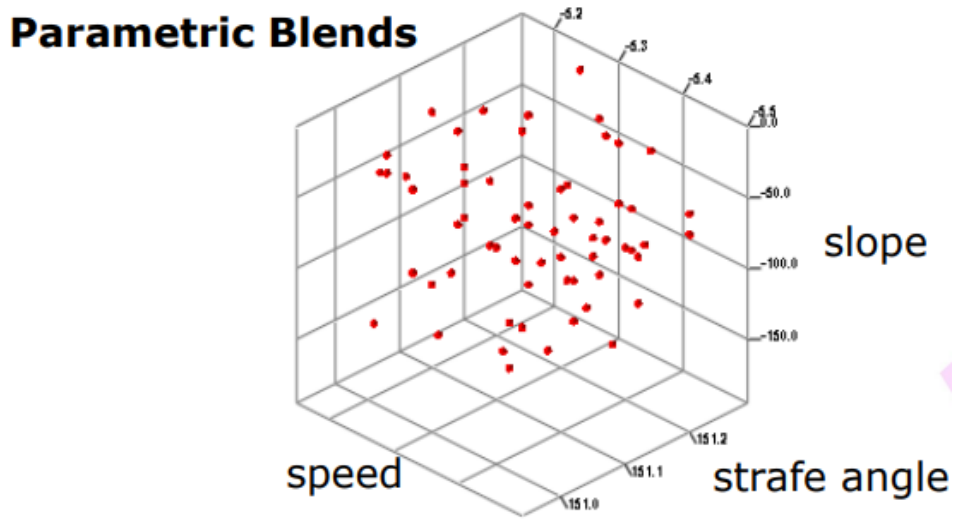
```
if (speed > 3.0f)
{
    PlayAnim (RunAnim);
}

else if (speed > 0.0f)
{
    PlayAnim (WalkAnim);
}

else
{
    PlayAnim (IdleAnim);
}
```

State Machine Based Animation



State Machine Based Animation: Problem

- 当动作量增大时，状态机复杂度平方级增长，极大加大了BUG出现的频率与调试的事件
- 给资源管理带来困难
- Blend tree的参数与插值方式需要非常精细地调整



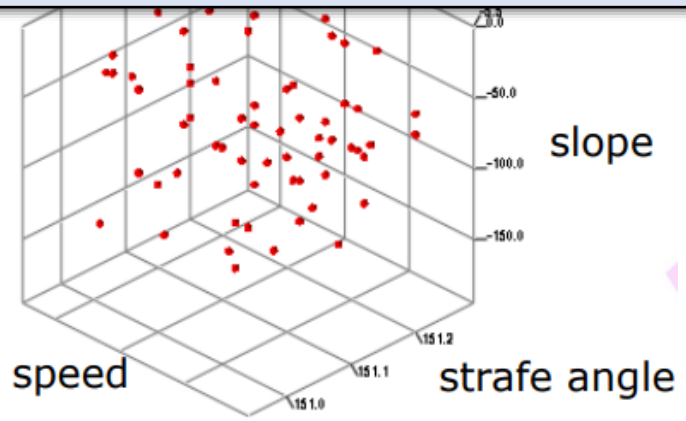
Name	Speed	SightVelocityAngle	SlopeVelocityAngle	Banking	AwarenessContext	Stances	Bump
Hero_Nav_RunInField_5Msec	5	0	0	0	Tall Grass	Run	0
Hero_Nav_WalkFwd1Msec_BankRt	1	0	0	1	Default	Walk	0
Hero_Nav_WalkFwd1Msec_BankLt	1	0	0	-1	Default	Walk	0
Hero_Nav_MoCapNarrativeWalkSlow_1Msec	1	0	0	0	Default	Walk	0
Hero_Nav_JogSlow_3Msec							
Hero_Nav_JogSlow_4Msec							
Hero_Jst_WalkFast_3Msec							
Hero_Fight_WalkFwd_BankR							
Hero_Fight_WalkFwd_BankL							
Hero_Nav_WalkSlope_30Deg							
Hero_Nav_TransitionLoop_F							
Hero_Nav_RunBump_LtSide							
Hero_Nav_RunBump_RtSide							
Hero_Nav_RunFwdDirection							
Hero_Nav_RunFwdDirection							
Hero_Agg_SprintFwd_BankR							
Hero_Jst_walkAggresive_2M							
Hero_Nav_Run_5m_NewSty							
Hero_Agg_SprintFwd_BankL							
Hero_Agg_SprintBump8m_L							
Hero_Agg_SprintBump8m_R							
Hero_Agg_SprintBump6m_L							
Hero_Agg_SprintBump6m_R							
NPC_Walk_Slow							
Hero_Nav_Run_5m_NewSty							
Hero_Nav_Sprint_stop_6m							
Hero_Nav_Sprint_7m_NewS							
Hero_Jst_walkAggresive_2M							
Hero_Nav_Run_Slope_30DegUp_5m	5	0	-30	0	Default	Sprint	0
Hero_Nav_Run_Slope_30DegDown_5m	5	0	30	0	Default	Sprint	0
Hero_Nav_Sprint_8m	8	0	0	0	Default	Run	0
Hero_sprint_10m	10	0	0	0	Default	Run	0
Hero_Nav_Run_5m_NewStyle_LongStride_20Fps	5	0	0	0	Default	Sprint	0

```
// a parametric-blend query
AnimQuery query;

// let's strafe up a slope
query.AddDesiredFeature("speed", 3.0f);
query.AddDesiredFeature("slope", 35.0f);
query.AddDesiredFeature("strafeAngle", 90.0f);

Array<float> blendFactors = ComputeBlendFactors(query);

SetBlendFactors(blendFactors);
```

State Machine Based Animation: Problem

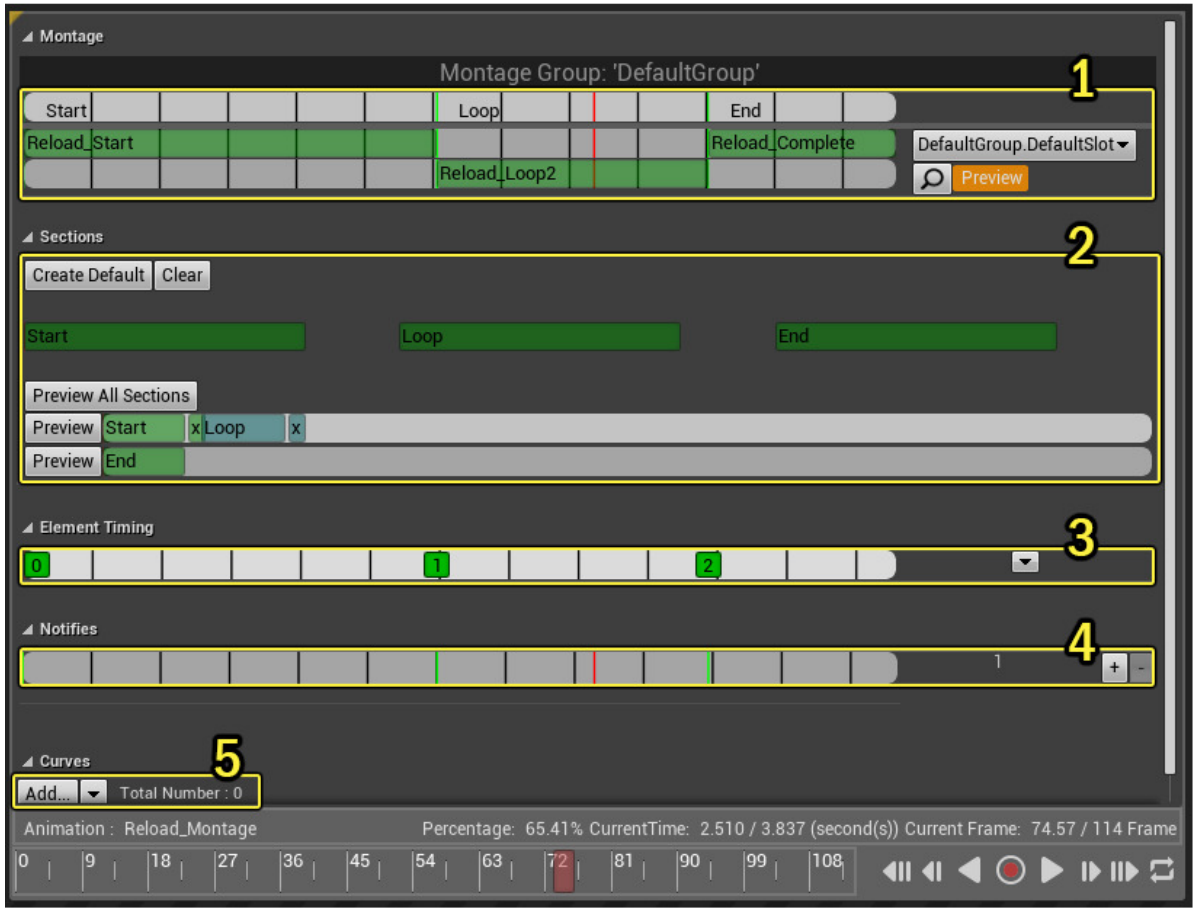
- 当动作量增大时，状态机复杂度平方级增长，极大加大了BUG出现的频率与调试的事件

基于状态机的动画复杂度较大，需要大量人工调试



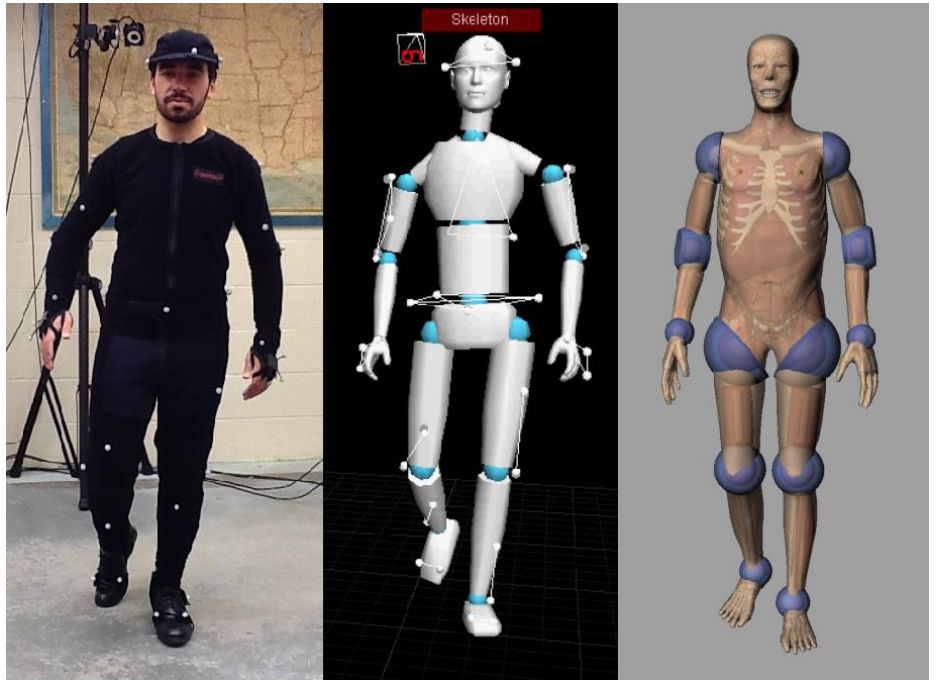

```
// a parametric-blend query
AnimQuery query;

// let's strafe up a slope
query.AddDesiredFeature("speed", 3.0f);
query.AddDesiredFeature("slope", 35.0f);
query.AddDesiredFeature("strafeAngle", 90.0f);

Array<float> blendFactors = ComputeBlendFactors(query);

SetBlendFactors(blendFactors);
```

Montage in UE



1. 蒙太奇区域，包含了Section、Slot
   - 片段Section：表示一个动画片段
   - 插槽Slot：一条轨道，可以创建多个Slot实现不同的动画版本
2. Section区域
3. Element Timing区域，从蒙太奇和Notify区域提取信息，以帮助设定不同片段的时间
4. Notify区域：将事件设置在动画的特定事件发生
5. Curve区域

Motion Capture



Motion Capture: Pros & Cons

- 自由度高
- 质量好
- 在某些情况下复杂度更低

- 较为昂贵
- 仍然需要后处理
- 只能应用到类人体模型中

Motion Matching —— 概念与发展

Table of Contents

- 基于状态机的动画
- Motion Matching的引出与基本概念
- Motion Matching的要点与对比
- Motion Matching的发展
- 小结

Motion Matching: Motivation

- 能不能找到一种统一的方式处理状态机中的转移、循环、融合等复杂操作?
- 如果能，那要怎么找到下一个要播放什么**动画**?

Motion Matching: Motivation

- 能不能找到一种统一的方式处理状态机中的转移、循环、融合等复杂操作?
- 如果能，那要怎么找到下一个要播放什么**动画**?

Motion Matching Deals With It!

Motion Matching: Motivation

当动画、状态很多的时候，想要避免无序混乱的状态图

⇒ 状态机的本质是“接受**输入**，根据**预定规则**从**当前动画**转移到**下一个动画**”

⇒ 期望能够根据**当前的动画**和**玩家输入**自动判断下一个动画是什么

⇒ 根据当前动画信息与玩家输入信息自动抽取最合适的**下一帧**动画

⇒ 输入：当前帧的动画信息&玩家输入

输出：下一帧动画

Motion Matching: Motivation

当动画、状态很多的时候，想要避免无序混乱的状态图

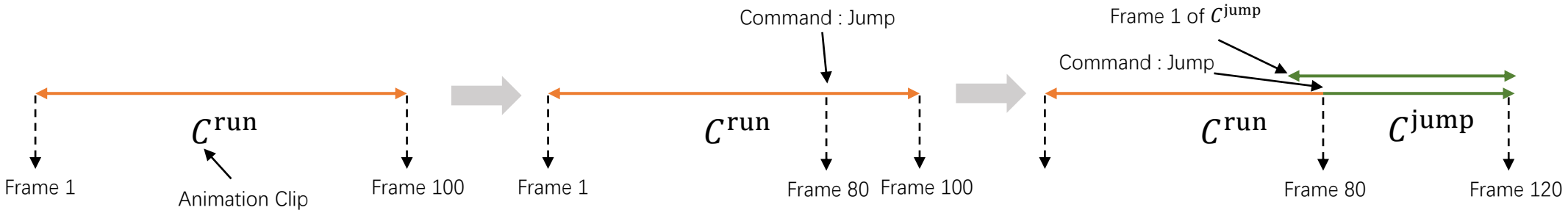
⇒ 状态机的本质是“接受**输入**，根据**预定规则**从**当前动画**转移到**下一个动画**”

⇒ 期望能够根据**当前的动画**和**玩家输入**自动判断下一个动画是什么

⇒ 根据当前动画信息与玩家输入信息自动抽取最合适的下一帧动画

⇒ 输入：当前帧的动画信息&玩家输入

输出：下一帧动画



Motion Matching: Motivation

当动画、状态很多的时候，想要避免无序混乱的状态图

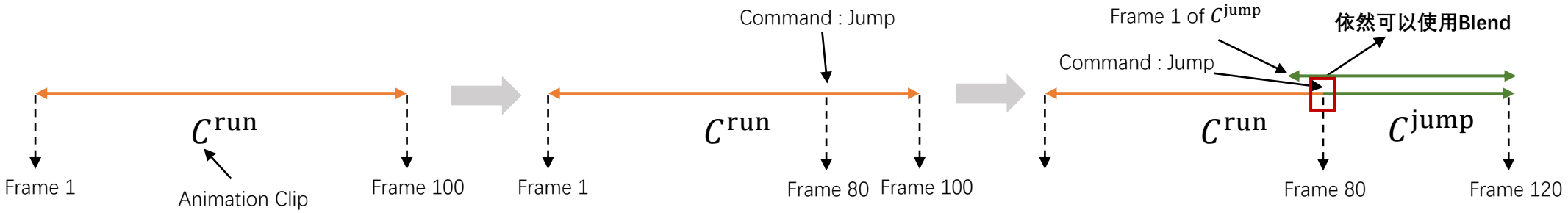
⇒ 状态机的本质是“接受**输入**，根据**预定规则**从**当前动画**转移到**下一个动画**”

⇒ 期望能够根据**当前的动画**和**玩家输入**自动判断下一个动画是什么

⇒ 根据当前动画信息与玩家输入信息自动抽取最合适的下一帧动画

⇒ 输入：当前帧的动画信息&玩家输入

输出：下一帧动画



Motion Matching: Motivation

EA出品的UFC3 (2018) 全部使用了MM

A Demo showed by UbiSoft

A Demo showed by UbiSoft

Motion Matching: Steps

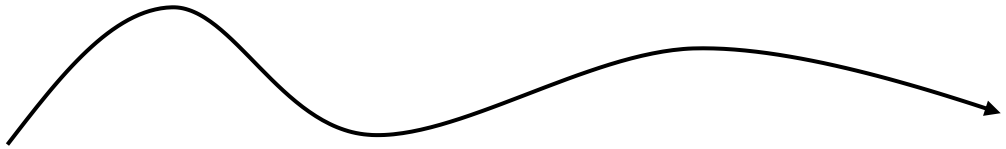
1. 获取当前状态+期望未来轨迹
   - 当前状态：当前帧的pose
   - 期望未来轨迹：位置+速度，来自玩家输入或无玩家输入
2. 从动画数据库中检索查找最匹配的一帧动画
   - 与“期望未来轨迹”的差距越小越好，但同时也不能偏离当前动画太多
   - $C^* = \text{argmin}_{C^{\text{candidate}}}$ CalculateCost $(\text{C}^{\text{current}}, \text{C}^{\text{candidate}}, \text{T}^{\text{target}})$
3. 从最优帧开始播放动画

Motion Matching: Current Pose

1. 获取当前状态+期望未来轨迹
   - **当前状态：当前帧的pose**
   - 期望未来轨迹：位置+速度，来自玩家输入或无玩家输入
2. 从动画数据库中检索查找最匹配的一帧动画
   - 与“期望未来轨迹”的差距越小越好，但同时也不能偏离当前动画太多
   - $C^* = \text{argmin}_{C^{\text{candidate}}}\ \text{CalculateCost}\ (\text{C}^{\text{current}}, \text{C}^{\text{candidate}}, \text{T}^{\text{target}})$
3. 从最优帧开始播放动画

Motion Matching: Current Pose

- Trajectory: root animation



- Pose: defined as the joint transforms **including** a trajectory section
  - A trajectory section is always associated with a pose

Motion Matching: Current Pose

- Root velocity
- Root position
- Feet velocity / Hand velocity
- Feet position / Hand position
- Weapon position
- Etc.
- 每个Pose的信息都可以预先计算并存储起来

Motion Matching: Future Trajectory

1. 获取当前状态+期望未来轨迹
   - 当前状态：当前帧的pose
   - **期望未来轨迹：位置+速度，来自玩家输入或无玩家输入**
2. 从动画数据库中检索查找最匹配的一帧动画
   - 与“期望未来轨迹”的差距越小越好，但同时也不能偏离当前动画太多
   - $C^* = \text{argmin}_{C^{\text{candidate}}}\ \text{CalculateCost}\ (\text{C}^{\text{current}}, \text{C}^{\text{candidate}}, \text{T}^{\text{target}})$
3. 从最优帧开始播放动画

Motion Matching: Future Trajectory

- 包含位置与速度信息
- 如果没有捕获到玩家输入
  - 则将当前帧之后的一定时间区间内的轨迹视为Future Trajectory
- 如果有玩家输入
  - 使用模拟得到预测未来路径
    - 键盘、鼠标
    - 手柄摇杆
  - AI未来路径

Motion Matching: Calculate Cost

1. 获取当前状态+期望未来轨迹
   - 当前状态：当前帧的pose
   - 期望未来轨迹：位置+速度，来自玩家输入或无玩家输入
2. 从动画数据库中检索查找最匹配的一帧动画
   - **与“期望未来轨迹”的差距越小越好，但同时也不能偏离当前动画太多**
   - $C^* = \text{argmin}_{C^{\text{candidate}}}\ \text{CalculateCost}\ (\text{C}^{\text{current}}, \text{C}^{\text{candidate}}, \text{T}^{\text{target}})$
3. 从最优帧开始播放动画

Motion Matching: Calculate Cost

1. 获取当前状态+期望未来轨迹
   - 当前状态：当前帧的pose
   - 期望未来轨迹：位置+速度，来自玩家输入或无玩家输入
2. 从动画数据库中检索查找最匹配的一帧动画
   - 与“期望未来轨迹”的差距越小越好，但同时也不能偏离当前动画太多
   - $C^* = \text{argmin}_{C^{\text{candidate}}}$ **CalculateCost** $(C^{\text{current}}, C^{\text{candidate}}, T^{\text{target}})$

3. 从最优帧开始播放动画

Motion Matching: Calculate Cost

```
float CalculateCost (currentPose, candidatePose, targetTrajectory)
{
    float cost = 0.0f;

    cost += ComputePoseCost (currentPose, candidatePose);

    float responsiveness = 1.0f;

    cost += responsiveness *
            ComputeFutureCost (candidatePose, targetTrajectory);

    return cost;
}
```

总的Cost由两部分组成

Responsiveness控制了轨迹匹配的精确度，或者响应度

```
float ComputePoseCost (currentPose, candidatePose)
{
    float cost = 0.0f;

    cost += Distance (currentPose.RF.pos - candidatePose.RF.pos);
    cost += Distance (currentPose.RF.vel - candidatePose.RF.vel);
    cost += Distance (currentPose.LF.pos - candidatePose.LF.pos);
    cost += Distance (currentPose.LF.vel - candidatePose.LF.vel);
    cost += Distance (currentPose.RH.pos - candidatePose.RH.pos);
    cost += Distance (currentPose.RH.vel - candidatePose.RH.vel);
    cost += Distance (currentPose.LH.pos - candidatePose.LH.pos);
    cost += Distance (currentPose.LH.vel - candidatePose.LH.vel);

    return cost;
}
```

计算关节位置和速度的差距

```
float ComputeFuture Cost (candidatePose, targetTrajectory)
{
    float cost = 0.0f;

    cost += Distance (candidatePose.R.pos - targetTrajectory.pos);
    cost += Distance (candidatePose.R.vel - targetTrajectory.vel);

    return cost;
}
```

计算Pose与Trajectory的差距

Motion Matching: Play

1. 获取当前状态+期望未来轨迹
   - 当前状态：当前帧的pose
   - 期望未来轨迹：位置+速度，来自玩家输入或无玩家输入
2. 从动画数据库中检索查找最匹配的一帧动画
   - 与“期望未来轨迹”的差距越小越好，但同时也不能偏离当前动画太多
   - $C^* = \text{argmin}_{C^{\text{candidate}}}\ \text{CalculateCost}\ (\text{C}^{\text{current}}, \text{C}^{\text{candidate}}, \text{T}^{\text{target}})$
3. **从最优帧开始播放动画**

Motion Matching: Play

1. 获取当前状态+期望未来轨迹
   - 当前状态：当前帧的pose
   - 期望未来轨迹：位置+速度，来自玩家输入或无玩家输入
2. 从动画数据库中检索查找最匹配的一帧动画
   - 与“期望未来轨迹”的差距越小越好，但同时也不能偏离当前动画太多
   - $C^* = \text{argmin}_{C^{\text{candidate}}}\ \text{CalculateCost}\ (\text{C}^{\text{current}}, \text{C}^{\text{candidate}}, \text{T}^{\text{target}})$
3. **从最优帧开始播放动画**

至少播3帧

Motion Matching —— 概念与发展

Table of Contents

- 基于状态机的动画
- Motion Matching的引出与基本概念
- Motion Matching的要点与对比
- Motion Matching的发展
- 小结

Motion Matching: Pros & Cons

Pros:

- 自动化处理，提升开发效率（尤其是增量开发）
- 产生高质量结果
- 实现相对简单
- 可嵌入到State Machine中

Cons:

- 依赖高质量数据
- 需要设计Cost函数
- 当前难以处理复杂行为（如攀爬、射击和复合动作）

State Machine vs. Motion Matching

	State Machine	Motion Matching
动画来源	制作/动捕	动捕
动画转移	手工设计	自动匹配
数据要求	任意动画数量	对规模和质量有一定要求
复杂度	随动画数量平方级增长	随动画数量线性增长
适用性	复杂动作下需要大量调优	理论上能适用任何动作
伸缩性	复杂情况下难以变动	可通过数据和算法随意扩展
多样性	可用于任意模型	当前只能用于人类模型

Motion Matching —— 概念与发展

Table of Contents

- 基于状态机的动画
- Motion Matching的引出与基本概念
- Motion Matching的要点与对比
- Motion Matching的发展
- 小结

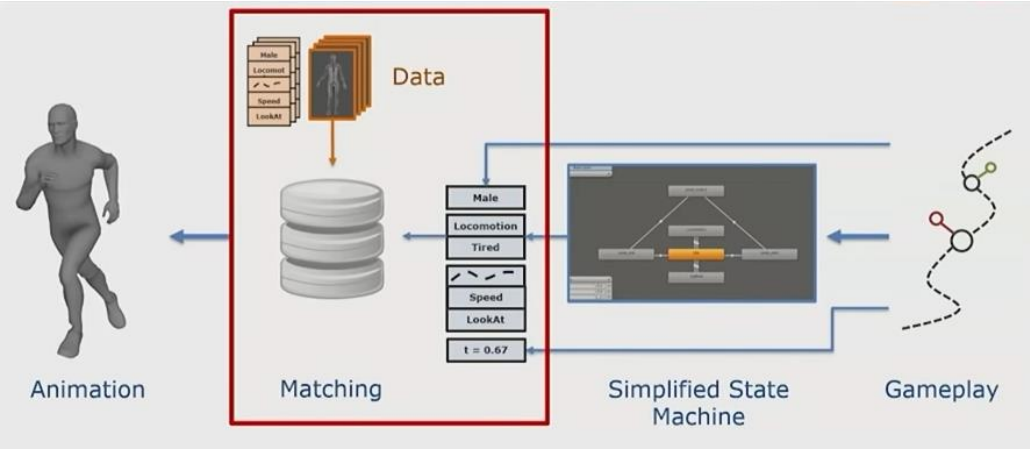
Optimizations of Motion Matching

问题：Naïve Motion Matching计算量过大
目标：减小计算量

- 增大更新间隔（Update Interval）
- 增大容忍度（tolerance）
- 标记不参与搜索的Pose
- 对pose进行分类，在搜索时只在某些类别中搜索
- 提高搜索效率：KD-Tree / PCA / Candidate Set / KNN search / Neural Network

Integrating MM into State Machine

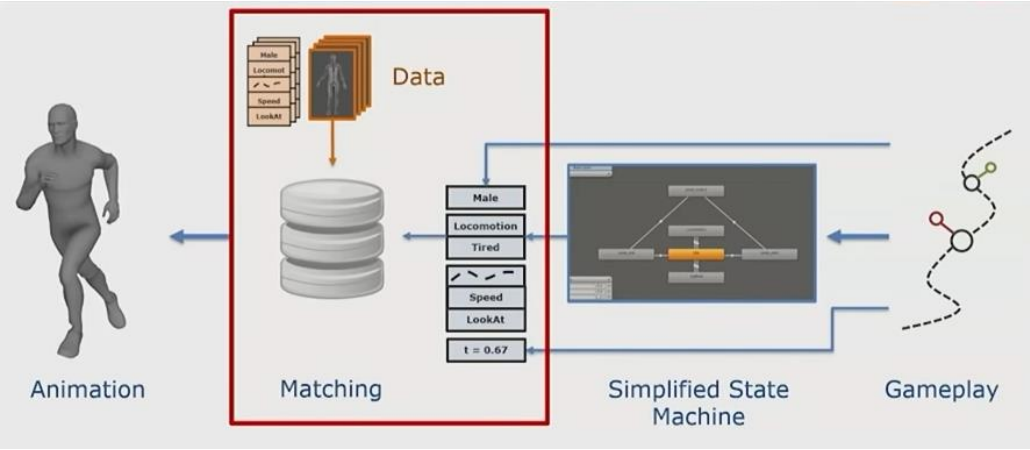
- 将状态机分为Gameplay与Animation两个部分
  - States -> Variables
  - Querying -> Matching



GDC 2018, Character Control with Neural Networks and Machine Learning, https://www.youtube.com/watch?v=o-QLSjSSyVk

Integrating MM into State Machine

- 将状态机分为Gameplay与Animation两个部分
  - States -> Variables
  - Querying -> Matching
- 可以使用KNN找到最优Pose



Input x

- **Joint Positions** in the previous frame.
- **Joint Velocities** in the previous frame.
- **Target Position** of the root in 1 second.
- **Target Velocity** of the root in 1 second.
- **Target Direction** of the root in 1 second.

Function f

- **Call** every 1 second or…
- **Call** if the user input changes.

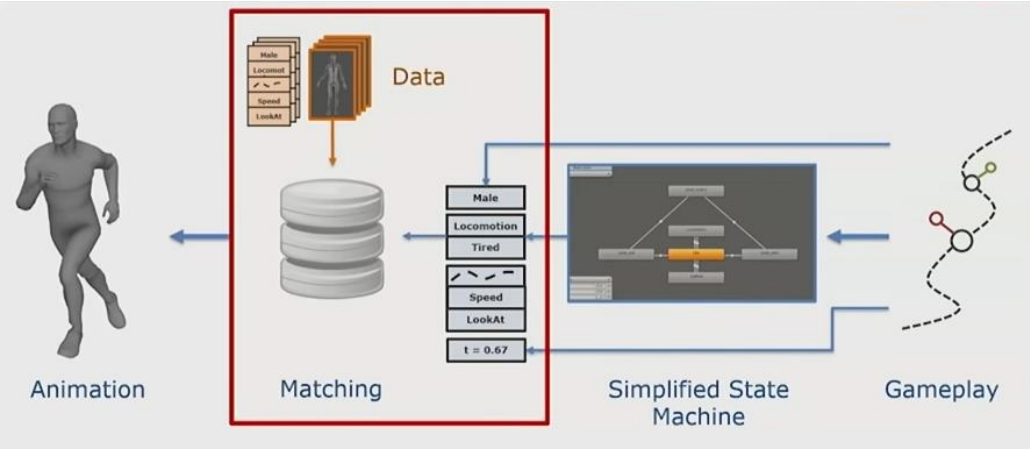
Output y

- **Joint Positions** for the next 1 second.
- **Joint Rotations** for the next 1 second.

GDC 2018, Character Control with Neural Networks and Machine Learning, https://www.youtube.com/watch?v=o-QLSjSSyVk

Integrating MM into State Machine

- 将状态机分为Gameplay与Animation两个部分
  - States -> Variables
  - Querying -> Matching
- 可以使用KNN找到最优Pose
- 或者可以使用Neural Network!



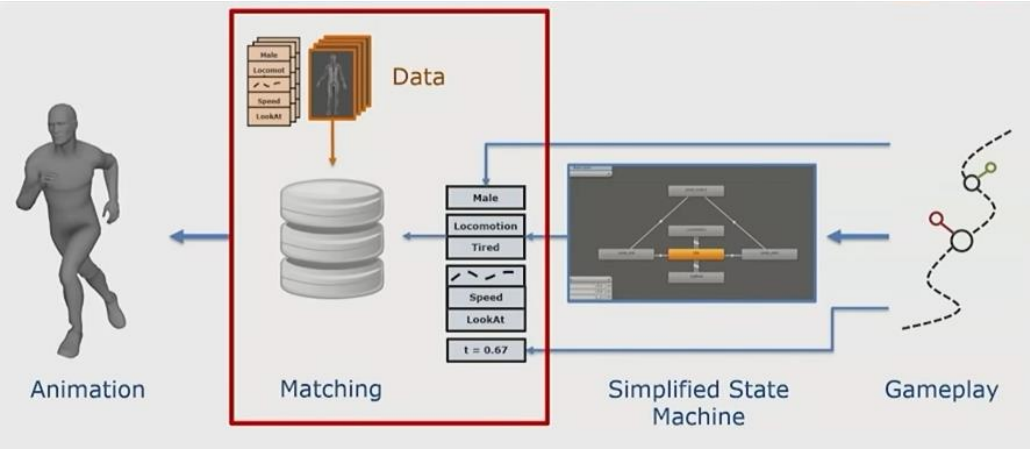
$$\underbrace{y = W_2\sigma(W_1\sigma(W_0x + b_0) + b_1) + b_2}_{y = f(x)}$$

$$y = \begin{bmatrix} 5.12 \\ 2.12 \\ \vdots \\ 0.31 \\ 2.54 \end{bmatrix} \qquad x = \begin{bmatrix} 1.33 \\ 7.21 \\ \vdots \\ 9.03 \\ 1.28 \end{bmatrix}$$

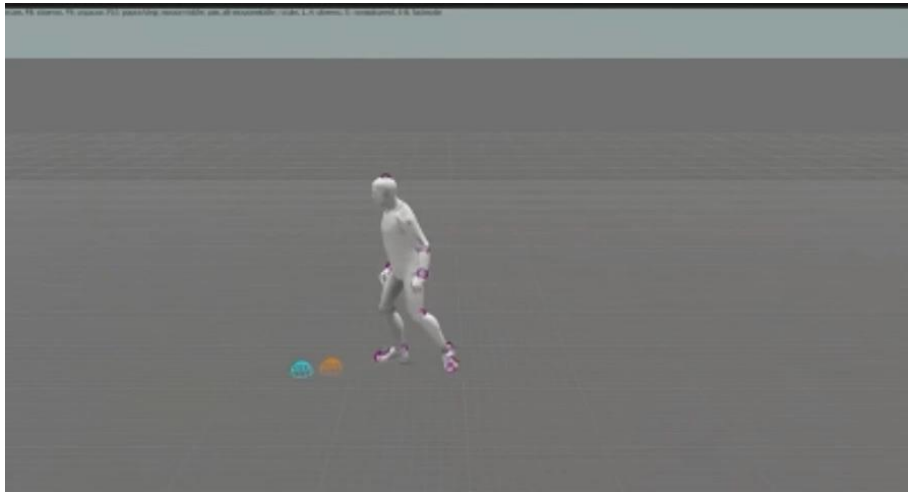
GDC 2018, Character Control with Neural Networks and Machine Learning, https://www.youtube.com/watch?v=o-QLSjSSyVk

Integrating MM into State Machine

- 将状态机分为Gameplay与Animation两个部分
  - States -> Variables
  - Querying -> Matching
- 可以使用KNN找到最优Pose
- 或者可以使用Neural Network!



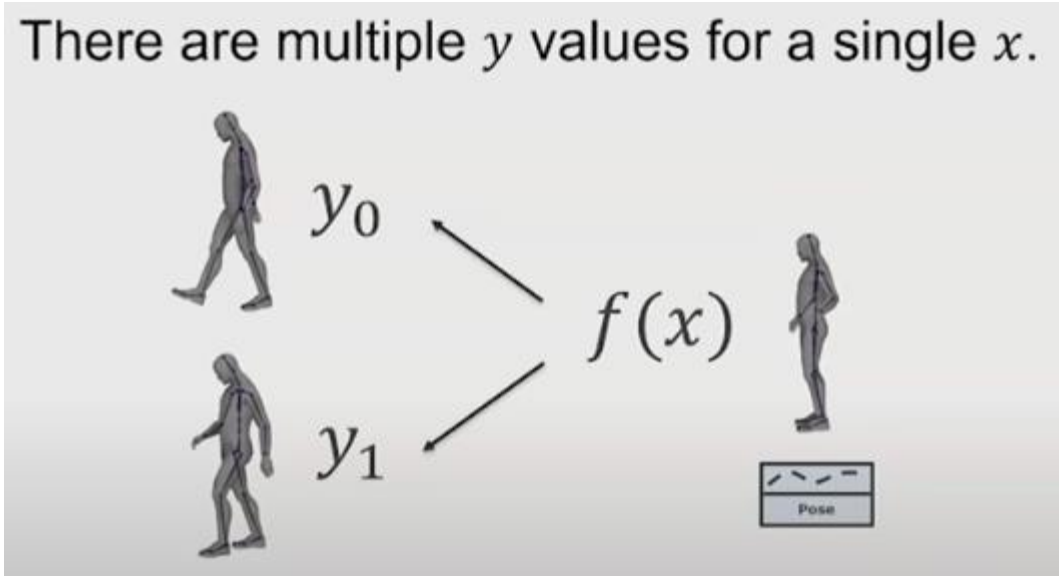
但是效果……



GDC 2018, Character Control with Neural Networks and Machine Learning, https://www.youtube.com/watch?v=o-QLSjSSyVk

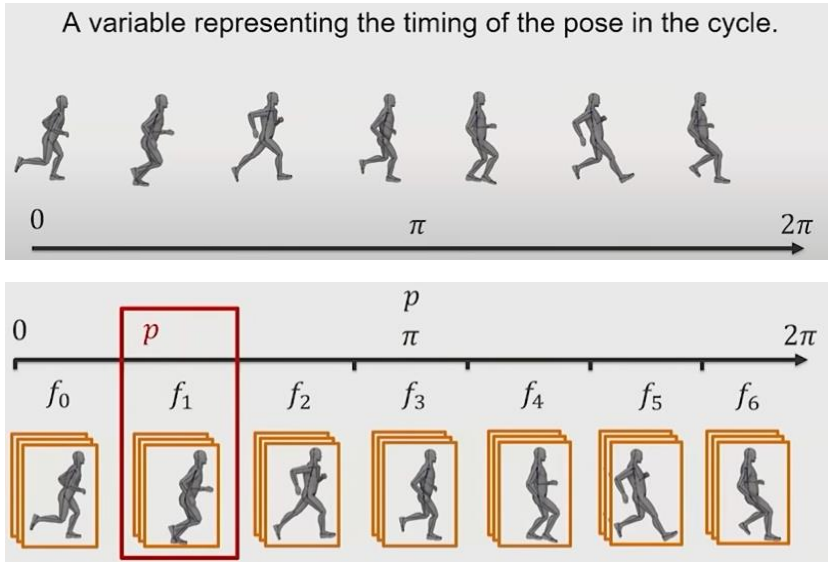
Why NN does not work

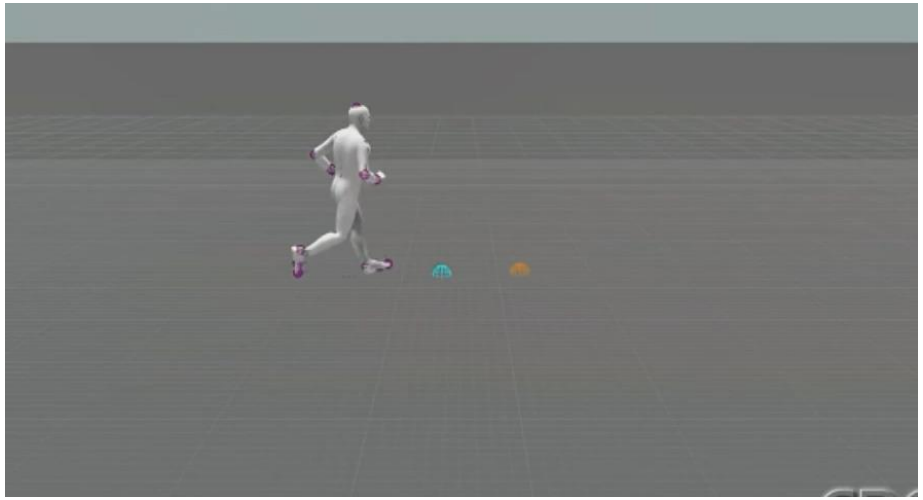
- 一个输入（X）可能对应多个可行的输出（Y）
- 模型将多个可行的输出进行了平均
- 如何解决?



PFNN：Introducing Animation Phases

- 引入Phase：代表了当前Pose处于所在Animation Clip的哪个阶段
- 不同的Phase有不同的模型去处理
- Phase $p \in [0, 2\pi)$

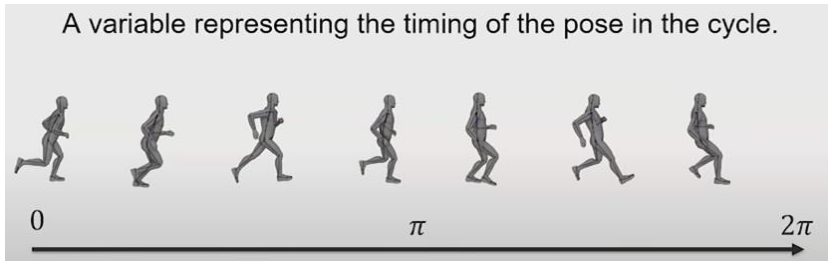


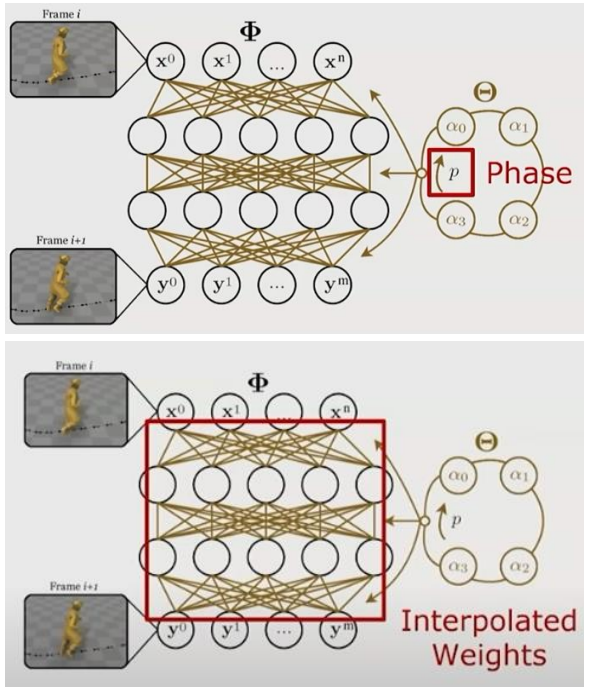


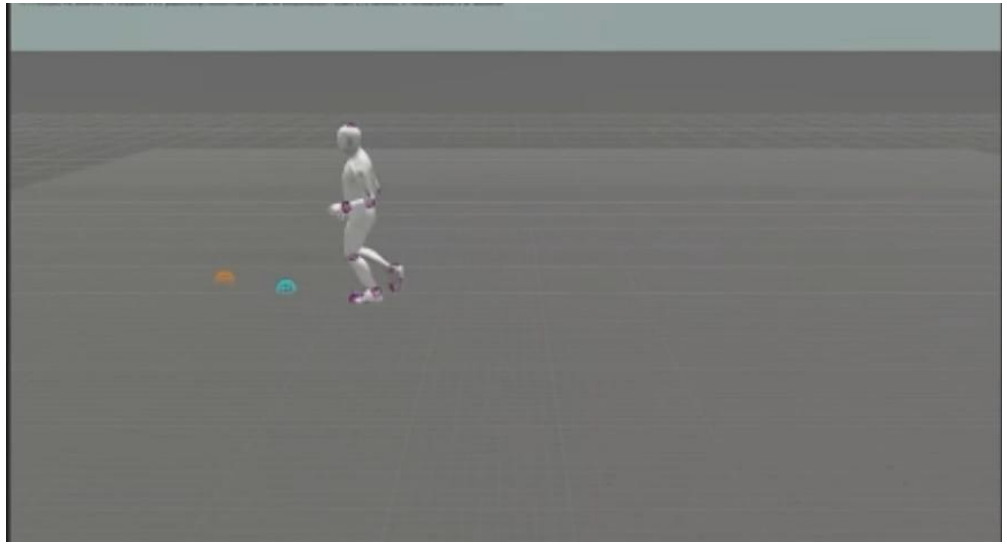
Holden, Daniel et al. “Phase-functioned neural networks for character control.” ACM Transactions on Graphics (TOG) 36 (2017): 1 - 13.

PFNN：Introducing Animation Phases

- Blend不同phase model的参数



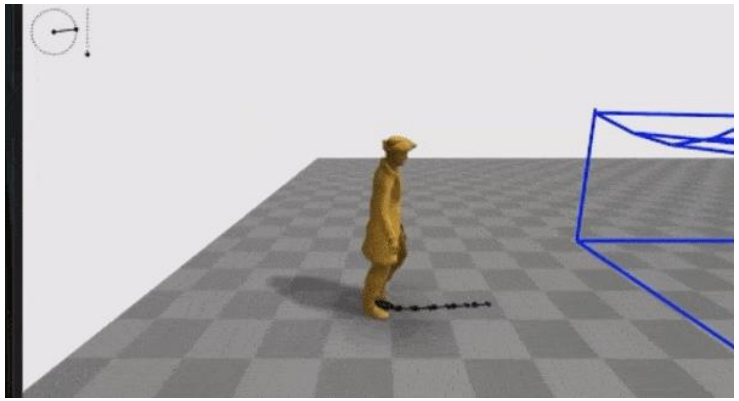
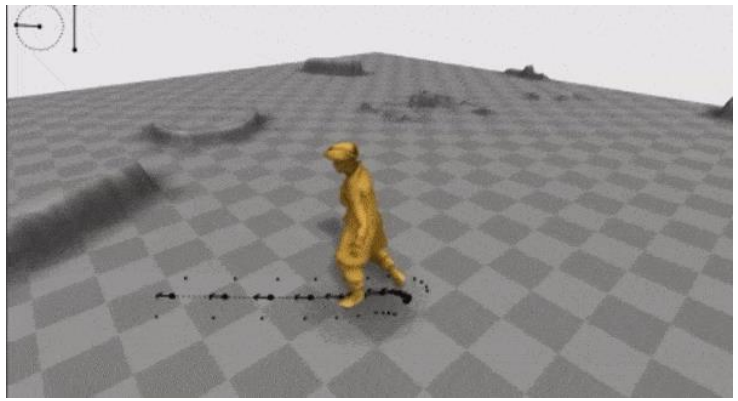




Holden, Daniel et al. “Phase-functioned neural networks for character control.” ACM Transactions on Graphics (TOG) 36 (2017): 1 - 13.

PFNN：Introducing Animation Phases

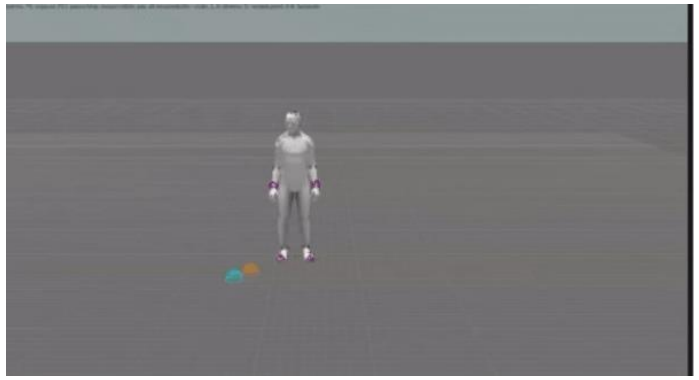
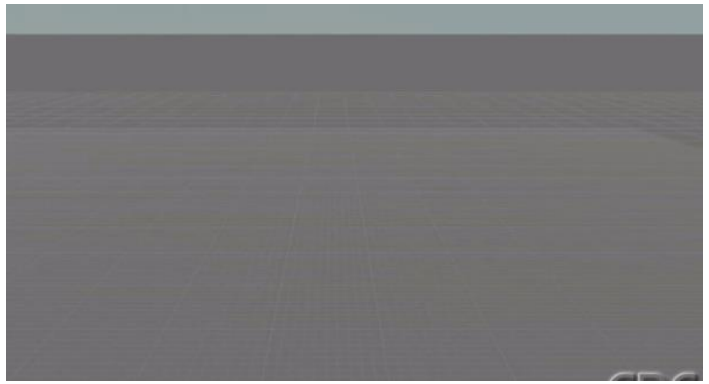
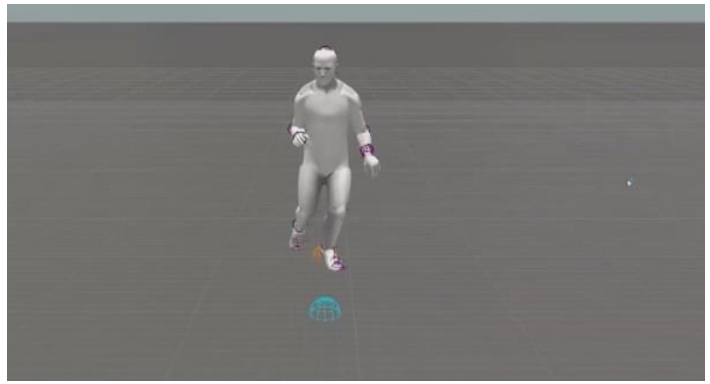
- 给不同的动作加上Tag，提高scalability

Holden, Daniel et al. “Phase-functioned neural networks for character control.” ACM Transactions on Graphics (TOG) 36 (2017): 1 - 13.

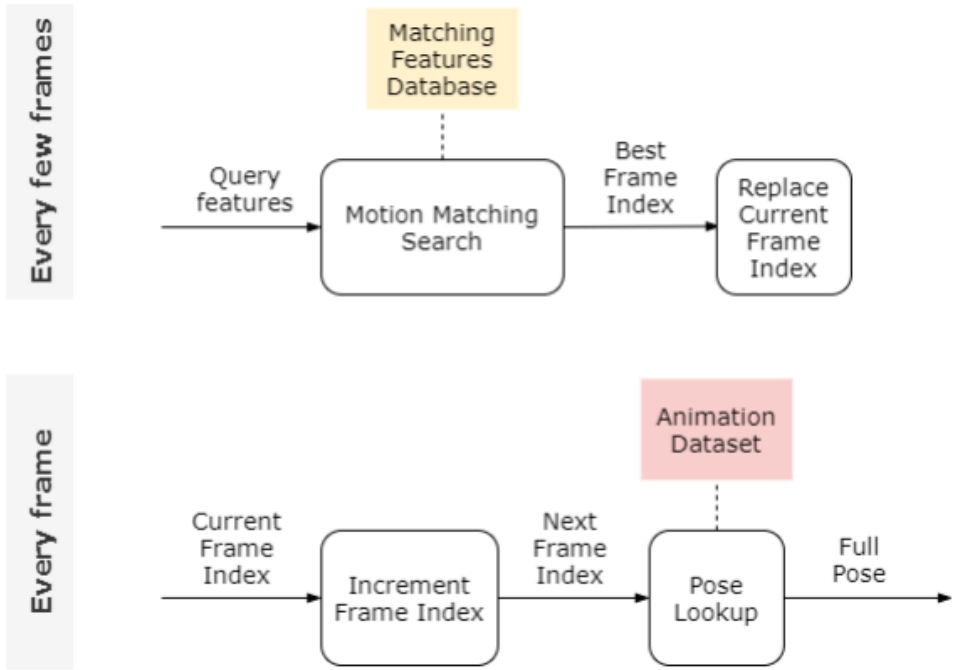
PFNN：Introducing Animation Phases

- 但是如果调不好模型……

Learned Motion Matching

- 既然要追求机器学习，那就贯彻到底咯……
  - 目标：减少资源消耗



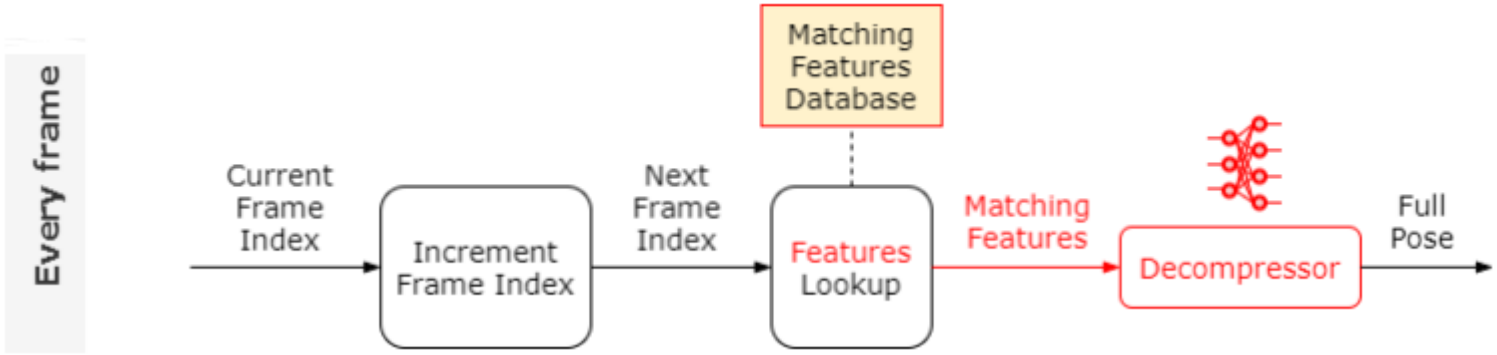
- Matching Features Database：由feature vector组成，每个feature vector都是动画库中每个pose的位置、速度数据
- Animation Dataset：原始动画数据库
- Matching Features Database与Animation Dataset都必须在运行时加载进内存
- Every few frames：每过几帧进行一次MM
- Every frame：每一帧都要从动画数据库中抽取下一帧动画进行播放

Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

Learned Motion Matching

- 第一步：从内存中移除Animation Dataset
  - Motivation：Feature vector已经包含了原始动画的许多重要信息，可以通过feature vector还原

训练一个Decompressor网络，输入是feature vector，输出是pose信息（每个joint的pos/vel）



Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

Learned Motion Matching

- 第一步：从内存中移除Animation Dataset
  - Motivation：Feature vector已经包含了原始动画的许多重要信息，可以通过feature vector还原

训练一个Decompressor网络，输入是feature vector，输出是pose信息（每个joint的pos/vel）

- 但是这是从低维映射到高维，难免会有信息损失
- 解决方法：输入时加入额外信息（类似phase的思路）

Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

Learned Motion Matching

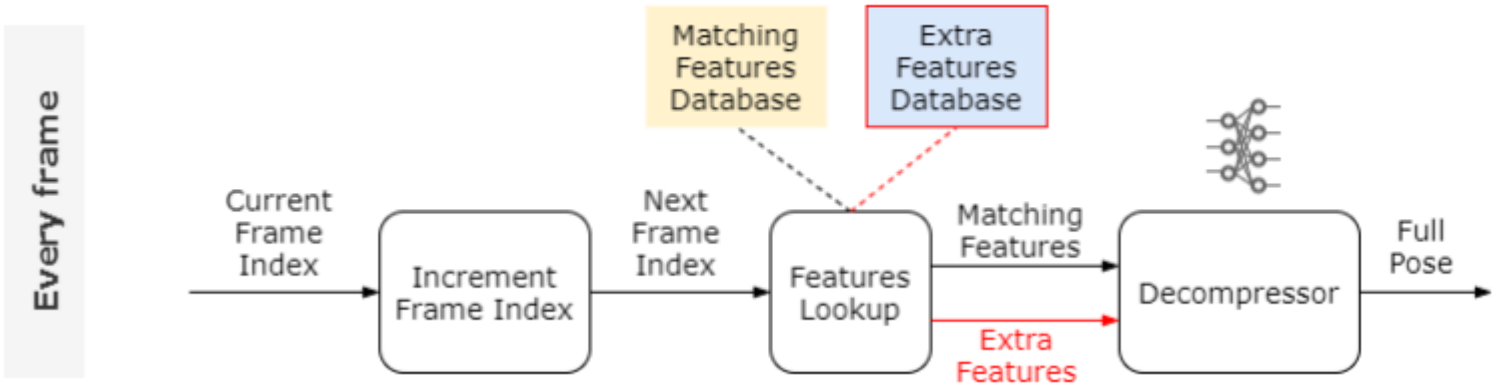
- 第一步：从内存中移除Animation Dataset
  - Motivation：Feature vector已经包含了原始动画的许多重要信息，可以通过feature vector还原

训练一个Decompressor网络，输入是feature vector，输出是pose信息（每个joint的pos/vel）

- 但是这是从低维映射到高维，难免会有信息损失
- 解决方法：输入时加入额外信息（类似phase的思路）

训练一个自编码器，**自动**提取每个feature vector的额外信息，然后把该extra feature一起输入到Decompressor中

形成一个Extra Features Database



Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

Learned Motion Matching

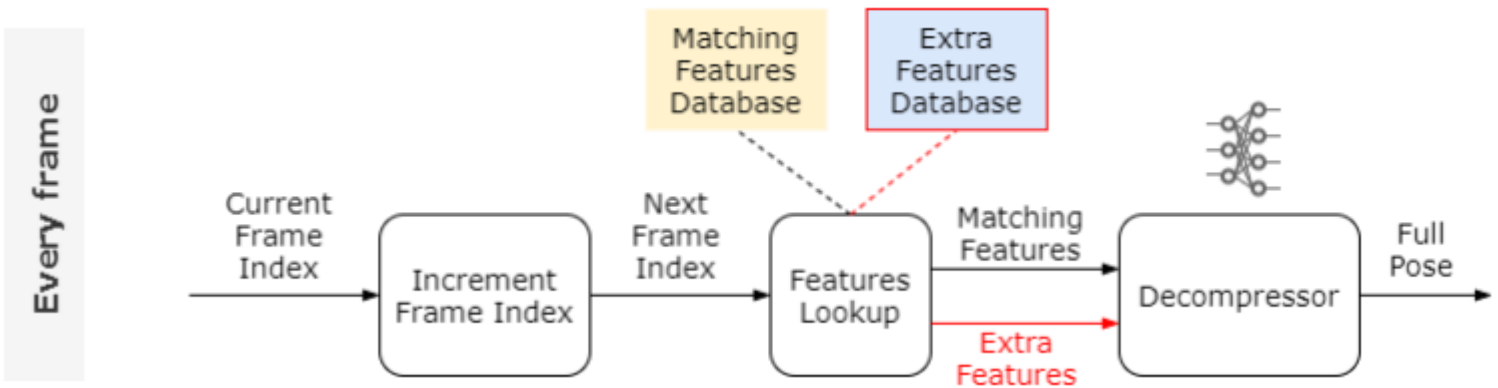
- 第一步：从内存中移除Animation Dataset
  - Motivation：Feature vector已经包含了原始动画的许多重要信息，可以通过feature vector还原

训练一个Decompressor网络，输入是feature vector，输出是pose信息（每个joint的pos/vel）

- 但是这是从低维映射到高维，难免会有信息损失
- 解决方法：输入时加入额外信息（类似phase的思路）

训练一个自编码器，**自动**提取每个feature vector的额外信息，然后把该extra feature一起输入到Decompressor中

形成一个Extra Features Database



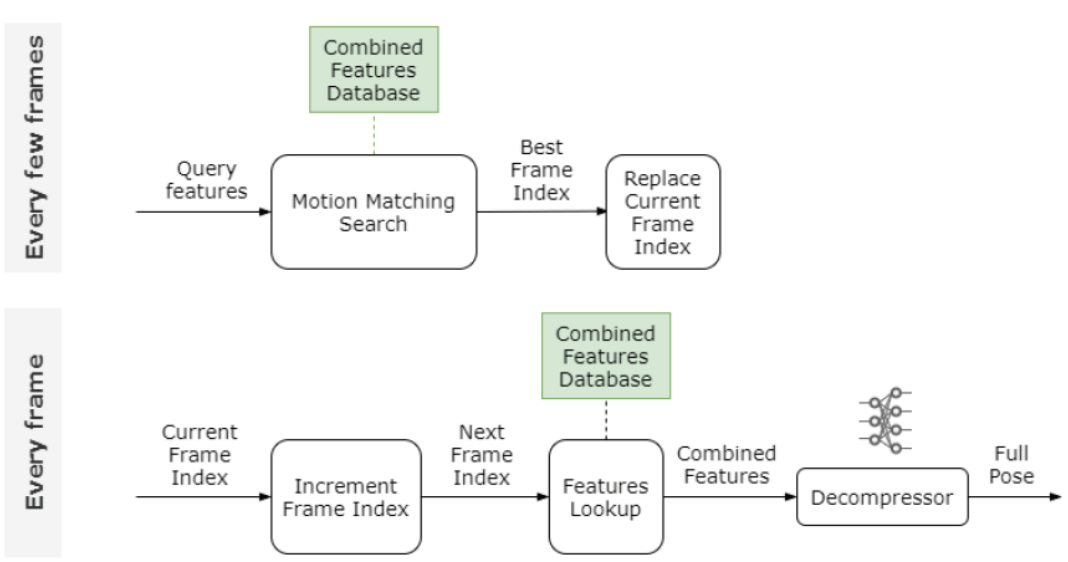
	Pose Lookup	Decompressor
Animation Dataset	42.7 MB	-
Matching Features Database	9.4 MB	9.4 MB
Extra Features Database	-	11.1 MB
Decompressor Weights	-	0.9 MB
Total	**52.1 MB**	**21.4 MB**

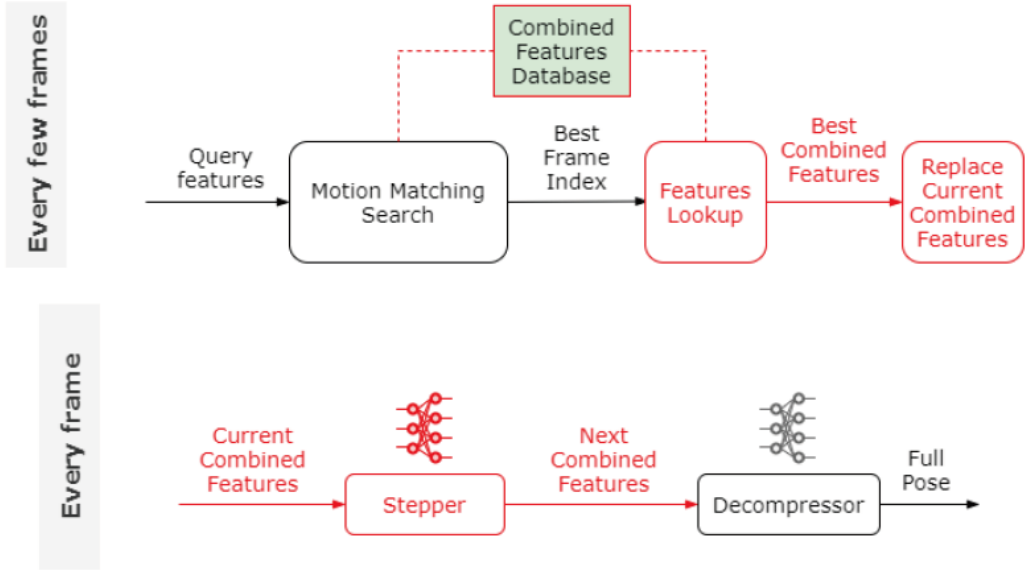
Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

Learned Motion Matching

- 第二步：从内存中移除Combined Features Database (合并两个database)

训练一个Stepper网络，在**每帧**执行的时候输入上一帧的Combined feature，输出下一帧的feature





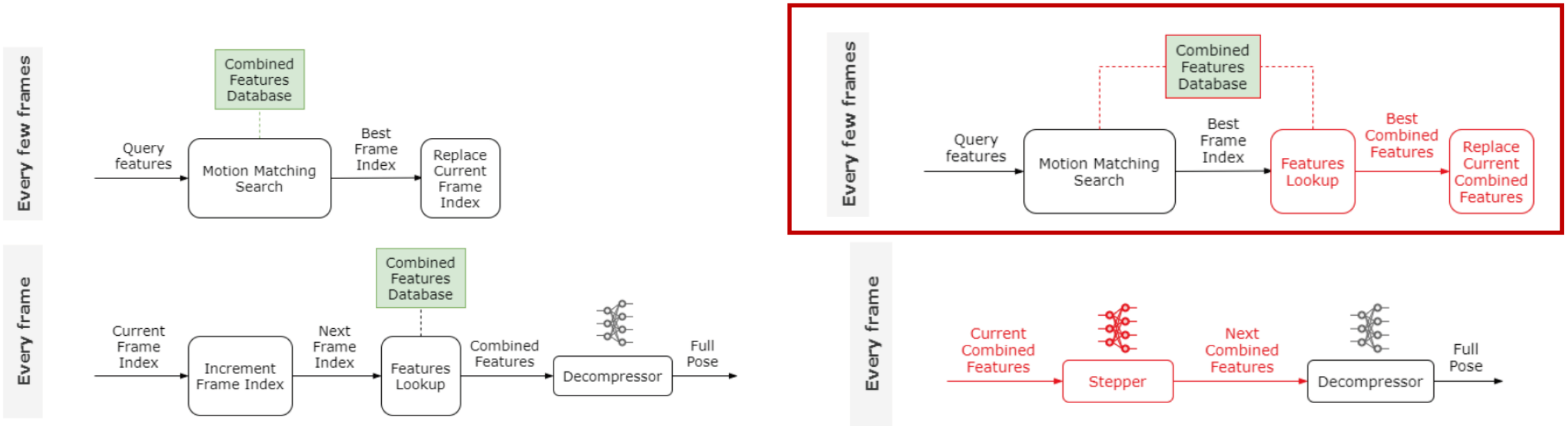
Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

Learned Motion Matching

- 第二步：从内存中移除Combined Features Database (合并两个database)

训练一个Stepper网络，在**每帧**执行的时候输入上一帧的Combined feature，输出下一帧的feature

但是在Every few frames阶段仍然需要访问Combined Features Database



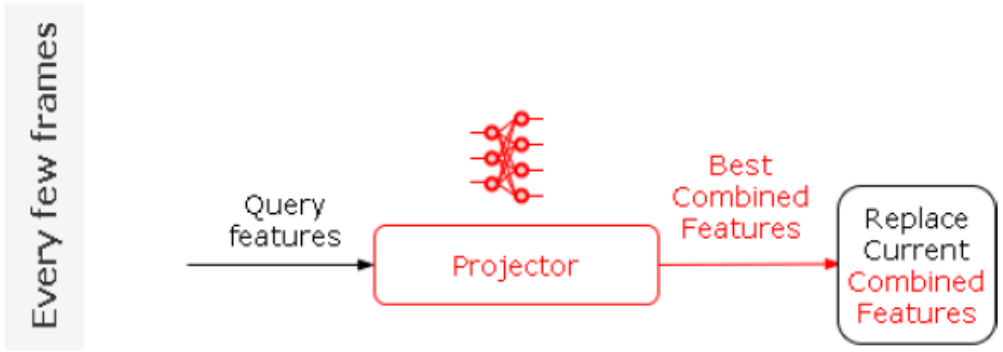
Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

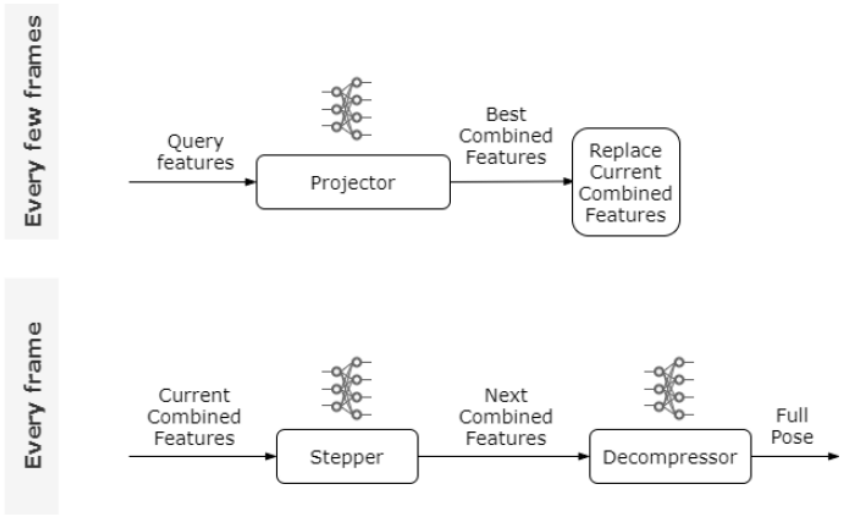
Learned Motion Matching

- 第二步：从内存中移除Combined Features Database (合并两个database)

训练一个Stepper网络，在**每帧**执行的时候输入上一帧的Combined feature，输出下一帧的feature
但是在Every few frames阶段仍然需要访问Combined Features Database

训练一个Projector网络，输入是query vector，输出是combined feature





Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

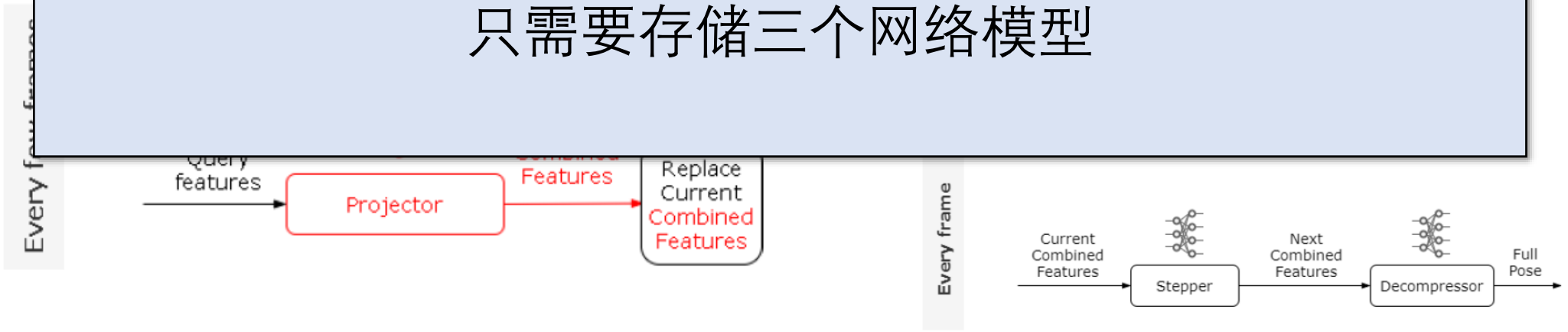
Learned Motion Matching

- 第二步：从内存中移除Combined Features Database (合并两个database)

训练一个Stepper网络，在**每帧**执行的时候输入上一帧的Combined feature，输出下一帧的feature

但是在Every few frames阶段仍然需要访问Combined Features Database

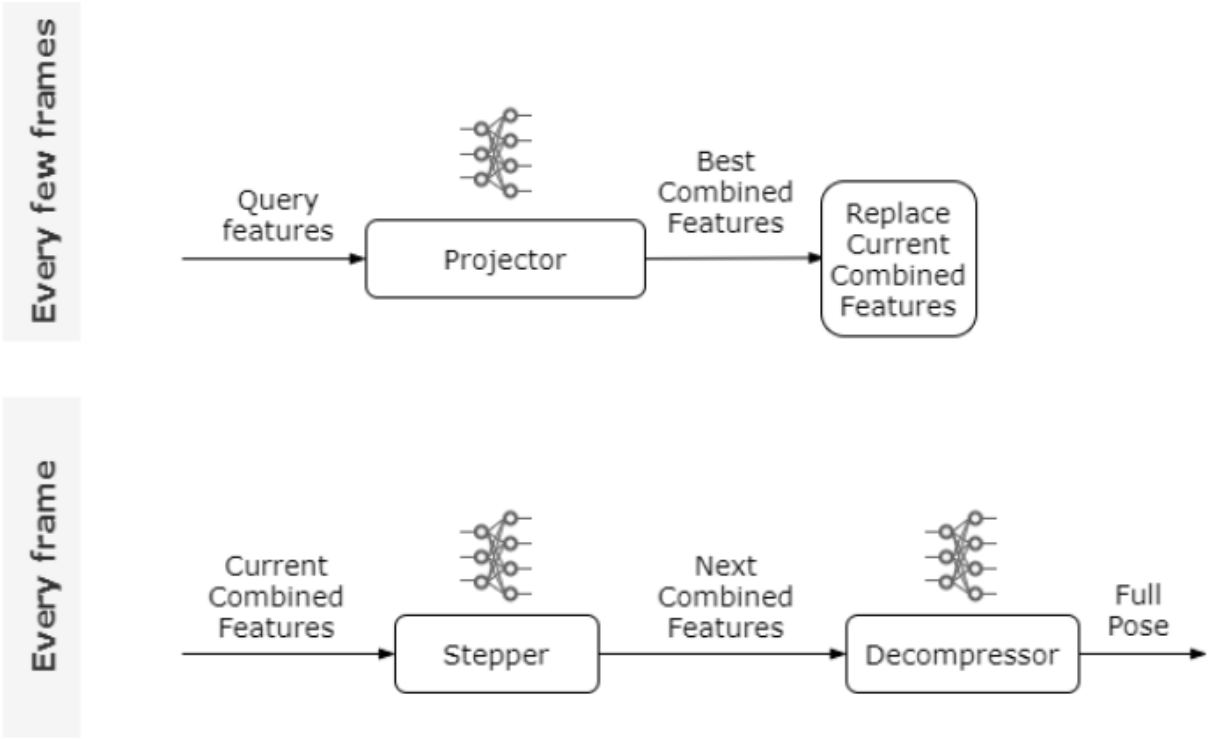
至此，内存中已不再需要存储动画数据与Feature数据
只需要存储三个网络模型



Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

Learned Motion Matching

- Decompressor：给定combined feature预测full pose
- Stepper：给定当前combined feature预测下一帧combined feature
- Projector：给定query预测当前combined feature



Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

Learned Motion Matching

	Pose Lookup	Decompressor	Learned Motion Matching
Animation Dataset	42.7 MB	-	-
Matching Features Database	9.4 MB	9.4 MB	-
Extra Features Database	-	11.1 MB	-
Decompressor Weights	-	0.9 MB	0.9 MB
Stepper Weights	-	-	1.2 MB
Projector Weights	-	-	3.2 MB
Total	**52.1 MB**	**21.4 MB**	**5.3 MB**

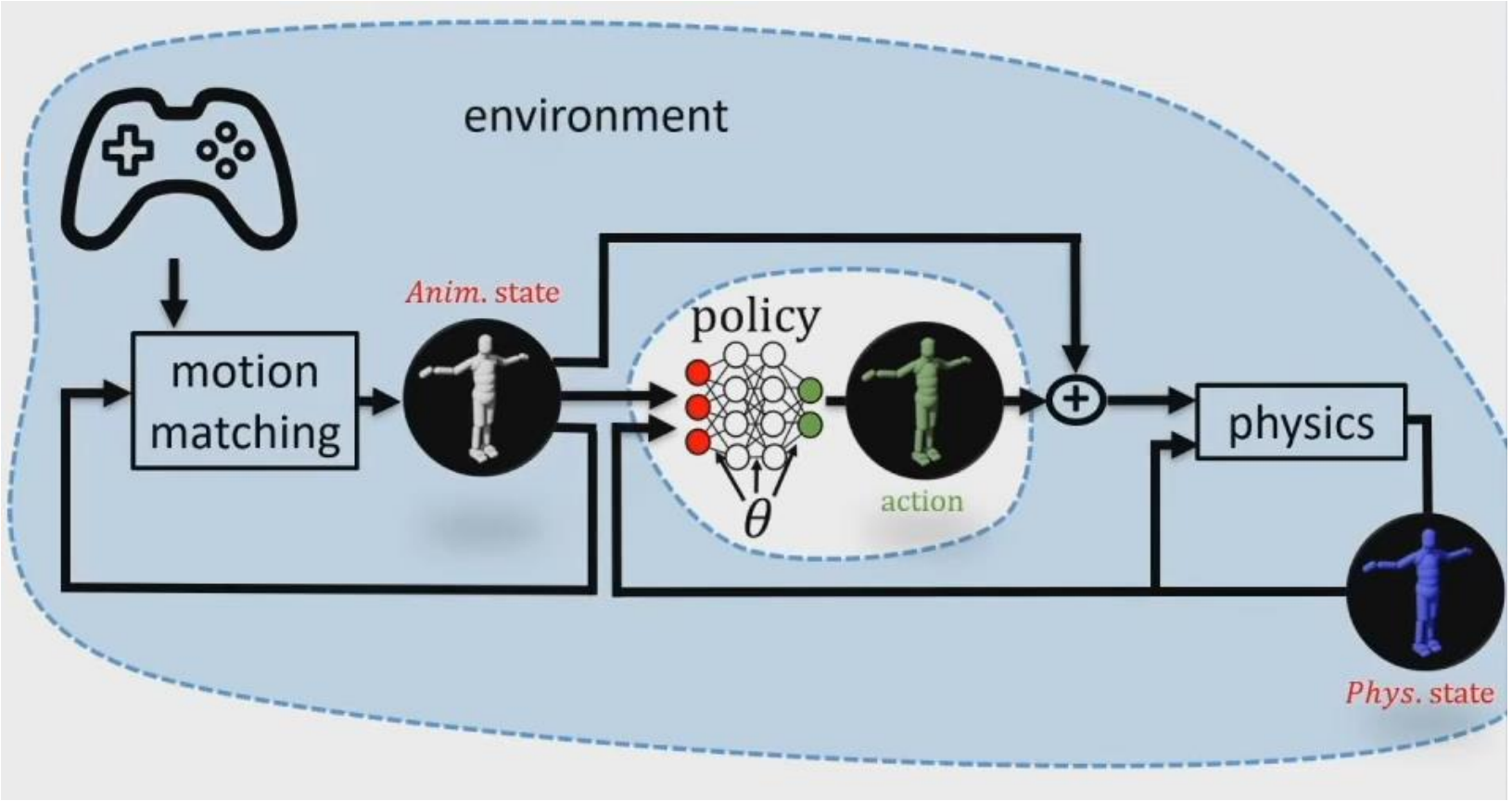
	Pose Lookup	Decompressor	Learned Motion Matching
Animation Dataset	461.0 MB	-	-
Matching Features Database	129.8 MB	129.8 MB	-
Extra Features Database	-	47.2 MB	-
Decompressor Weights	-	2.4 MB	2.4 MB
Stepper Weights	-	-	1.4 MB
Projector Weights	-	-	13.0 MB
Total	**590.8 MB**	**179.4 MB**	**16.8 MB**

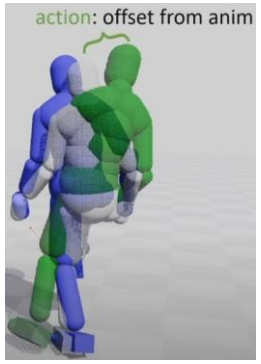
Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

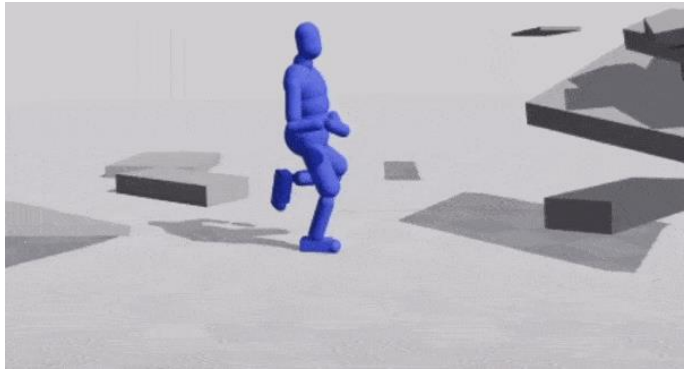
Learned Motion Matching

Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.

Using Reinforcement Learning







GDC 2020, Machine Learning Summit: Ragdoll Motion Matching, https://www.youtube.com/watch?v=lN9pXZzR3Ys

Motion Matching —— 概念与发展

Table of Contents

- 基于状态机的动画
- Motion Matching的引出与基本概念
- Motion Matching的要点与对比
- Motion Matching的发展
- 小结

Summary

- Motion Matching是近年来动画系统的一大发展，已经有越来越多的**大厂**开始使用这项技术，并取得了不错的效果
- Motion Matching的基本思想是根据玩家输入自动查找下一帧最匹配的动画，从而实现连续局部最优
- Motion Matching也存在许多不足，比如对数据要求很高、处理复杂度较大、难以应对复杂动作等等
- Deep Learning + Motion Matching是未来的一大发展方向
- 长期来看，仍然以State Machine为主

References

- **GDC 2016 - Motion Matching, The Future of Games Animation… Today, https://www.youtube.com/watch?v=KSTn3ePDt50**
- **[Nucl.ai 2015] Motion Matching - The Road to Next Gen Animation, https://www.youtube.com/watch?v=z_wpgHFSWss**
- **Realtime Animation Tips & Tricks**, https://rockhamstercode.tumblr.com/post/178388643253/motion-matching
- **Character Control with Neural Networks and Machine Learning, https://www.youtube.com/watch?v=o-QLSjSSyVk**
- Machine Learning Summit: Ragdoll Motion Matching, https://www.youtube.com/watch?v=lN9pXZzR3Ys
- Motion Matching (MxM) for Unity - Introduction & Overview, https://www.youtube.com/watch?v=j2HadMKiD9A
- **Machine Learning for Motion Synthesis and Character Control in Games, https://i3dsymposium.org/2019/keynotes/I3D2019_keynote_MichaelButtner.pdf**
- **Introducing Learned Motion Matching, https://montreal.ubisoft.com/en/introducing-learned-motion-matching/**
- Holden, Daniel et al. “Learned motion matching.” *ACM Transactions on Graphics (TOG)* 39 (2020): 53:1 - 53:12.
- Yi, Gwonjin and Jung-Geun Jee. “Search Space Reduction In Motion Matching by Trajectory Clustering.” SIGGRAPH Asia 2019 Posters (2019): n. pag.
- Holden, Daniel et al. “Phase-functioned neural networks for character control.” ACM Transactions on Graphics (TOG) 36 (2017): 1 - 13.
- Starke, Sebastian et al. “Neural state machine for character-scene interactions.” ACM Transactions on Graphics (TOG) 38 (2019): 1 - 14.
- Starke, Sebastian et al. “Neural animation layering for synthesizing martial arts movements.” ACM Transactions on Graphics (TOG) 40 (2021): 1 - 16.
- Starke, Sebastian et al. “Local motion phases for learning multi-contact character movements.” ACM Transactions on Graphics (TOG) 39 (2020): 54:1 - 54:13.
- Zhang, He et al. “Mode-adaptive neural networks for quadruped motion control.” ACM Transactions on Graphics (TOG) 37 (2018): 1 - 11.