Physics-based character animation controllers for videogame and virtual reality production

Joan Llobera*
joan.llobera@artanim.ch
Artanim Foundation
Geneva, Switzerland

Joe Booth
joe@joebooth.com
Independent Researcher
Seattle , USA

Caecilia Charbonnier
caecilia.charbonnier@artanim.ch
Artanim Foundation
Geneva, Switzerland

ABSTRACT

Physics-based character animation has dramatically evolved over the last five years. However, it is still difficult to use these techniques within the usual production pipeline. This is due both to technical factors, such as the fact that you need quite some specialised knowledge to set up and train these controllers, as well as to practical factors, such as the fact that the proposed solutions are difficult to integrate with content production tools. We present a physics-based animation system designed to work easily within a standard animation pipeline. The main benefits are three: it works within a standard game engine commonly used in videogame and virtual reality production (Unity3D), it supports the procedural creation of physics-based controllers from a rigged character with kinematic animations (and optionally a kinematic controller), and that it is available open source.

KEYWORDS

3D character animation, Physics-based animation, Deep Reinforcement Learning

ACM Reference Format:
Joan Llobera, Joe Booth, and Caecilia Charbonnier. 2021. Physics-based character animation controllers for videogame and virtual reality production. In Poster *MIG'21*. ACM, New York, NY, USA

1 INTRODUCTION

Physics-based character animation has dramatically evolved in the last five years. However, the integration of these techniques in video game or virtual reality production projects is still challenging. This is caused by a combination of factors, both practical and technical.

Currently, the main method used to create interactive animations in videogame or virtual reality productions is based on segmenting animation sequences, and defining points of transitions among them. Generally, these sequences and transitions are mapped to states and transitions in a state machine. The transitions are then triggered by binary conditions. In this way, animation cycles or one-shot animations are triggered by logical conditions, evaluated within the game loop, depending on custom game logic or by the input from the user. These state machines are available as animation engines, and are built-in modules in what are the *de facto* standards of video game production (Unity3D and Unreal Engine). A relatively recent alternative that has seen adoption by the video game industry is the use of *Motion Matching* [2]. Motion Matching uses a different principle, where possible transitions between a database of animations is pre-computed, on the basis of pose similarity. The benefit of this technique is that it provides much smoother transitions, and more diverse combinations of the reference animation fragments. Some rare games (for example, *Totally Accurate Battle Simulator*, by Landfall Games[1]), use physics-based animation, but those remain the exception, rather than the norm.

Here we introduce a new system for physics-based character animation that makes particular emphasis on integrating with established production pipelines. Our goal is to simplify the implementation of physics-based character animation. We place particular emphasis in developing a solution that works within the current production pipeline, leveraging the know how of technical artists. This implies a system that uses rigged characters animated with existing animation controllers, and modular enough to easily integrate different physics-based animation systems.

Our solution works within a standard game engine commonly used in videogame and virtual reality production (Unity3D), it is available open source[2] and it works with any kind of kinematic controller: we have validated the approach with a character that has a simple animation, an interactive controller based on a hierarchical state machine, and a motion matching algorithm. A modular architecture also simplifies exploring the main elements involved in a physics-based controller separately (reward function, early termination criteria, motor update method).

2 ANIMATION SYSTEM

2.1 Design Decisions

Our aim is to create a system that can work easily within the existing production pipelines. Aiming for this, we have developed an architecture that completely separates animations in the kinematic space and in the physical space. As such, each character is animated using four different skeletal hierarchies:

(1) **The reference character.** This rigged character renders the sequences of animations used as reference, as controlled by the animation controller. This can be in principle any animation controller, but we have evaluated our solution

*Corresponding author

Poster *Motion, Interaction and Games, November 2021, Lausanne (Switzerland)*

[1] https://landfall.se/totally-accurate-battle-simulator
[2] https://joanllobera.github.io/marathon-envs/

with the hierarchical state machine built in Unity3D and an off-the-shelf available motion matching animation engine.
(2) **The reference rag doll.** A rag doll that translates these movements to rigid bodies, used as targets for the physics controller.
(3) **The rag doll controller.** A rag doll controlled by the physics-based controller that applies forces to constrained rigid bodies, trying to match the orientations found in the previous rag doll.
(4) **The target animation rig.** This translates the outcome of the physics-based controller to cinematic trajectories applied on a copy of the original character, on which the cinematic controller was applied. This last step is generally absent from physics-based animation systems, and it significantly simplifies the use of those in production projects.

We keep a separate skeleton for each step of the process, and process them sequentially: the output of moving the first skeleton is used for the second skeleton, etc. This simplifies considerably the debugging of the system, and the adjustment of different parts. In addition, the skeletons related with cinematic animation and with physics can be updated in separate threads (typically, these are updated in the main update loop and in the physics loop, respectively).

2.2 Creation of the physical controller

To transform the reference animations to the space of physical actions, train the physics controller, generate the actions, and then transform the resulting movements back to the space of the rigged character the simplest, most robust strategy seems to have, as much as possible, a one to one correspondence between the different parts. To do this, we generate procedurally the two hierarchies of rigid bodies in order to match exactly the hierarchy of joints in the rigged character, and do a copy of the rigged character to render the outcome of the physics-based controller in a skinned character. This last body is the final animation outcome, and its pose is the direct application of the rotations of the rag doll controller.

The rigid body articulations are placed at the same position that the joints of the skinned character. Colliders are placed between the joints. The height of the collider corresponds directly to the distance between the joints, and the width is fixed. For the joints forming the spine, colliders are placed horizontally, the width of the collider is determined by the distance between spine joints and the height is calculated from the distance between the spine and the shoulder articulations. The result is a ragdoll that has roughly the same volume than the skinned character. The weight of each rigid body is calculated from a fixed mass density ($1000kg/m^3$), together with the volume of the colliders. The weight was placed in the middle of the rigid body. Finally, to ensure a perfect match between the two rag dolls, the second one was created procedurally, from the first.

To determine the degrees of freedom N_{dof} in the different joints and to fix the articulation limits for each joint we let the kinematic character move with the reference animations. From these movements we determine the range of motion of the physical actuators decomposing the rotations in twist and swing components. We determine the degrees of freedom for each limb by considering every range of motion in every articulation that goes above a certain threshold (5º). Finally, for each degree of freedom we configure the range of motion of the physical actuators to encompass the range of motion of the parsed animations, adding +10º and -10º to the maximum and minimum values of the range of motions. This is done to give extra space to the physics controller, otherwise the result of the training shows movements that are too rigid. The space of observations is also calculated automatically with the following formula:

$$N_{obs} = N_{dof} + N_{colliders} * 6 * 2 + N_{stats} \quad (1)$$

In the previous formula, N_{dof} is the degrees of freedom previously reported, $N_{sensors}$ is the number of elements that detect physical contacts (2 per feet), $N_{colliders}$ is the number of colliders placed between the articulation joints in the ragdoll (in our case, 15), and N_{stats} is the number of additional measures used in the final reward designed. The result of generating our physics controller automatically is comparable to previous papers (see Table 1).

Model	Rigid Bodies	Degrees of Freedom
Peng et al. 2018 [3]	13	34
Bergamin et al. 2019 [1]	23	54
Our system	22	49

Table 1: A comparison of our procedural system with previous physics-based animation systems

3 RESULTS

Our solution has been tested on the main OS platforms (Windows, OS X, ubuntu) and it trains successfully. Once the training environment is created procedurally, the motor update is done with simple PD controllers, as available within the Unity3D engine. we have also explored using the feedback loop found in [1], although not found significantly different results. To facilitate comparison, we use the same rewards and training parameters of DReCon [1].We hope different people will use this tool to explore how to integrate their animated characters with a physics controller, and that it simplifies its adoption in video game and virtual reality productions. In our experience, the main factor affecting training is the number of observations, and much less to the reward terms. We also hope this will help having a common ground for the comparison of different techniques for physics-based animation controllers and that other researchers and hobbyists to explore novel techniques for interactive character animation.

REFERENCES

[1] Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. 2019. DReCon: data-driven responsive control of physics-based characters. *ACM Transactions On Graphics (TOG)* 38, 6 (2019), 1–11.
[2] Simon Clavet. 2016. Motion matching and the road to next-gen animation. In *Proc. of GDC*.
[3] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.