High Performance Animation in Gears of War 4

David Bollo
Microsoft
dbollo@microsoft.com

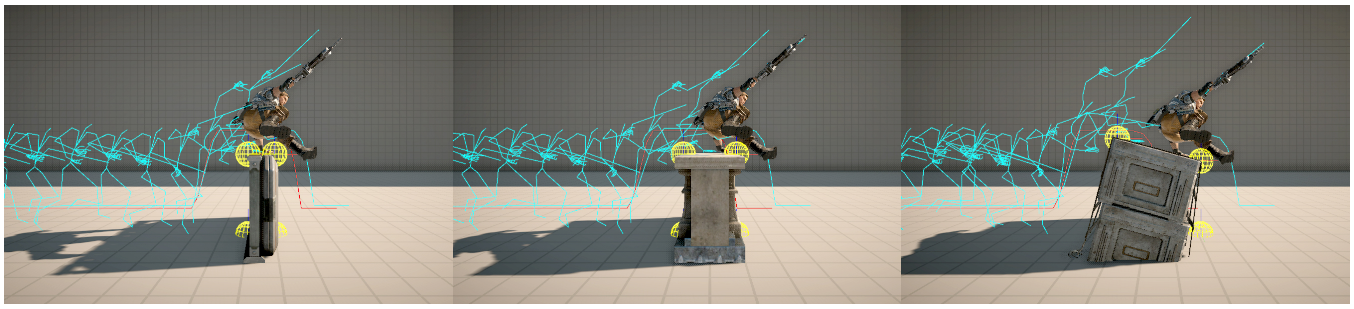

Figure 1: Warping an animation over obstacles of varying height, width and slope.

ABSTRACT

In this talk, we present three of the techniques that we developed to deliver high performance animation in the Gears of War 4[1] video game. First, we present a "warp point"-driven system for dealing with character traversal through an irregular environment. This system builds on previous motion warping work and is over twice as fast as a more traditional blend space based approach. Next, we introduce a novel approach to handling motion transitions that eliminates traditional blended transitions and replaces them with an animation post-processing step that is 60% cheaper to compute overall. Finally, we introduce a fast but effective heuristic for improving the quality of these motion transitions by automatically matching the locomotion foot phase between the transitioning animations.

CCS CONCEPTS

• **Computing methodologies** → **Animation**;

KEYWORDS

Animation, Gears of War

ACM Reference format:
David Bollo. 2017. High Performance Animation in Gears of War 4. In *Proceedings of SIGGRAPH '17 Talks, Los Angeles, CA, USA, July 30 - August 03, 2017,* 2 pages.
https://doi.org/10.1145/3084363.3085069

[1]Gears of War 4 Copyright © 2016 Microsoft Corporation. All Rights Reserved.

1 MOTION PATH WARPING

In Gears of War 4, characters must traverse through a varied and complex environment. To believably interact with this environment, characters must adjust their animations to conform to nearby objects. For example, when vaulting over an obstacle as in Figure 1, a character must be prepared to deal with variations in height, width and even slope.

Our system builds on previous motion warping work from Gleicher [Gleicher 2001] and Lockwood et al. [Lockwood and Singh 2011], but, unlike in those methods, the animators do not directly warp the animations. Instead, at authoring time, the animators augment their animation scenes with markers ("warp points") that correspond to physical landmarks in the environment. At runtime, the game analyzes the environment and places appropriate runtime warp points in the world (shown as yellow spheres in Figure 1). Then, when the animation system is playing back a warpable animation, it modifies the character's trajectory in the world to align the authored warp points from the animation with the runtime warp points in the world. For each warp point in the scene, the animators have independent control over the timing of when to warp the character's translation, rotation and facing towards its target.

Compared to traditional blend space [Epic Games 2017] based approaches, our approach is significantly faster, as it only needs to evaluate a single animation sequence, whereas blend spaces must decode multiple animation sequences to produce a single frame of output. A 1-dimensional blend space evaluates two animation sequences (typically 20 µs each on Xbox One) and blends the result (an additional 10 µs), for an overhead of 150% compared to evaluating a single animation sequence by itself. A 2-dimensional blend space evaluates three animation sequences and two blends for a 300% overhead compared to evaluating a single animation sequence by itself.

2 MOTION TRANSITIONS

Blended motion transitions are another significant source of computational cost in most video game animation systems. While the blends themselves are relatively cheap, the system must evaluate both the source and target animation states for the duration of any transitions, effectively doubling the computational cost during this time.

In Gears of War 4, we eliminate blended transitions altogether and choose instead to handle motion transitions as an animation post-process. Eliminating blended transitions gives us a significant performance boost as it is no longer necessary to evaluate the source animation during a transition; we simply cut immediately to the target animation and let the post-process clean up the results.

A straightforward approach to handling transitions as a post-process is to save the last pose of the source animation and then smoothly interpolate from that pose towards the (moving) target pose for the duration of the transition. However, while this gives pose continuity, the velocity is noticeably discontinuous, as can be seen by the unevenly spaced lines in Figure 2b. If we save one additional pose from the source animation, then we can also match velocity; however, this can lead to overshoot when transitioning from a fast-moving source animation. This is evident in Figure 2c, where the character's weapon swings too far down and to the right before finally recovering back to its target position.

Our method (Figure 2d) matches velocity while guarding against overshoot, guaranteeing good results even in the presence of fast-moving source animations. At the start of a transition, we record the source animation's velocity and the difference between the source and target poses. We construct a quintic polynomial interpolating curve that matches the pose difference and velocity at the beginning of the transition and converges to zero by the end of the transition. We then simply add the curve to the target animation for the duration of the transition. By controlling the curve's initial acceleration, we are able to enforce an overshoot-free transition.

Because our method does not need to evaluate the source animation state during the transition, our approach is significantly faster overall than the traditional approach, even though our post-process transition curves are slower to evaluate than a simple cubic blend (12 µs vs 10 µs). Consider that, even in the simplest case where we are blending from only a single animation sequence, the 12 µs overall cost of our method is 60% less than the 30 µs overall cost of the traditional approach (20 µs to evaluate the source animation sequence + 10 µs to evaluate the blend).

3 PHASE MATCHING

When transitioning back into a cyclic locomotion animation, if the target animation's foot phase is mismatched with the source animation's foot phase, then the resulting transition will exhibit unnatural footwork.

For Gears of War 4, we developed a simple but effective heuristic for automatically matching locomotion foot phase for bipedal characters.

We first developed a simple expression to compute a scalar "phase quotient" from the position of the character's left foot relative to its right foot, projected against its velocity (or its "forward" direction if it is stationary). This quantity oscillates approximately sinusoidally

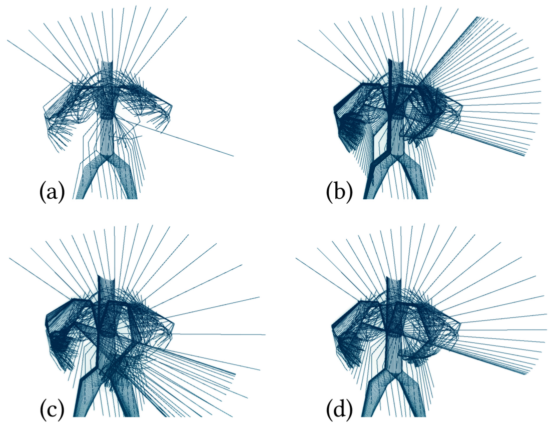

Figure 2: A comparison of post-process motion transition techniques. In all 4 images, the character is swinging his weapon overhead and then transitioning to his final position with the weapon pointing down and to the right. The original motion with no post-process is shown in (a). In (b) the character matches the pose but not velocity. In (c) the character matches velocity but then overshoots his target and has to swing back. Our method (d) matches velocity without suffering from overshoot.

as the character moves through its locomotion cycle. In order to distinguish between the incoming and the outgoing phases of the cycle, we negate and offset the phase quotient during the incoming phase. This way the phase quotient increases continuously throughout the entire locomotion cycle and only resets when the cycle itself resets.

During game startup, we compute phase quotients for each animation frame of our main locomotion cycles and save these in a table. After filtering the table to ensure that the phase quotients are monotonically increasing, we invert it and approximate it with a piecewise linear curve, such that we now have a simple mapping from phase quotient back to locomotion cycle frame number.

At runtime, when we are transitioning from a non-locomotion state back into locomotion, we simply compute the phase quotient for the character's current pose and then use the previously computed piecewise linear curve to map this phase quotient back to its corresponding locomotion cycle frame number. We then begin our locomotion state at that frame.

REFERENCES

Epic Games. 2017. Unreal Engine Blend Spaces. https://docs.unrealengine.com/latest/INT/Engine/Animation/Blendspaces/index.html. (2017).

Michael Gleicher. 2001. Motion Path Editing. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics (I3D '01)*. ACM, New York, NY, USA, 195–202. https://doi.org/10.1145/364338.364400

Noah Lockwood and Karan Singh. 2011. Biomechanically-Inspired Motion Path Editing. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '11)*. ACM, New York, NY, USA, 267–276. https://doi.org/10.1145/2019406.2019442