The Right Foot in the Wrong Place

Half-Life Character LocomotionCharacter Locomotion in Half-Life: Alyx

Joe van den Heuvel
Software Engineer, Valve Software
joev@valvesoftware.com

James Cunliffe
Animator, Valve Software
jamesc@valvesoftware.com

Eddie Parker
Software Engineer, Valve Software
eddie@valvesoftware.com

ABSTRACT

This paper describes the non-player character locomotion system developed for the VR game Half-Life: Alyx. Our solution uses a stride retargeting system, footstep prediction, and a custom motion matching system to animate humanoid and non-humanoid characters as they navigate tight, dense virtual environments in real time.

CCS CONCEPTS

• **Applied computing** → Computers in other domains; Personal computers and PC applications; Computer games.

KEYWORDS

Animation, Virtual Reality, VR, Games

ACM Reference Format:
Joe van den Heuvel, James Cunliffe, and Eddie Parker. 2021. The Right Foot in the Wrong Place: Half-Life Character LocomotionCharacter Locomotion in Half-Life: Alyx. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '21 Talks), August 09–13, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3450623.3469664

1 INTRODUCTION

Character movement in virtual reality presents some unique challenges for character locomotion, while also requiring high animation quality. Our solution consisted of 3 main parts. The first part is a stride retargeting system that would analyze the motion of the character's feet offline, store it as a direction-independent trajectory, and then use that information at runtime to modify the steps of the character to ensure that they traveled exactly along a given path. The second part was an enhancement to the game's existing animation system to leverage the stride metadata to predict the location where the feet will land next and modify it by adjusting the root motion of the animation. And the third part was a motion matching system that would continuously pick the best animation to play based on the desired path and the phase of the current stride.

2 STRIDE RETARGETING

Our Stride Retargeting system is based on the semi-procedural locomotion system described by [Johansen, 2009]. We improved upon it by developing a more robust step detection algorithm, support for multiple strides in the same animation, support for non-linear root motion, including the horizontal rotation of the foot as part of the stride data, and a novel storage format that handles stride overshoot and in-place steps.

The system works by first defining a 'footbase' as a shadow of the foot that remains horizontal, and the lowest part of the character's foot must always be in contact with it (See Figure 1).

As an offline process, we would find all the strides taken by all the character's animations and record the frame, position, and rotation of the footbase relative to the root of the character at the start and end of the stride. Then we would calculate the position and rotation of the footbase relative to the stride for each frame of the animation. These footbase "trajectories" would be stored to disk alongside the animation data as Progression value, and a position and rotation offset. Progression is the interpolation value between the previous and next footstep, and TranslationOffset and RotationOffset are the stride-relative offsets from the interpolated position and rotation.

Given a set of locations in the world for a previous and next footstep, the flight of the foot could then be recreated each frame by interpolating the position and rotation of the previous and next footsteps based on the Progression and adding the offsets. The resulting motion preserves the general motion and timing of the footbase from the animation even if the direction, length, and height of the stride are different from the original.

3 FOOTSTEP PREDICTION

At runtime, we would use the foot motion metadata to look up the time when the next footstep would land based on the current animation, and then how much distance the character would travel during that time. We would then predict the future location of the character in the world by advancing their current position along their current path by the calculated distance. Body yaw rotation is also considered based on the current traversal mode of the character, such as whether they should turn to face down the path or maintain a heading that allows them to face a target. From the predicted character position and rotation, we would use the character-relative position and rotation offsets for the end of the current stride that were saved as part of the offline pre-process to determine the horizontal location of where the foot will be when it lands and perform a ray-cast against the ground to determine its height. With both previous and next footstep positions known, we could then animate the position of the footbase over time from one to the other by calculating its position for each frame based on the precalculated trajectory data and use the footbase as an IK target to animate the leg. The effect is that the footsteps follow along the path, while preserving the characteristics of the original animation.

SIGGRAPH '21 Talks, August 09–13, 2021, Virtual Event, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8373-8/21/08.
https://doi.org/10.1145/3450623.3469664

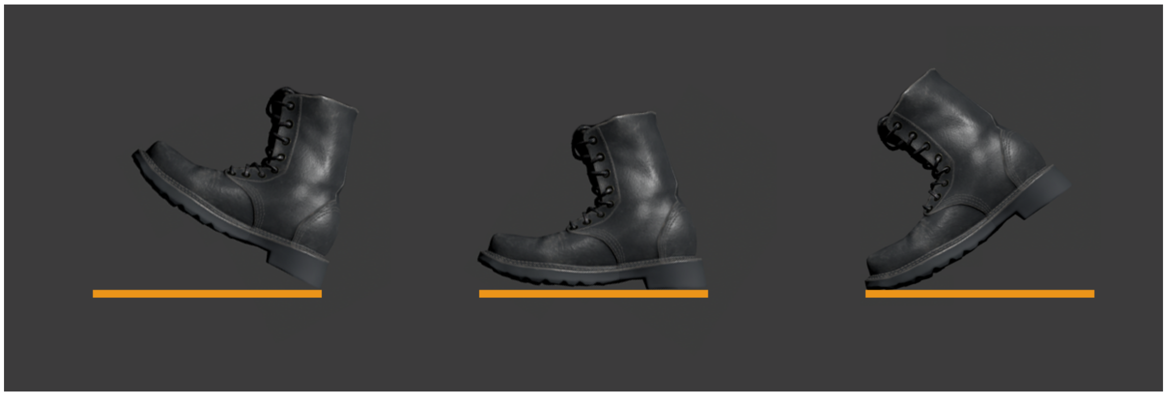

Figure 1: Illustrating the relationship of the foot to the footbase. The lowest part of the foot always touching the footbase. This way the foot can land and take off without sliding while the footbase is stationary.

Using this technique, we could slow the character's speed without introducing any foot sliding simply by scaling the amount of root motion of the animation. This was useful in cases such as going up and down steep slopes or turning sharp corners. By scaling the vTranslationOffset value from the footbase trajectories, we could effectively reduce the height of the footsteps in proportion with the amount we scale the root motion, so the amplitude of the strides matched the length. We would also scale the root motion so that run-to-stop animations would end exactly at the end of the path, with the feet coming to rest at the correct locations without sliding.

This system applied equally well to creatures with more than two legs, and all the nuance of the original animations was preserved, including any intentional foot sliding, twisting, jumping, and even kicks.

4 MOTION MATCHING

The stride retargeting allowed us to make any of our animations walk in any direction, but extreme changes in direction from the original animation would yield unnatural movement. The best results were when the original and the altered movement directions were relatively close. We needed another system that would automatically pick animations to play that were 'close enough' to the direction the path was going, which is exactly what motion matching does: search the frames of the set of available animations and find one that most closely matches the current state of the character and some desired state such as a future position or velocity.

The motion matching implementation described by [Clavet, 2016] and [Zadziuk, 2016] suggested calculating multiple samples of the desired future positions along the navigation path for the character based on a spring equation to simulation the acceleration of the character over time. These samples would then be compared to the locations where each animation would take the character after the same time intervals. In practice, we found that the acceleration rates in our data set was so varied that we were not able to come up with settings for the spring equation that produced samples that matched all the acceleration curves of our data, resulting in the search failing to correctly pick the best available animation. Instead, we created the future position samples based on distance instead of time. I.e.: sample the path ahead at several fixed distances, then sample where the animations would be after having moved the same distance, regardless of how long it would take them to get there. Doing this produced much more accurate results when scoring animations for how well the fit the desired future state.

We augmented our search algorithm to include filters as well as metrics. Filters would work by rejecting a sample that was outside a certain range, which was useful for forcing the matching algorithm to do the correct thing in problem situations. For example, we added a data channel to the motion matching data set that contained the distance remaining in the animation before it came to a stop. We could then apply a filter to the search so that all animation samples that did not reach the end of the path would be rejected, ensuring that the search would return a clip that would take the character all the way to the end.

We were able to make use of the footstep metadata to help the search select good transition points as well. Specifically, we could search for clips that matched the current stride progression and the position of the foot. We also limited the times when the motion matching system picked a new clip to when a foot was on the ground, as this allowed us to avoid awkward transitions where the character appeared to change direction in mid-air.

REFERENCES

Rune Skovbo Johansen. 2009. Master's Thesis "Automated Semi-Procedural Animation for Character Locomotion"

Simon Clavet. 2016. "Motion Matching – Road to Next-Gen Animation", *Game Developer's Conference* 2016

Kristjan Zadziuk, 2016. "Motion Matching – The Future of Gameplay Animation. . . Today", *Game Developer's Conference*, 2016