Procedural Pipeline: Procedural animation and flock behaviour using *Unity*

Alba Reinders Sánchez
Technical University of Denmark (DTU)
Copenhagen, Denmark
s212729@dtu.dk

Abstract

The proposed project involves the implementation of a procedural pipeline, for the purpose of 3D character modelling, rigging, and animation using procedural techniques. The final output of the pipeline includes a flock behaviour of the characters to demonstrate the results. The implementation of this pipeline utilized *Blender* for the creation of procedural models and bone structures, while *Unity* was employed for the development of procedural animation and the flock algorithm. We developed an automated pipeline that enables simplified character and procedural animation development for populating 3D worlds. This report focuses on the *Unity* steps of the pipeline, including the setup of different components and scripts, as well as the implementation of the flock algorithm.

1 Motivation and Goals

The initial inspiration for this project comes from the desire to simplify the implementation of procedural animation, and automate the process for developers seeking a tool to facilitate 3D model animation. With this goal in mind, we developed the proposed pipeline that combines 3D modelling and rigging of characters with procedural animation. Culminating in the simulation of a flock behaviour to visualize the performance of the animated characters within a given environment.

We recognize the potential of procedural animation as a powerful but often labor-intensive method for animating 3D models. With the aim of simplifying the process, we try to create a pipeline that minimizes user input by allowing for the creation of procedural animation setups and code to be carried out only once, and subsequently be reused for all variations of 3D characters based on an initial one. The *Unity* part of the pipeline was designed with this goal in mind, in addition to the implementation of the flock algorithm.

The goal is to employ procedural animation techniques to animate the legs of 3D characters made in *Blender*, which have varying shapes and leg lengths relative to their body proportions. Once the procedural animation has been applied, the animated characters will be incorporated into an environment with various obstacles and assigned a flock behaviour.

The full project can be found in the GitHub repository: Pheithar/procedural-animation-flock.

2 Methods

This section provides an overview of the two main methods used in the second part of the pipeline. The first method is the implementation of procedural animation, including its automation. The second method involves the implementation of the flock algorithm and the interconnectivity between the *Blender* part of the pipeline and the final output, where the animated characters are visualized.

2.1 Procedural animation

Procedural animations are a valuable tool for animating 3D models in video games and interactive environments. Unlike traditional key frame animations, procedural animations can consider physics and interact with the environment and terrain. However, their creation typically demands a significant amount of time and effort, and designers may lack the necessary control over the animation process. In our project, we seek to address the challenges associated with the creation of procedural animations, and we intend to use *Unity* to develop and work with procedural animations.

We used as guide for our design an article [3] that implemented procedural animation in *Unity*. Given our plan to employ this software, the article proved to be a valuable resource, serving as a starting point for our approach.

The first step involved manually preparing the procedural animation for the legs of one of the 3D characters. After importing it and its rig into *Unity*, we created a *Unity* custom armature based on the original rig and added several additional components. Each leg comprised three bones, namely the shoulder, anterior, and posterior bones. For each leg, we included a target element for the end bone (foot) and a hint element for the joint (elbow), where the anterior and posterior bones intersected. These elements were critical for moving the 3D character and generating the walking animation, they are shown in Figure 1.

The process of coding the procedural animation entailed the creation of a script that was attached to each leg, which contained instructions on how to move the leg, while a main script controlled which leg to move at any given moment. To determine which leg to move, we first considered the default position of the armature and offset it based on the current transform of the 3D character. We then calculated, for each leg, the desired position, taking into account the current placement of the 3D character and how far it was from the

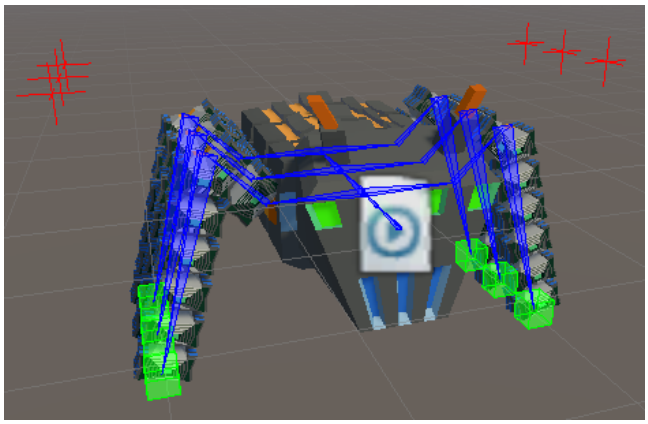

Figure 1. Rig and animation setup created in *Unity*. Armature (blue skeleton), targets (green cubes) and hints (red crosses).

target position. The legs that were moved were the ones with the greatest distance between these two positions, triggered by taking a step in the direction towards the desired position.

After successfully creating the procedural animation for this specific 3D character, we proceeded to develop the necessary code to automate the entire process for any other bot design.

2.2 Flock algorithm

The flock algorithm was implemented to test the capabilities of the procedural animation and to explore potential uses of the population. The algorithm is based on a simulation of flock behaviour in nature, where each individual is called a 'Boid' [2].

The Boids have a limited view range and follow a simple set of rules. The first rule, 'Separation', requires each Boid to steer away from any other Boid that is too close, and also includes collision detection with obstacles. The second rule, 'Alignment', instructs each Boid to match the velocity and direction of its neighbours, while the third rule, 'Cohesion', directs each Boid towards the center of mass of all its neighbours. These rules produce a realistic flock behaviour, as depicted in Figure 2.

3 Results

The proposed pipeline for creating procedural animations and implementing a flocking algorithm in *Unity* was successfully implemented and tested. The pipeline was able to generate walking animations for 3D characters of varying leg lengths and body shapes, as well as simulate realistic flock behaviour using the flock algorithm. The automated nature of the pipeline allowed for rapid generation of animations and facilitated experimentation with different designs and parameters.

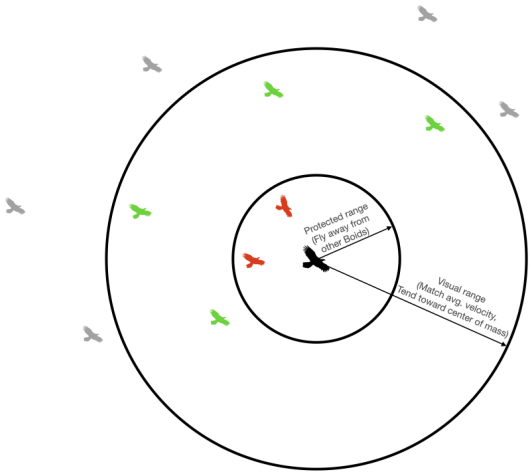

Figure 2. Flock algorithm overview [1]

After testing the pipeline, we designed a 3D terrain and placed a few obstacles to evaluate the capabilities of the procedural animation and flock algorithm. However, we encountered limitations with our prototype, as we found flaws in the interaction between the colliders in the Boids and the terrain, which resulted in undesired behaviour when using bigger 3D characters, because their larger proportions made it more difficult to move smoothly. These limitations suggest that further improvements are needed to ensure the proper interaction of the Boids with the environment. Nonetheless, the pipeline was successful in generating realistic flock behaviour for the 3D characters and showed potential for future development.

In Figure 3, we display our testing environment with an example of a population, following a flock behaviour.

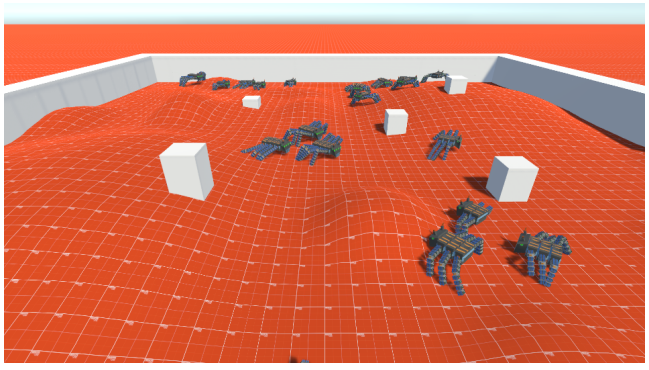

Figure 3. Capture from flock of bots in the environment

References

[1] V. Hunter Adams. n.d.. Boids algorithm - augmented for distributed consensus. Retrieved from https://vanhunteradams.com/Pico/Animal_Movement/Boids-algorithm.html. (vha3@cornell.edu).

[2] Iain D. Couzin, Jens Krause, Nigel R. Franks, and Simon A. Levin. 2005. Effective leadership and decision-making in animal groups on the move. *Nature* 433, 7025 (2005), 513–516. https://doi.org/10.1038/nature03236

[3] Mina Pêcheux. 2022. *Creating procedural animations in Unity/C#*. https://medium.com/codex/creating-procedural-animations-in-unity-c-8c5c2394739d