POLITECHNIKA BIAŁOSTOCKA

WYDZIAŁ INFORMATYKI

PRACA DYPLOMOWA INŻYNIERSKA

TEMAT: SKELETAL ANIMATION USING INVERSE KINEMATICS IN THE UNITY ENGINE

WYKONAWCA: ŁUKASZ BIAŁCZAK

.....

podpis

PROMOTOR: DR INŻ. ADAM BOROWICZ

BIAŁYSTOK 2022 r.

Contents

1. Introduction

1.1 Motivation

Animation is the technique of displaying different positions of a character or object in rapid succession to create the illusion of movement. It is used in various forms of entertainment, such as movies and video games. In the latter, unlike in the former, the animation sequences are performed in real time and therefore impose additional constraints. Without the freedom to process a single frame for minutes or hours during the rendering of the scene, the animator must compromise on the quality and realism of the sequence in order to optimize for gameplay. One such optimization is the use of skeletal animation in which animation sequences are performed by manipulating a tree-like structure of interconnected bones, represented by transforms, to create the desired motion of the character. Furthermore, the interactive nature of video games makes it impossible for the artist to create predefined animation sequences for every possible situation that may occur in the game. As a result, predefined animation sequences are often generic and do not allow the character or object to interact naturally with their surroundings. Game developers have come up with many methods to improve the realism of animation in games such as playing cutscenes for critical interactions between a character and the world. However, this paper will focus on the use of procedural animation and, more specifically, the application of inverse kinematics to skeletal animations in video games.

Inverse kinematics is a technique used in fields such as robotics and computer graphics to determine the joint angles of a kinematic chain that will result in a particular part of the chain, usually an end effector, reaching a specified position in 3D space. In computer graphics specifically, the technique is used to animate the movement of characters and objects such that they interact with their surroundings in a more realistic manner.

There are multiple approaches and algorithms that exist within the inverse kinematics domain, such as analytical methods, gradient descent, and optimization techniques. The choice of approach varies depending on the complexity of the use case, the desired realism of the animation, and system limitations.

1.2 Problem Formulation

The aim of this dissertation is to gain a better understanding of the basic algorithms used in inverse kinematics, discover the built-in functionalities that the Unity engine offers for such implementation. The project implementation will apply these concepts to create pairs of animations which consist of baked and inverse kinematics variants. The use cases will expand the problem by introducing additional constraints which will be required to keep the consistency and realism of the animations. The variations will then be compared through the lens of realism and performance.

The author will begin by discussing the theory of the different approaches and algorithms used to solve the inverse kinematics problem, and the resulting choice of the algorithm to be used in the project implementation. The following sections will explain in depth the implementation of two use cases which demonstrate the purpose of inverse kinematics as a skeletal animation technique. Experiments will then be conducted to compare the inverse kinematics animations with their baked counterparts based on realism and performance. Finally, a summary and conclusion of important points will be presented to the reader.

2. Related Work

2.1 Overview

2.2 Use Cases

2.3 IK Algorithms

2.4 Advantages and Disadvantages

3. Inverse Kinematics in the Unity Engine

The demo application written for the purpose of this dissertation includes two separate use cases of skeletal animation using inverse kinematics in the Unity engine. The first example is that of a four legged spider which uses the technique as a means to more naturally adjust its limbs to the terrain it moves around upon. The second example is the application of inverse kinematics to an animation sequence of a human character pressing multiple buttons in succession. The use of inverse kinematics allows the character to adjust its animation to hit all the buttons without the need for a baked animation targeted towards each button, as well as dynamically adjust the order of the buttons to be hit.

Although there are two separate use cases demonstrated in this application, both use the same implementation of the FABRIK algorithm.

3.1 FABRIK implementation

This implementation of the FABRIK algorithm is based on the paper written by the author of the Unity engine FABRIK implementation. (CITE THE PAPER LMAO). For the purposes of this application, the basic algorithm is implemented without many additional constraints and options available. The one constraint added on to this implementation is that of pole targets which will be further explained when discussing the code behind them.

The script which implements the algorithm in this project takes in a few parameters required to set up the mechanism. First and foremost, the root and leaf nodes must be provided in order to define the kinematic chain which is to be manipulated. The next object which the script must have knowledge of is the target transform which the end effector will attempt to move to. All the aforementioned parameters are required for the script to function. Additionally, a pole target object may be optionally passed in if the use case requires it to function in a manner that is desired.

Maybe throw in a snippet of the script which takes in parameters

The first case which the algorithm must cover is if the distance from the root to the target object is greater than the sum of distances between each adjacent bone in the defined kinematic chain. In this case, the target is out of reach. Given that the bones in a skeleton

are expected to keep a fixed length, the end effector will not be able to reach the target, and instead the kinematic chain straightens and extends in the direction of the target. There is no need to continue with the iterative portion of the algorithm.

In the implementation below, square magnitudes are used to avoid the calculation of square roots, thus slightly optimizing the initial check.

Show code

The scenario where the target is within the reach of the kinematic chain utilizes iterative forward and backward component after which the algorithm is named. Before each iteration, the positions of the joint transforms are copied. All operations and calculations are performed on these copied transforms and at the end of the full pass, the new positions are then copied back to the kinematic chain. **CHECK IF THE ALGORITHM WORKS WITHOUT THIS. MIGHT SAVE SOME MEMORY LMAO**. It is also important to note that the modification of the transforms is done in Unity's *LateUpdate* function. When using a mix of inverse kinematics and baked animation, the object to which the IK script is attached will have Unity's built-in *Animator* component. If the IK script updates joint transforms in the *Update* method, then they may be overwritten by the *Animator* component.

The forward pass of the algorithm iterates through the chain starting from the end effector and ending at the root. At the start, the end effector's position is set to be equal to the position of the target. A straight line is then drawn from the end effector to the following node. This neighbours position is then interpolated along the line so that the original distance between the two nodes is kept. The same operation is performed for each pair of neighbouring nodes throughout the pass.

show code

When the forward pass is complete, the root node is displaced from its original position. This is undesired, as the root's node position should not be affected by the algorithm. To remedy this, the next step is to repeat the forwards pass, but this time in reverse. The root node's position is set equal to what it was at the beginning of the frame. The next node is then interpolated between its current position and the root to keep the initial bone length. As with the forward pass, this is repeated for each subsequent pair of nodes.

show code

These two steps are repeated together until the end effector is within a threshold dis-

tance of the target. The FABRIK algorithm is a heuristic algorithm, and as such is does not lead to an exact result. Instead, it aims to approximate the correct solution and solves the problem in a less complex and more optimized way. Again, the square distances are used to avoid the calculation of square roots.

show code

POLE TARGETS. FIND A SOURCE WHICH EXPLAINS THE METHOD USED IN MY PROGRAM. IF NOT THEN REFER TO THE YOUTUBE VIDEO LOLOL

3.2 Spider Movement

The first use case for inverse kinematics in the demo application is that of a four legged spider. The algorithm is used to adjust the creature's limbs to uneven terrain, leading to a much more natural and realistic movement.

3.2.1 Project Setup

Each one of the spiders legs is treated as a separate kinematic chain. The spider prefab consists of a container which holds the spider object itself, set of empty objects to which the four IK scripts are attached for the purpose of easy yet separated access. The prefab also contains sets of ray casts and targets. The ray casts serve to scan the surface of the terrain under the spider, and mark the targets to which each leg should move.

Ray casts are dispatched from above the spiders legs, and aim in the creatures local negative Y axis. This ensures that no matter what orientation the spider finds itself in, the rays are always pointing at the surface which it is standing on. Masks are applied to the rays, making sure that only terrain objects are taken into account, while the creature's body itself is not. The ray cast hit point positions are then applied to each leg's respective target object. These targets serve as markers for the limbs end effectors.

3.2.2 Scripts

With the project set up in this manner, scripts must now be added on to make the scene functional. A script is required for the main movement of the spider, the ray cast logic, and **CHECK BACK LATER**.

Ray casts

The ray cast objects contain a script component which dispatches the rays and sets the appropriate target positions. First, a mask must established, which will then be passed into the ray cast operation. This is required so that only terrain is counted as a valid hit. The lack of such mask may result in unexpected behaviour, such as targets being set on the spiders body itself. The ray cast object is then created, shooting in the local negative Y axis direction. This ensures that no matter the orientation of the creature, the rays are sent towards the surface that the spider is standing on.

3.3 Human Animation Sequence

3.3.1 Project Setup

3.3.2 Scripts

4. Experiments

5. Conclusion

Bibliography

[1] Jasvir Nagra, Clark D. Thomborson, and Christian S. Collberg. A functional taxonomy for software watermarking. In *ACSC*, pages 177–186, 2002.

[2] ISO/IEC-9126, International Standard ISO/IEC. In *Information technology: Software product evaluation: Quality characteristics and guidelines for their use*. International Standards Organisation, 1991.

[3] http://www.tex.ac.uk/cgi-bin/texfaq2html?label=multirow.

[4] http://www.texample.net/tikz/examples/computer-science-mindmap/.

[5] http://www.ctan.org/.

[6] H. Partl i inni T. Oetiker. Nie za krótkie wprowadzenie do systemu LaTeX.

[7] Wojciech Myszka. W³¹czanie grafik do tekstów w LaTeX.

[8] R. Kostecki. W miarê krótki i praktyczny kurs LaTeX w π^e minut.

[9] Forum gust. http://www.gust.org.pl/.

[10] Wykresy - TikZ & PGF Manual for Version 2.00.