A Motion Planning System for Gibbon Brachiation

April 22, 2022

Jason Lee
Department of Visualization
Texas A&M University
College Station, Texas, USA
jzblee@tamu.edu

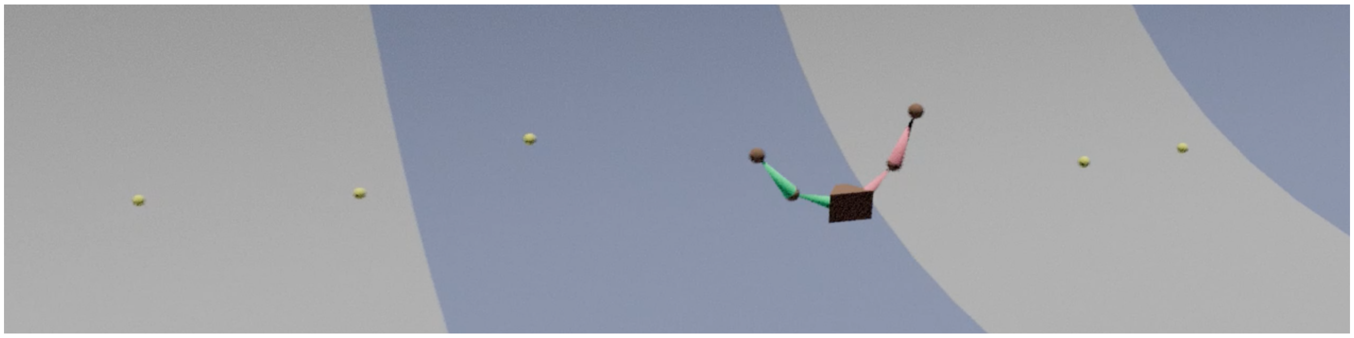

Figure 1: Procedural animation of brachiation

ABSTRACT

This paper presents two methods for authoring motion resembling gibbon brachiation in digital content creation (DCC) software: one using key-based animation in Autodesk Maya, and one using physically-based animation in Epic Games' Unreal Engine. By blending techniques from both approaches, brachiation may be achieved along an artist-defined path in 3D space.

KEYWORDS

motion system, physically-based modeling, constraints, animation, brachiation

1 INTRODUCTION

Digital content creation (DCC) applications have simplified virtual depictions of creature locomotion. This capability is in high demand in countless areas, from film to research. The list of techniques at artists' disposal is heavily spurred by developments in computer graphics. There is a similarly large list of patterns of animal movement to simulate. One such pattern is brachiation - a form of arboreal locomotion used by gibbons to swing between trees. Regardless of the motion being simulated, a hybrid system that uses computer control but still affords a great degree of artistic freedom is most desirable. To this end, the systems presented take a set of locations and solve for a path along which brachiation may occur.

2 BACKGROUND INFORMATION

2.1 Brachiation

Brachiation is the primary mode of locomotion used by gibbons. The motion is characterized by the grasping of tree branches (or "substrate") by alternating limbs and a graceful swinging motion. An analysis of brachiation by Zhang identifies six phases of the movement, in which the brachiator manipulates its limbs and center of gravity to successfully navigate a path. Although uncommon, a gibbon may fail to grasp the next handhold. If so, it attempts to swing backward and forward again, to cover the greater distance [6].

2.2 Previous Approaches

Zhang and Wong present a method using physically-based modeling to simulate brachiation with a 3-link model in the 2D space, i.e. each rotating joint in the model has one degree of freedom (DOF) [7].

Ufford uses MATLAB's differential equation solver to integrate equations of motion for a 2-link brachiating robot, also in 2D [5].

Berseth uses reinforcement learning in a case study on brachiation and generalizes the problem to 2D pendulum navigation [1].

2D is often chosen to keep the problem manageable; the equations for simulating a pendulum system, let alone an already complicated problem like brachiation, are extremely complex in 3D.

2.3 Animation Techniques

Predefined animation, key-based animation, and physically-based animation are three approaches to modeling creature locomotion.

Predefined animation is entirely created by the artist. Most character animation in feature films is done using this method. Although the artist works with pre-built rigs that define the range of motion, they can manipulate them to convey expression and emotion. DCC software may be configured to interpolate between the artist's defined key poses, but the artist is otherwise given a great degree of control.

Procedural animation delegates more of this control to the computer. For many animals, gaits can be expressed relatively simply. Human walking, for example, is a commonly assigned animation exercise due to its cyclic nature. Such an attribute leads to easier translation to DCC software and the expression of motion in simple formulae. Other approaches place even more emphasis on real-world reference. The gaits of quadruped animals have been studied extensively and are frequent cases for applying procedural animation, such as in analyzing animal footage to reconstruct gaits [3].

Physically-based animation incorporates concepts such as classical mechanics to model dynamic phenomena [4]. To that end, the resulting simulation takes into account the physical attributes of the object being animated, any internal and external forces, and the characteristics of the environment. Consequently, the technique lends itself best to the animation of objects such as hair and cloth. Ragdoll physics, another application of physically-based animation, is common in video games.

3 KEY-BASED BRACHIATION

A path for key-based brachiation has one important requirement: consecutive handholds should be at most two arm lengths apart. A desirable path should be a reasonably energy-efficient traversal between two points in space given a set of handholds (i.e. possible intermediate points). To that end, a shortest-path algorithm can minimize the distance that the brachiator must cover.

Handhold positions are randomly generated by dividing the scene space into smaller uniformly-sized volumes and performing stratified sampling. With two handholds in opposing corners of the scene space designated as the starting and ending points, a graph consisting of the set of all handholds and the set of edges representing each pair of handholds that are less than two arm lengths apart is constructed. Finally, Dijkstra's algorithm is used to generate the shortest path between the start and end points [2].

Due to the constraints and the nature of stratified sampling, there may not always be a valid path. If that is the case, a regeneration is advisable.

3.1 Path Animation

The rig used in this approach is a rigid model consisting of five separate parts and three pairs of joints for the hands, shoulders, and elbows. An inverse kinematics solver computes arm positions. Each important pose is generated programmatically and interpolated by the DCC application.

As described in the previous section, Dijkstra's algorithm yields the handholds that make up the shortest path between the starting and ending points. Analysis of brachiation footage identifies stages during which gibbons use both hands to grasp the substrate. We may use each consecutive pair of points in this list to solve for the body position during such contact poses.

A gibbon's arms approximately form the two equal sides of an isosceles triangle, with the third side spanning the two handholds. Since all three of the side lengths and two of the point positions are known, trigonometry may be used to determine the angles of the triangle. The cross product of one vector spanning the two handholds and another vector pointing straight down gives an ideal rotation axis. A vector of one arm length aligned to the spanning distance may then be rotated into position using the rotation axis and angles to calculate the correct position for the body.

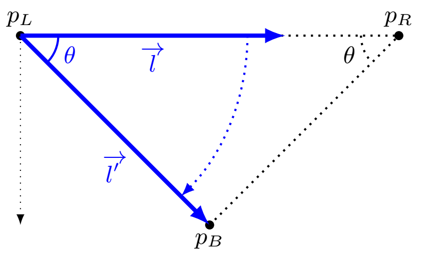

Figure 2: Using trigonometry and vector math to solve for the body position p_B at each contact pose.

Various correctives are made to enhance the quality of the animation, such as the lateral movement of the body during phases where the center of gravity has shifted to one side. In addition, the body is always kept at one arm's length from the contacted handhold at each frame, resembling a hard constraint. Finally, the free arm is animated to swing down and around the torso during each half-cycle.

4 PHYSICALLY-BASED BRACHIATION

4.1 Unreal Engine and PhysX

This project used the Early Access 2 build of Unreal Engine 5. While Epic Games' Chaos physics solver became the default option upon UE5's official release on April 5, 2022, this project continued to use Nvidia's PhysX, the previous default.

A large part of Unreal Engine's flexibility arises from the availability of both a node-based system (Blueprint Visual Scripting) or a code-based system for programming entity behavior. For the physically-based approach, C++ was used to generate an Actor with a simple component hierarchy and control movement at each engine update (or tick).

4.2 Physics Constraints

Physics-based simulations use constraints when a distance or angle should not change. Hard constraints enforce this requirement by correcting the system to resolve all inconsistencies at each tick, while soft constraints apply a compensatory force to each element instead.

Unreal Engine provides Physics Constraint components that allow for the creation of soft constraints between any two physics objects. A variant of this spring-like attachment draws the body and free arm towards the next handhold during each cycle.

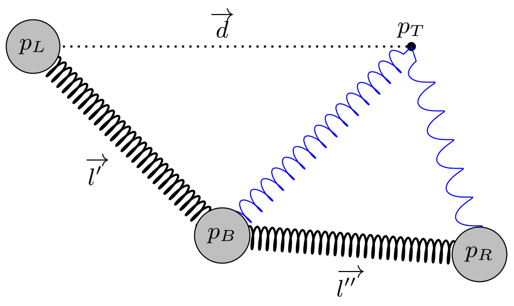

Figure 3: Using spring constraints to maintain the pendulum's shape and generate an appropriate force on the body p_B and free arm (currently p_R) so that they are drawn towards the target point p_T. The spring constant and rest length of the spring connecting p_T and p_R are much higher than those of the spring connecting p_T and p_B.

4.3 The Double Pendulum Brachiator

The rig consists of three spheres representing a body and the left and right hands. This geometry simplified testing, as PhysX calculates moments of inertia.

Each half-cycle of brachiation consists of one locked arm, and one free arm. Throughout the movement, the free arm swings from the previous handhold to the next one. Target spring forces are activated only when a free arm is detected to be at the bottom of its swing; that is when its Z velocity moves from negative to positive.

In addition, the brachiator keeps track of whether it is undergoing frontswing or backswing, based on when the free arm reverses direction at the tops of swings. Target spring forces are only activated in the case that the brachiator is in frontswing, and is in the process of moving up toward the target node.

One potential issue with keeping state this way occurs when the brachiator's free arm makes a complete loop around the locked handhold instead of swinging backward, as this results in an inconsistent state.

Throughout the entire movement, the velocity is also constantly being modified to bias motion in the direction towards the next handhold, and to diminish the effect of any motion that is orthonormal to this direction.

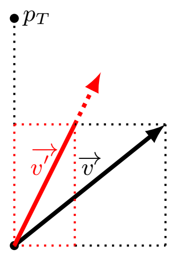

Figure 4: Modifying the velocity to decrease the component normal to the direction from the free arm to the target handhold.

5 DISCUSSION

5.1 Intended Audience and Applications

This tool was made to allow artists to more easily author brachiation in Unreal Engine's real-time environment. For the tool to be most effective, the intended audience should not need to worry about the underlying aspects of the code, the engine, the physics solver, or other low-level components. They only need to think about the specifics of the path that the gibbon should take as well as the distances between each handhold. Issues with the physics solving or interface can be referred to the author to fix. This allows for the abstraction of the entire motion of brachiation, reducing the artist's mental load.

This tool has applications in previsualization, as the specifics of each swing and the exact geometry of handholds are not locked. Just as predefined animation encompasses many methods, such as pose-to-pose, straight-ahead and layered, so can brachiation: via the key-based approach, the physics-based approach, or a combination of multiple approaches. In previsualization, the physically-based simulation may serve as a jumping-off point for refinements made later in the process (i.e. in layout and animation). This routine resembles that of motion capture, in which 3D tracking data is collected and then refined by animators.

5.2 Determinism and Tradeoffs

Another substantial point of discussion is the use cases of deterministic and nondeterministic simulations. The former gives reliably consistent output since the timestep always remains fixed. Although the state at one frame depends on the state in previous frames, using the same initial conditions is sufficient to guarantee the same output each time. However, nondeterministic solutions incorporate some form of adaptation. For example, the timestep may change because of variations in the state of the motion system and hardware. Bullet and PhysX are examples of deterministic and nondeterministic physics solvers, respectively.

The nature of a real-time environment like Unreal Engine, in which good performance is critical, makes the use of a nondeterministic system desirable. Deterministic physics systems are helpful for film, as each frame is predefined and can be rendered offline. Therefore, simulations can be performed with greater precision at the cost of greater simulation runtime. However, games necessitate a careful balance of visual fidelity and system performance.

Variations in the timestep can lead to instability in the simulation. When the brachiator changes handholds, the pendulum system needs to start rotating around a different fixed point. At the same time, the physics constraints are reconnected such that they point from the root down. This may lead to repeated attempts by the engine to reconcile the geometry of the individual bones, leading to instances where velocity is set to an abnormally high value.

For example, merely moving the mouse in the viewport during a simulation could drastically lengthen the timestep. Most of these problems were fixed by forcing the engine to use a higher minimum framerate, which also raised the time resolution of the simulation. Doing so came at the cost of reduced engine performance.

5.3 Adding Character

Stability issues aside, the challenge of balancing physics and artistic direction is paramount. A larger degree of control affords artists the ability to add personality to the movement. A common method for adding character to biped walks is to focus on the animation principle of exaggeration: to amplify the weight or personality of a character. However, several concepts that have been explored in much detail for biped walks will need to be redefined for brachiation, a movement largely resembling that of a pendulum. At present, characterization may be improved by tuning the simulation: the physics constraint components contain parameters that control the strength of various spring forces. Raising or lowering these values may be used to show vigor or lethargy, respectively.

6 FUTURE WORK

One possible extension is the implementation of a Backward Solve using Control Rig. Control Rig is a plugin that allows for bidirectional rig solving within an engine: a forward solve derives bone transforms from control transforms; a backward solve derives control transforms from bone transforms, allowing a rigged geometry to be mapped onto bone animation. Since the three joints present in the simulation could be mapped onto any rig control, the rig may introduce additional intervening joints such as shoulders and elbows.

Other improvements include the addition of more movement types, including traversals where both limbs are unlocked at some point (e.g. to cover larger distances). Another issue that arose was excess velocity leading to the free limb making a full loop around the locked point, which could be helped by damping the velocity or adding an upper limit.

In addition, many examples of reference footage show gibbons traversing structures that are also moving in turn; they are affected by external forces like the wind, or by the force imparted by the gibbon's movement. As this system directs the free arm of the brachiator in the direction of the next handhold at each tick, additionally simulating the motion of each handhold is sufficient for path generation to proceed as normal.

The author is creating a proof-of-concept application that takes advantage of Unreal Motion Graphics to facilitate the placing of the handholds and simulation of motion through a user interface.

An entirely different approach to this problem is to use reinforcement learning, as shown in [1]. Through this method, simulated brachiators may complete increasingly plausible traversals of motion paths.

7 CONCLUSION

The primary feature of this tool is to allow for quick iteration on designing brachiation paths for previsualization and layout. In addition, the capabilities of Unreal Engine provide various avenues for extension through the use of plugins such as Unreal Motion Graphics and Control Rig. Finally, although this problem has undergone many iterations in 2D space, this paper gives an approach to creating said movement in 3D space.

ACKNOWLEDGMENTS

Many thanks to my advisory committee - Tim McLaughlin (chair), Phil Galanter, and Dr. John Keyser - for helping me along the way and to Jason Spencer for generously providing computing hardware.

REFERENCES

[1] Glen Berseth. 2019. *Scalable deep reinforcement learning for physics-based motion control.* Ph. D. Dissertation. University of British Columbia. https://doi.org/10.14288/1.0378079

[2] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

[3] Laurent Favreau, Lionel Reveret, Christine Depraz, and Marie-Paule Cani. 2004. Animal Gaits from Video. *Graphical Models* 68 (08 2004). https://doi.org/10.1145/1028523.1028560

[4] Donald H. House and John C. Keyser. 2017. *Foundations of Physically Based Modeling and Animation.* CRC Press, Boca Raton, FL.

[5] David Ufford. 2009. Simulation of a 2-link Brachiating Robot with Open-Loop Controllers. (Jun 2009). http://hades.mech.northwestern.edu/images/e/e4/399UffordMonkeyBotSim-v2.pdf

[6] Zheng Zhang. 2006. *High Controllability and Automaticity for Physically-based Animation of Brachiation.* Ph. D. Dissertation. Nanyang Technological University. https://doi.org/10.32657/10356/35724

[7] Zheng Zhang and Kok Cheong Wong. 1999. Animating Brachiation. In *Eurographics 1999 - Short Presentations.* Eurographics Association. https://doi.org/10.2312/egs.19991065