Improved Character Animation Over Uneven Terrain

Charles E. Colvin, *MIT Candidate,* and Jani Kajala, *Supervisor, The Guildhall at SMU*

***Abstract*—Artist-rendered animation data struggles to establish believable character animations for immersive games without programmatic techniques. This study uses raw animation data and a series of blending techniques to explore the animation quality possible for a bipedal character on uneven terrain. An animation-blending tree organizes and optimizes animation changes and inverse kinematics tweaks animations to align properly with nearby geometry. Visual analysis in the test environment demonstrates the strength of developed research artifacts.**

***Index Terms*—Character Animation, Animation Blending, Inverse Kinematics**

I. Introduction

VIDEOGAMES often use cinematic gameplay to tell stories and immerse the player in environments, characters, and events. This illusion improves the success of narrative games; animation and graphical glitches can break immersion, removing the player from their experience and reminding them that they are playing a videogame. Artist animations achieve limited realism when the circumstance of their use is unknown, and programmatic techniques can produce results relevant to the current game state. Technologies used in real time game animation include skinned models that use bone weights, animation blending, and inverse kinematics (IK) to make character actions look natural for their current situation and environment. Smooth, intelligent character animations can improve the immersion of gameplay, facilitating dynamic traversal of environments. Developing reusable assets, whether source code or art assets, costs less than developing many similar assets. In addition, reducing the number of artist-created assets reduces the memory footprint of a videogame and frees up animators' time during development.

Setting up final character animations often involves trees of nodes called animation trees, which modify the characters' animated skeletons. In this study, raw biped animation data developed for flat terrain environments pass through an animation tree capable of intelligently improving the resulting skeletal transformations by visiting its nodes. The raw animations are a combination of artist work and motion capture, but local environment geometry helps determine final position and orientation of extremities. Inverse kinematics calculations use animation data and terrain geometry to modify foot position and leg orientation. Animation blending uses smooth weight values to transition animations. This study explores how well programmatic techniques can enhance animations with limited animation data. The expected result is that the character's foot position and leg orientation appear correct for the terrain geometry.

II. Research Review

A. Animation Blending Trees

Animation blending trees organize character animations in a logical way that facilitates blending. Reference [1] details several important concepts involved in smoothly animating a character model using blending trees. The game *MechWarrior* uses an animation blending tree that sources 152 animations per character to achieve their smooth results.

The procedure implemented in *MechWarrior* carries out the following steps:

1) Build an animation tree based on the in-game states of the character.
2) Advance necessary source animations to the next frame.
3) Determine the appropriate weights for each animation throughout the tree.
4) Visit nodes in the animation tree that blend the skeletal animation for rendering the frame.

Transitions from state to state use feather blending, a technique where the previous animation phases out and the new animation phases in over a short period. Axis blending is a technique that blends movement forward, backward, and strafing movements. Determining the direction and mapping the orientation of movement to a two-dimensional grid of animation sources is essential to the movement of characters. Distances from each animation's ideal point on the grid determine blend weights for directional source animations. Fig. 1 illustrates the feather blending and axis blending techniques as shown in the *MechWarrior* article.

Manuscript received April 26, 2013.

Charles Colvin is with Cohort 17 at the Guildhall at Southern Methodist University, Plano, TX 75024 USA (phone: 832-489-9460; e-mail: ccolvin@smu.edu).

Jani Kajala, Advisor and Supervisor, is with Software Development at the Guildhall at Southern Methodist University, Plano, TX 75024 USA (e-mail: jkajala@smu.edu).

Squirrel Eiserloh, Reader, is with Software Development at the Guildhall at Southern Methodist University, Plano, TX 75024 USA (e-mail: beiserloh@smu.edu).

Jeff Wofford, Reader, is with Software Development at the Guildhall at Southern Methodist University, Plano, TX 75024 USA (e-mail: woffordj@smu.edu).

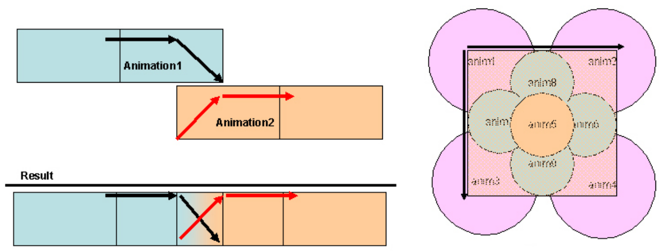

Fig. 1. Diagrams from [1] illustrate feather blending and axis blending techniques used in *MechWarrior*. Feather blending manages transitions between animation states, while axis blending covers the range of character movement with directional animations.

Computation optimizations are available by culling regions of the animation blending tree. Determining the necessary nodes and animations before visiting the tree nodes reduces CPU time spent on calculations and decreases the amount of animation data for each frame. Similarly, optimizations in animation data structures allow more animation assets to co-exist in memory, reducing number of read cycles during execution.

In a previous thesis work at the Guildhall [2], the theories used in animation blending breaks down in some situations. Michael Guerrero's solution describes the use of artist poses and animations, mapping those animations to gameplay states; at runtime, his solution dynamically maps states to weight values that determine how to blend artist animations. After he implemented this solution, his tests revealed accuracy problems that he addressed by developing additional animation assets for the problematic regions of gameplay states. The use of a similar technique for testing inadequate limb orientations reveals where additional work or new artist animation is necessary. When these new artist-made animations map to an axis blending node, they correct awkward positions in animation blends. As an alternative to developing additional resources (as in Guerrero's solution) using an IK technique to modify the skeletal structure at runtime could fix problems in animation blending without additional artwork or memory.

B. *Inverse Kinematics*

Simon Yeung's article [3] details relevant mathematics and techniques for foot placement using inverse kinematics. Yeung states that two pieces of data must be determined from outside the inverse kinematic equations. The first is a point on the ground for foot placement. The second is the leg orientation in the artist-made animation. A general solution attaches the foot to the ground, but there are infinite orientations of the leg that will place the foot at the same location. After placing the foot on the ground at an arbitrary angle, an algorithm determines the final, most correct orientation by minimizing the angle between the final orientation and artist animation. Yeung's simple two-joint IK technique, based on animation and environment constraints, makes up a significant part of the algorithm developed for the research artifact.

C. *Game Animation Technology Review*

Game techniques for developing smooth, high-fidelity animations go beyond animation blending trees and inverse kinematics, using divergent designs specifically created and optimized for their intended game experience. Animation blending transitions, interpolations, and layering techniques make up the backbone of animation blending systems, and motion capture and variations of IK are common.

Battlefield 3 developer Mikael Hogstrom states that temporal blend control and location alignment are important methods for improving animation believability [4]. The speed of transition between animations affects believability more than the quality of motion during a blend. Interpolating positions between gameplay and an in game cinematic creates a seamless transition between the two, as do state transitions when navigating an environment. Hogstrom calls this technique aligning. Transitions from free movement states, such as running, to animations that bind the character to an element of the environment, such as grabbing a ledge, require similar alignment techniques to smooth out the character locomotion. Achieving smooth animation without aligning between states would require highly accurate input from the player. Requiring such precise movement from the player increases his or her difficulty in traversing the environment and negatively transforms the player's experience. Tobias Dahl states that they no longer employ traditional animators at DICE, the studio that develops *Battlefield 3*; their animation system has become a combination of animation blending, state transitions, and motion capture data that calls upon traditional animation skills only to fix up animations before use [4].

Uncharted 2 uses a state graph, where each state has its own smaller animation blending tree [5]. Custom syntax defines animation behavior, allowing animators and designers to affect blend trees without direct programmer involvement. Transitions between states require information about blend time, curve type, and restrictions on transition timing during animations. The custom syntax defines these variables and the blend trees used by characters. A concrete data structure exposes a limited set of variables describing the character's state and environment for animation blending. Having this information clearly defined before the animation tree executes its nodes simplifies the blend process and design of the *Uncharted 2* pipeline. Three tiers of animation specificity shown in Fig. 2 allow the reuse of common structures. At the top layer, all characters transition between shared AI and

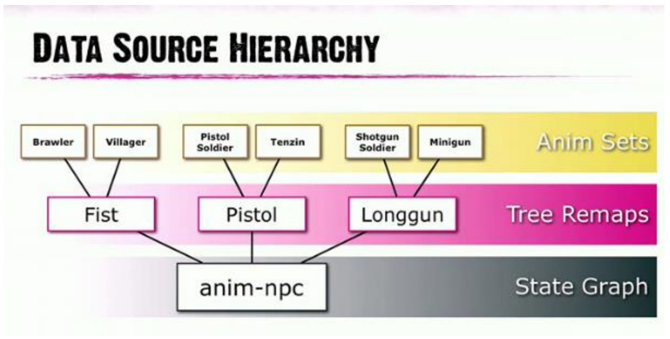

Fig. 2. From [5], this diagram shows the layers that allow reuse in *Uncharted 2*. State graph anim-npc controls a variety of animation tree classes, which each blend with different source animation sets.

gameplay behavior states. In the middle tier, classes of characters with similar behavior share animation blending trees. Finally, individual characters replace animation nodes in the blending tree with animations made specifically for their character model. Optimizations and reuse in *Uncharted 2* significantly improved the efficiency of memory use for animation data, leaving room for more animated characters and resulting in an overall richer cast. Additionally, the data-driven animation language and standardized state data structure allowed developers to prototype more quickly and see the results of their work.

Crysis 2 utilizes a parametric blending scheme instead of the standard transition, interpolation, and layering techniques that the term animation blending usually represents [6]. The system developed by Crytek uses a three-dimensional graph of animations that functions like axis blending, but locomotive data in the motion capture animations automatically generates animation locations in the graph. Parameters of movement speed, turning speed, and escalation act as coordinates for the graph, used to determine animation sources and blend weights. These parameters are each scalar values, together defining Cartesian coordinates for the graph. Convex shapes, defined by designers, fill spaces between animations and define which animations apply to the Cartesian coordinates within. The parametric Cartesian coordinates map to Barycentric coordinates to interpolate weights for animations. Extrapolated differences between animations allow for extreme situations that would otherwise require additional motion capture data. The system suffers from a conceptual design limitation, as pushing toward a fourth parameter (movement direction) resulted in a system that was difficult to debug and manage. However, the interpolation and extrapolation devised by Crytek requires only 34, leaving a small memory footprint and requiring little animation development. A four-dimensional system with three animations per axis, which would cover a decent set of game motions, would require 216 animations. These techniques optimize memory usage, allowing for increased overall visual fidelity, especially for current-generation consoles with limited available system memory.

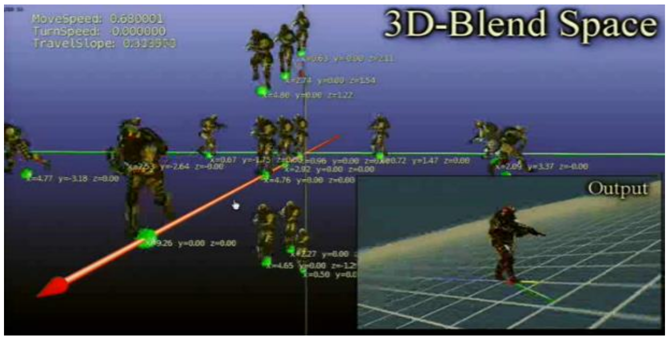

Fig. 3. From *Crysis 2* [6], this visualization shows how move speed, turn speed, and travel slope map to a three-dimensional axis. Using a single axis blend to represent a parametric state allows a small number of animations to represent a full range of locomotion.

Techniques for improving animations can drastically improve the animation fidelity of games, but there are several caveats that affect the design choices made by studios. In general, the value of time invested in advanced techniques depends entirely on the game design needs and technical restrictions placed on development by the engine and hardware. Consoles and lower-end machines possess limited memory resources and benefit from optimized animation systems that yield good results, such as the animation systems in *Crysis 2* and *Uncharted 2* [5], [6]. Top-down and first-person perspective views rely less on strong character animation systems because character models occupy less space on the screen. High-quality animation, high-polygon models, and high-resolution textures are all most crucial in the third-person perspective.

Games with simple environments and environmental interactions also simplify the animation technology required to make the character movements look clean and correct. Games with only flat surfaces to stand on have much simpler animation systems than those that allow players to climb over every surface variant. As with many aspects of a game experience, advancing quality beyond certain limits adversely affects the rest of the game. Memory and available processing power limit how much simulation and calculation can occur during a single frame, but in the case of character locomotion and animation, realism directly competes with smooth controls. For example, realistic animation for a jumping character would use a fixed trajectory. This control scheme feels restrictive and hurts players' experiences from a gameplay standpoint. From *Super Mario Bros.* to the modern *Tomb Raider* reboot, player input causes characters to move unrealistically in the air, forcing animation alterations to depict mid-air changes. Designers choose a balance between technical realism and creating a fun control scheme, and prioritization toward fun controls creates a better user experience. Simplification of animation systems and environment interaction reduces the negative effect of strong controls on animations.

For the purposes of the research artifact, navigation occurs on a simple environment with gently rolling hills. Expected game animation technologies required to navigate the environment smoothly include a single parameter axis blend based on current speed and an inverse kinematics algorithm for foot placement.

III. Methodology

A. Motion Capture Experimentation

Technology field review during early phases of artifact development includes motion capture involving iPi Soft software and two Microsoft *Kinect* devices. This solution costs less than motion capture suits and captures usable motion capture data; quality good enough for use in development of *Halo 4* cinematic movies [7]. To capture different perspectives, the *Kinect* devices face the same area of the room at a wide acute angle. Calibration of the two *Kinect* devices is moving a flat surface, such as a piece of cardboard, out in front of the devices. Then, before recording a motion, the actor stands in the standard model T pose: a standard standing pose for character models with arms outstretched. After recording calibration and motions, the software processes the recorded data into smooth animation frames that animators apply to models and edit through common 3D modeling and animation software. At game studios, animators would review motion capture data before use, fixing occlusion problems, glitches, and seams for looping animations. These skills and experience were unavailable for this research project. Despite being relatively inexpensive, the hardware and software costs exceed the budget for the project. Due to these barriers to using motion capture, purchased, pre-processed models and animation data appear in the final research artifact. However, experimentation reveals the iPi Soft pipeline produces useful motion capture, confirming its value for animators.

B. Asset Appropriation

Creating art assets for videogame projects requires skilled labor beyond what is practical for conducting this research, yet visual evaluation of the research artifact requires art assets designed for use in videogames. A bipedal character model and looping idle and walk animations are adequate assets to show character movement for the research artifact. A character model obtained online still requires skinning, a complicated and time-consuming task for an artist [8]. An online script at [9] uses guidelines provided by a user to assign skin weights for a character mesh. With some minor editing, two purchased animations become idle and walk loops in 3ds Max scenes for the skinned character.

A test environment that proved easy to obtain and implement is a grid of quads aligned at intervals of four, with elevation manipulated by a noise formula in the up direction. Aligned grid positions simplify queries for position. A smooth, rolling terrain simplifies character movement by limiting changes in elevation. Fig. 4 shows these assets together without IK.

C. Blending Tree Development

A series of nodes reference animations and state data to perform relevant skeletal adjustments and improve animation. After completing an operation, a node passes the skeleton up to influence other nodes in the blending tree. The core research artifacts are the animation blending tree and the operations performed on the character's animation set. An axis blending algorithm, inverse kinematics algorithm, player input, raw animation data, and terrain geometry data make up the animation nodes and their operations, as shown in the final animation blending tree, Fig. 5.

This blending tree is a referenced member of the model class. The walk and idle animation nodes, walk-idle blender node, and IK fix-up node all inherit from a base node class. The node's work function modifies the bone structure based on the node's desired operation, and then returns the skeleton to its parent node. Inheriting from a common base class allows for interchangeable and extendable functionality.

Setting external data, such as control stick weight and a reference to the terrain, allows the tree to perform its work during a single function call. At a high level, the tree calls its nodes in a child-first manner. In the research artifact, animation nodes advance their animation frame, walk the skeletal structure, and then return the results. The walk/idle transition, a linear blend node, uses linear interpolation to produce a combination of the two sourced animations by blending relative transformations. Finally, the IK fix-up node, a highly specialized node for placing feet of a biped character on terrain, alters hip placement and the orientation of thigh and calf bones.

D. Finite State Machine

The original design for proper foot placement utilizes a state machine that requires access to the model and animations, and expands the structure of temporary blending data. Complicated environment navigation benefits from a finite state machine, but simplifying and disconnecting the state machine from raw animation data makes the implementation cleaner and easier to maintain.

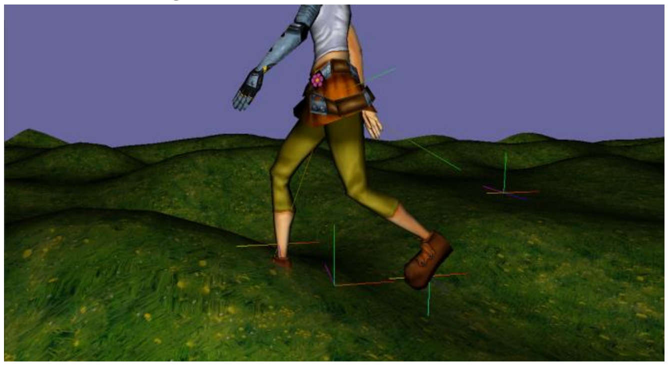

Fig. 4. Original walk animation, adjusted vertically for height of terrain but without any form of inverse kinematics. The right foot clips almost completely into the terrain surface and the left foot is floating far above the ground.

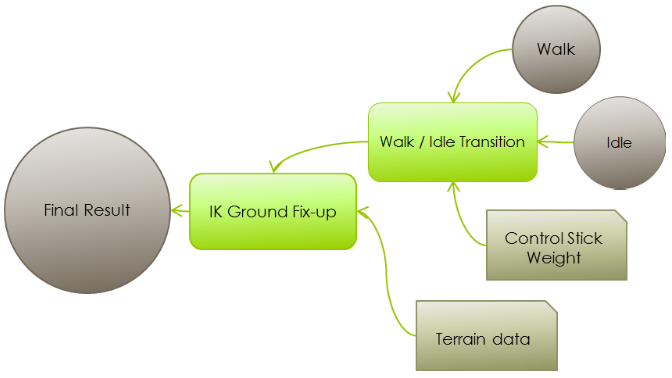

Fig. 5. Blend tree design for research artifact. Two animations seed a one-dimensional axis blending node based on the character speed. Terrain queries assist in establishing foot positions for the blended animation, which together are manipulated to produce the final result.

In this initial design, four states exist to describe the foot as flat, touching with the toe, touching with the heel, or lifted off the ground. Between states that touch the ground, the algorithm attempts to return the leg position smoothly to the base animation. Data analysis for this method includes analyzing the model to understand foot proportions and foot state. By calculating several frames in the future, the algorithm predicts when the base animation's feet touched the ground. A manually set integer constant determines the number of frames to predict, and a threshold distance above the ground defines when the foot touches the ground. When the foot touches the ground within a number of frames, a bi-cubic spline interpolates between the current position of the foot and the final, terrain-aligned position.

Several flaws exist in this convoluted method. Some animations do not need feet aligned with the ground, causing the prepared calculations to break down for fuller ranges of motion. Walking animation data requires additional time during load and scales badly with increased number of animations or number of animated characters. Predicting states requires calculating several frames of motion ahead, an expensive process to perform several times a second. Making smooth transitions to the target foot position requires additional tracking data, complicating the intermediate structure.

Initially, requirements stated foot locations and orientations must match the terrain underneath within a certain delta for a chosen point on the geometry, and follow believably first-order continuous movement, meaning both positions and velocity flow smoothly. Continuity tests that analyze specific position data for developed technologies could show such first-order behavior, but these tests assume that first-order continuity looks natural for human foot movement and the planting of feet. Instead of simulating natural movement, the alternate method for determining foot placement uses movement already defined in the animation data. This simpler process uses displacement of feet in the original animation, and allows the natural movement of feet over a surface to persist in an uneven environment.

IV. Results

A. *Blend Tree Implementation and Details*

In their base class, blend tree nodes feature a working frame used for developing intermediate results and a work function that returns the intermediate results. The working frame is composed of data structures that mimic the structure of the model and animations, and animation tree nodes assume that the bone structure of the model, animations, and working frames are identical in shape and name throughout. A node of the working frame contains pointers to child nodes, rotation, scale, and translation parts, a string containing its name for ease of reference, and a matrix for temporary calculations.

For the animation and linear blend nodes in the tree, performing the work task involves relatively simple operations that walk the bone structure in breadth-first fashion. Structural data created dynamically during the node's first use persists until the end of execution to improve memory performance.

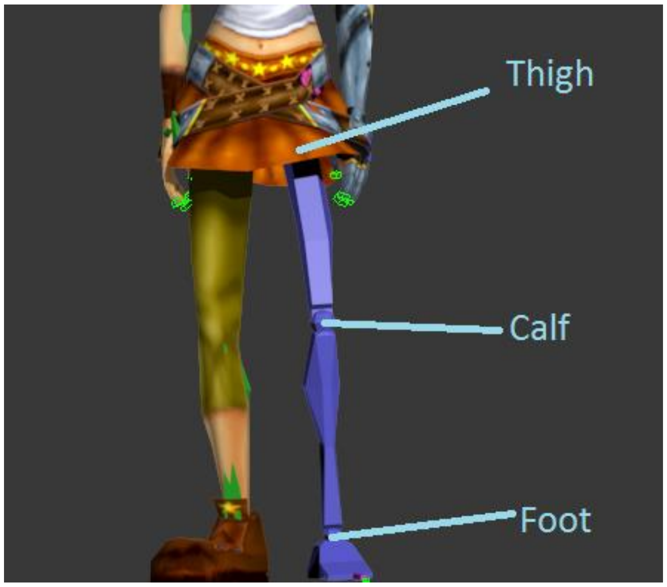

Fig. 6. Joints involved in inverse kinematics equations. Joint transformations in a character skeleton affect the starting points of the named body part. Maintaining character proportions limits joint modifications to the rotational part, as translation and scale stretch the character mesh.

The most complicated operations occur in the inverse kinematics node.

B. *Inverse Kinematic Foot Placement*

As previously stated, the initial solution involves detection of current state per foot, predicting the future state, and smoothly transitioning between target foot locations, followed by a general inverse kinematics algorithm. Because this method at some point requires using the animation to determine displacement between the foot and the ground, a simpler animation-based method removes the prediction and spline method completely. The final solution uses the original animation made for a flat environment to determine displacements in the up direction relative to the rolling terrain, ensuring that the movement remains believably human. Adding the terrain elevation to the desired foot displacement produces a reasonable target position.

With a problem involving only two limbs and so many known variables, calculating the relationship between the thigh and calf joints becomes a simple application of the law of cosines. Fig. 6 shows the bones involved in the IK fix-up. Knowing the lengths of the thigh, calf, and distance the leg needs to extend to the ground creates a triangle with known angles between the edges. The final solution IK fix-up solution uses the simple method for determining foot positions and triangle relationship for solving two joint IK chains.

An initial implementation fails to perform correctly in several regards due to use of simple application of calculated angles to canonical Euler angles. These initial calculations move foot locations near their final desired locations, but are visibly inadequate approximations. Symptoms include foot sliding and animation popping during forward leg movements. Because root node locations are determined by the terrain and original model displacement, one or both feet would often end up too short to reach the ground, and the impossible leg arrangements would produce bad transformations and disappearing geometry.

After attempting a naïve solution, carefully adopting and implementing an algorithm similar to that found in [3] yields far smoother results. The following algorithm adjusts leg orientations to uneven terrain in the research artifact:

1) Current horizontal axis-aligned locations of feet provide the basis for querying the terrain for elevation. In the case of the research artifact, the terrain is an axis-aligned grid that allows for indexed queries in to an array of quads. Barycentric coordinates calculated within the intended quadrilateral provide weights for interpolating points from the terrain mesh. Adding the height of the terrain to the displacement of the foot in the animation data creates a good target location for the foot.
2) If the full length of the leg (calf length + thigh length) cannot reach the new target positions, the algorithm will lower the character's root node. As Fig. 7 demonstrates, the resulting changes still look reasonable, but environments with sharper changes in altitude would cause abrupt changes in altitude using this method.
3) The law of cosines states a well-defined relationship between the edge lengths and the vertex angles of a triangle. Using this principle determines an angle for the inside of the thigh and calf, and the calf joint rotation is set using Euler angles. At this point, the leg from the thigh bone to the foot bone can be thought of as a single joint that will be rotated in to place from the hip. Fig. 8 shows the algorithm before rotating this single joint in to place. When developing a rotation to align the foot to the target foot location, if the starting and target vectors are close to opposite, the dot product returns an arbitrary axis and causes problems during the rest of the process. Setting the initial thigh orientation to 90 degrees prevents such an arbitrary rotation axis.
4) Vectors in Fig. 8 show current and desired orientations for the full leg limb. Using cross product between the two vectors determines the axis of rotation, and the geometric application of the dot product determines the angle of rotation for the thigh bone that places the foot in the correct final position.
5) As seen in Fig. 9, the leg is oriented in a strange direction, despite touching the ground correctly. One more rotation is required along an axis from the thigh to the foot. Solving the problem involves minimizing the difference between the calculated leg orientation and animated leg orientation. Rather than correcting with a vector travelling down the thigh as presented in [3], a perpendicular forward vector produces visibly correct results. The formulas in the research artifact come directly from [3], but reference [10] describes a general method in depth that solves a general system of several constraints, and may yield slightly better results by treating the orientation as two or three canonical vectors. Applying this rotation yields the final animation frame, as seen in Fig. 10.

The results reached are visibly successful for this environment and set of animations. Quick leg movements occur when the terrain descends drastically, suddenly extending the leg toward the ground. Larger declines would affect the root joint as well, but such an environment would require additional animations and finite states with different behavior.

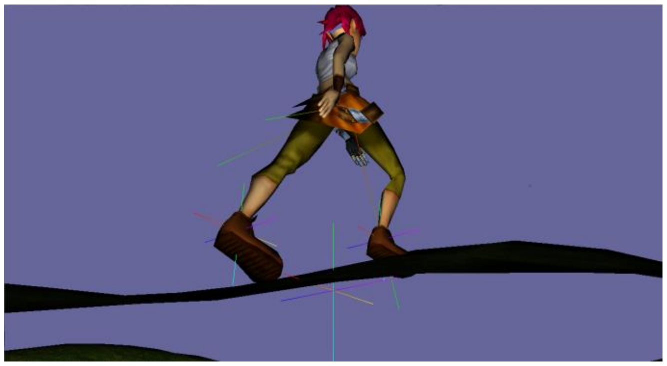
Fig. 7. In this arrangement, the character's back, or left, foot cannot reach the desired foot position without displacing the hip downward. The axis beneath the character is drawn in to the ground to indicate this change in the root bone. At this height, the left leg can just barely reach the desired foot position, causing the inverse kinematics to fully extend the leg.

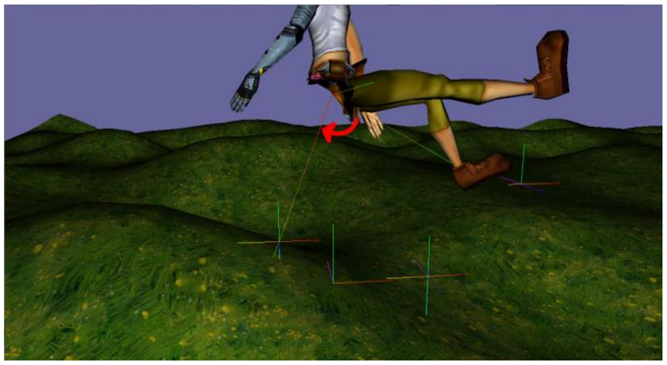
Fig. 8. Law of cosines defines a relationship between limb lengths and the angles between them. The algorithm sets the angle between the thigh and calf early in the process. Setting an initial thigh angle of 90° results in a useful axis of rotation, shown extruding toward the camera. The illustrated rotation brings the foot bone in to position.

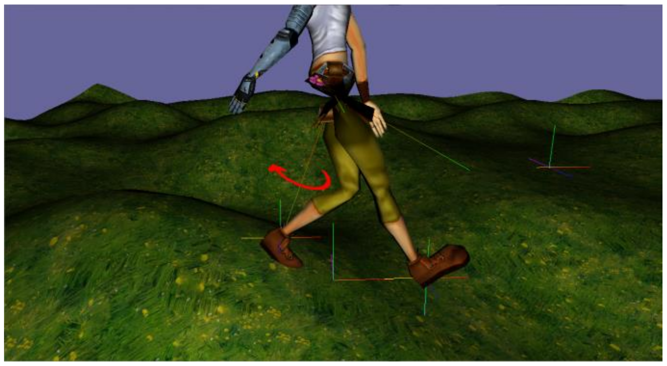
Fig. 9. Using formulas presented in [3], the final rotation uses the axis that preserves the foot location to minimize the angle between initial artist or motion capture data and the position calculated via inverse kinematics to reach a desired foot location. Minimizing using a forward vector for the bone rather than the vector aligning with the thigh produces consistently correct results.

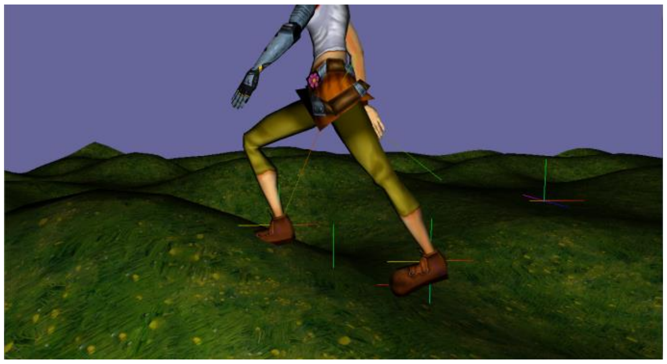

Fig. 10. Final animation, with inverse kinematics placement of feet, looks more realistic than the artist animation alone. Axes near the ankles show the better positions used to place the feet, and vectors extending from the thigh bone (visibly near the hip) show the initial and final positions for the leg, as well as the axis of rotation between the two.

V. Conclusion and Future Work

Through field review and development of the research artifact, this study shows improvement of animation fidelity through programmatic techniques; however, the needs and variations of techniques used to improve animation systems vary as widely as the game scenarios that need these animation systems. Animation blending trees and inverse kinematics, as explored by the research artifact, cover the common elements of more complicated animation needs for quality games and player immersion. These techniques – in tandem with situation-specific character states – are core techniques of programmatic enhancements for video game animation.

Complex environments and embedded data enable further research for character animation. Vertical navigation such as climbing, falling, and ledge grabbing would benefit from alternate applications of inverse kinematics and parametric blending. Environments with different kinds of terrain could apply animation blending to transition between states. The rolling hills environment used for research of animation techniques does not extend or scale to such needs and a better understanding of the geometry and methods for encoding gameplay data would enable richer environment navigation.

VI. References

[1] J. Edsall. (2003, July 4). Animation Blending: Achieving Inverse Kinematics and More. *Gamasutra* [Online]. Available: http://www.gamasutra.com/view/feature/131863/animation_blending_achieving_.php

[2] M. Guerrero, "Pseudo Inverse Kinematics via Skeletal Animation Blending," M.I.T. thesis, Software Development, Guildhall at Southern Methodist University, Plano, TX, Spring 2009.

[3] S. Yeung. (2012, January 20). Inverse Kinematics (two joints) for foot placement. *Gamasutra* [Online]. Available: http://www.gamasutra.com/view/news/129168/Inverse_Kinematics_two_joints_for_foot_placement.php

[4] T. Dahl and M. Hogstrom. (2012). Animation methodology for Battlefield 3. *GDC Vault* [Online]. Available: http://www.gdcvault.com/play/1015831/Animation-methodology-for-Battlefield

[5] J. Bellomy. (2011). Animating NPC's in UNCHARTED. *GDC Vault* [Online]. Available: http://www.gdcvault.com/play/1014738/Animating-NPC-s-in

[6] I. Herzeg. (2011). CRYSIS 2: Getting More Interactivity out of Animation-Data. *GDC Vault* [Online]. Available: http://www.gdcvault.com/play/1014527/CRYSIS-2-Getting-More-Interactivity

[7] V. Gray-Clark, R. Ecke. (March 11, 2013). Game On For iPi Soft and "Halo 4". *PRWeb* [Online]. Available: http://www.prweb.com/releases/2013/3/prweb10514154.htm

[8] *TurboSquid* [Online]. Available: http://www.turbosquid.com/FullPreview/Index.cfm/ID/451709

[9] Mixamo Auto-Rigger. *Mixamo* [Online]. Available: http://www.mixamo.com/c/auto-rigger

[10] L. Wang, C. Chen, "A Combined Optimization Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators," *IEEE Trans. Robotics and Automation*, vol. 7, no. 4, pp. 489-499, Aug. 1991.