Lightweight Procedural Animation with Believable Physical Interactions

Ian Horswill
Northwestern University, Departments of EECS and Radio/Television/Film
2133 Sheridan Road, Evanston IL 60208
ian@northwestern.edu

Abstract

I describe a procedural animation system that uses techniques from behavior-based robot control, combined with a minimalist physical simulation, to produce believable character motions in a dynamic world. Although less realistic than motion capture or full biomechanical simulation, the system produces compelling, responsive character behavior. It is also fast, supports believable physical interactions between characters such as hugging, and makes it easy to author new behaviors.

Overview[1]

Versatile procedural animation is a necessary component for applications such as interactive drama, in which characters participate in complex interactions that cannot be preplanned at authoring time. Although there have been procedural animation systems such as *Improv* (Perlin & Goldberg, 1996) for some time, and next-generation games are beginning to adopt commercial systems such as *Euphoria* (Natural Motion, 2006), these systems are still designed with scripted behavior in mind. General, extensible, toolkits for building autonomous characters are unavailable.

In this paper, I describe work in progress on *Twig*, a system for efficient procedural character animation that supports a simple dynamic simulation, including collision handling, pain detection, etc., and easy authoring of new behaviors. The system is built on the XNA platform (Microsoft, 2007) and is very efficient, running easily at 60Hz on a single core of a low-end machine. All code is open source.

The system consists of two main components, a simplified physics simulation, and a set of motion controllers for characters similar to the behavior-based control systems used in robotics (Arkin, 1998).

Although animation is ultimately driven by a low-fidelity dynamic simulation, character control is performed using a mixture of kinematic and dynamic control modes. Ballistic motions (reaching, stepping, head saccades) are performed by directly positioning end effectors (hands, feet, head) in Cartesian space, relying on constraint satisfaction in the dynamics system to act as an inverse kinematics system. Posture control is performed by applying simulated forces and torques to the torso and pelvis.

Interestingly, the use of a dynamic simulation actually simplifies control, allowing the use of relatively crude control signals, which are then smoothed by the passive dynamics of the character body and body-environment interaction; similar results have been found in both human and robot motor control (Williamson, 2003).

Twig shows that surprisingly simple techniques can generate believable[2] motions and interactions. Much of the focus of this paper will be on ways in which *Twig* is able to cheat to avoid doing complicated modeling or control, while still maintaining believability. This work is indebted to the work of Jakobsen (Jakobsen, 2001) and Perlin (Perlin, 1995, 2003; Perlin & Goldberg, 1996), both for their general approaches of using simple techniques to generate believable motion, and for specific techniques noted below.

Limitations

Twig is intended as a proof of concept. Its current repertoire of character behaviors is limited to navigation, reaching, grabbing, hugging, grappling, sitting/standing, gesturing, and withdrawal from pain. However, it demonstrates that its approach to simulation and control is effective for the class of applications for which it's designed. Further behaviors can be easily added. Similarly, the system could easily be extended to drive standard rigged character meshes even though it currently renders characters as collections of cylinders.

On the other hand, *Twig* is designed for versatility and believability (Bates, 1994) rather than physical realism. While it generates surprisingly compelling character motion, modifying it to be truly physical realistic require significant changes. A more accurate physics engine such as *Havok* (Havok, 2008) or *ODE* (Smith, 2006), or a more biologically-correct gait simulation (Hase, Miyashita, Ok, & Arakawa, 2003; Kuriyama, Kurihara, Irino, & Kaneko, 2002) may be more appropriate for works and genres requiring greater realism.

[2] In Bates' (1994) sense of "believable" as opposed to "realistic." A character is believable if an audience accepts it as if it were alive. It's realistic if it matches actual reality. Realism and believability in this sense are roughly orthogonal.

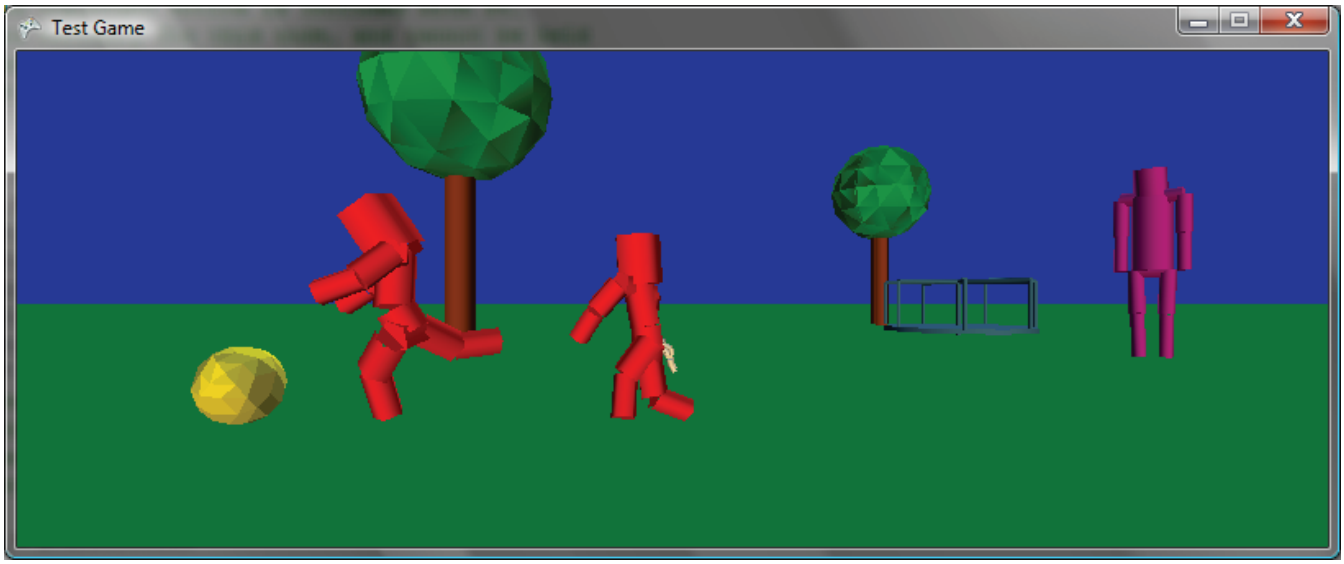

Figure 1: Screen shot from Twig. A parent watches over two children playing with a ball. The smaller child is carrying a ragdoll. The merry-go-round in the background is unoccupied, but functional (i.e. it spins when a character pushes it).

Geometric and kinematic modeling

Geometrically, *Twig* characters consist of a collection of cylindrical volumes. These are represented internally as kinematic chains of rigid links (the cylinders) connected at nodes (the joints). Link positions are determined by the positions of the nodes that form their endpoints. Object poses are thus determined entirely by the positions of the nodes that comprise them. In fact, nodes are the only containers of kinematic or dynamic state in the system. *Twig* uses a number of simulation techniques that come originally from the molecular simulation literature (Verlet, 1967), and its character models are effectively molecular models in which the nodes correspond to atoms and the links correspond to bonds.

In the schoolyard scene shown in Figure 1, the characters are modeled as 13 links (2 each for the spine and each arm and leg, and one each for the head, shoulders, and pelvis), connecting 16 nodes. The ball is represented as a single node. The merry-go-round, which is functional, is modeled as 18 nodes and 41 links; 25 of the links are visible and have collision volumes (the bars), and the rest are invisible links used only to hold the structure rigid.

Dynamics Simulation

Twig uses a mass-aggregate physics system (Millington, 2007) based on Jakobsen's work on the *Hitman* engine (IO Interactive, 2000; Jakobsen, 2001), in which objects are modeled as point masses (the nodes) connected by massless rods (the links), and motions are computed using Verlet integration (Verlet, 1967). In Verlet integration, the dynamic state of a particle is represented in terms of its position in the current and last frame, rather than its position and velocity. Given a fixed inter-frame interval Δt, we can describe the position $\mathbf{p}$ of a node at time $t+\Delta t$ in terms of its position in the previous frames as:

$$\begin{aligned}\mathbf{p}(t+\Delta t) &= \mathbf{p}(t) + (\mathbf{v}(t) + \mathbf{a}(t)\Delta t)\Delta t \\ &\cong \mathbf{p}(t) + \frac{\mathbf{p}(t) - \mathbf{p}(t-\Delta t)}{\Delta t}\Delta t + \mathbf{a}(t)\Delta t\Delta t \\ &= \mathbf{p}(t) + \big(\mathbf{p}(t) - \mathbf{p}(t-\Delta t)\big) + \mathbf{a}(t)\Delta t^2 \\ &= 2\mathbf{p}(t) - \mathbf{p}(t-\Delta t) + \mathbf{a}(t)\Delta t^2\end{aligned}$$

where $\mathbf{v}(t)$ is the node's instantaneous velocity and $\mathbf{a}(t)$ its acceleration at time t. If we want to model viscous damping, this can be done by modifying the relative weighting of the position in the two frames:

$$\mathbf{p}(t+\Delta t) = (2-d)\mathbf{p}(t) - (1-d)\mathbf{p}(t-\Delta t) + \mathbf{a}(t)\Delta t^2$$

where d is the damping factor.

This scheme has a number of advantages. First, the complete kinematic and dynamic state of an object is contained in positions of its nodes, together with their stored positions from the previous frame (links function only as collision volumes and distance constraints on node position). The lack of explicit representations of momentum, angular momentum, or even orientation, significantly simplifies the dynamics calculations. Second, it makes constraint satisfaction much easier, since node positions can be directly modified to enforce constraints, without having to compute their effects on orientation, angular momentum, etc. Finally, it allows the behavior system to control the characters and their nodes entirely in Cartesian space, without having to deal with joint angles or nested coordinate frames. The cost of the design is that in the few cases where momentum, orientation, or joint angles are needed to make control decisions, these need to be computed from

position data. The AGEIA PhysX engine (Müller, Heidelberger, Hennix, & Ratcliff, 2007) also uses a "position-based" approach to build a much more general dynamics engine. However, the simpler system discussed here is sufficient for our purposes.

Friction, drag and damping

The current system does not support accurate models of friction or drag. Instead, it provides a damping term (see equation above), whose coefficient is large when a node is in contact with a supporting surface, and smaller when in the air. While this is technically inaccurate (damping is linear in velocity, whereas drag is quadratic and friction includes a step function), the inaccuracies generally aren't apparent to a viewer.

Nodes can be damped relative to the environment frame (modeling air friction) and/or relative to another node in the object (a crude model of the biomechanical damping of muscles and tendons). Nodes can also be locked in place to model large static friction forces.

Kinematic constraints

Kinematic constraints (joint limits, rigid distance constraints, etc.) are implemented by projection, i.e. by moving a node that violates a constraint to a nearby position that doesn't violate it. On each update cycle, each object tests its nodes against its constraints and adjusts the positions of nodes to locally satisfy the constraint being evaluated. This has the potential to violate some other constraint that had just been satisfied, but such violations are generally not detectable by the user, especially if the object is moving. Moreover, if the object stops moving, it quickly relaxes into a configuration that satisfies the constraints.

To locally enforce the distance constraints imposed by a link, we measure the actual distance $\|\mathbf{p}_i(t) - \mathbf{p}_j(t)\|$ between its endpoint nodes and compare it to the desired distance, d. If the nodes are not the desired distance apart, we move each node half the difference between the desired and actual distances: [3]

$$\mathbf{p}_i(t) = \mathbf{p}_i(t) - \frac{\|\mathbf{o}\| - d}{2\|\mathbf{o}\|}\mathbf{o}$$
$$\mathbf{p}_j(t) = \mathbf{p}_j(t) + \frac{\|\mathbf{o}\| - d}{2\|\mathbf{o}\|}\mathbf{o}$$

where $\mathbf{o} = \mathbf{p}_i(t) - \mathbf{p}_j(t)$ is the offset between the nodes.

Projection is computationally efficient, but not especially accurate since it does not necessarily conserve energy or in all cases, even momentum. However, in practice, it generates motions that look real enough. Again, the goal is believability, not numerical accuracy.

[3] By moving the nodes in equal and opposite directions, we avoid adding momentum to the system.

Collision handling

Collisions are handled as a special case of constraint satisfaction. After each dynamic object is updated, its collision volumes are tested against the collision volumes of other objects and their nodes moved so as to separate their collision volumes.

We will discuss the link/link collision case, since most collision volumes in *Twig* are attached to links. Other cases can be handled similarly. Let the endpoints of one link be nodes *i* and *j*, and the endpoints of the other be nodes *k* and *l*, with positions $\mathbf{p}_i$, $\mathbf{p}_j$, etc. Since links are modeled as cylindrical collision volumes, this can be reduced to testing the distance between the line segments $\overline{\mathbf{p}_i\mathbf{p}_j}$ and $\overline{\mathbf{p}_k\mathbf{p}_l}$. If the distance between them is less than the sum of the radii of the two cylinders, then they are interpenetrating and need to be separated. To be physically accurate, we should determine the precise points of contact on the two cylinders, compute the relevant torques and moments of inertia, and update the positions of the endpoints accordingly. However, in practice, the links are almost always chained with other links that constrain their allowable motion. Since these inter-link constraints dominate the dynamics of the collision, we can obtain realistic looking collisions by translating the colliding cylinders apart, ignoring torques, and allowing the inter-link constraints to produce a realistic-looking motion.

In particular, let $\mathbf{n} = \frac{(\mathbf{p}_i - \mathbf{p}_j)\times(\mathbf{p}_k - \mathbf{p}_l)}{\|(\mathbf{p}_i - \mathbf{p}_j)\times(\mathbf{p}_k - \mathbf{p}_l)\|}$ be the contact normal along which the cylinders intersect. The distance between the spines of the cylinders is then $\mathbf{r} = (\mathbf{p}_i - \mathbf{p}_k) \cdot \mathbf{n}$. If the radii of the two cylinders are *a* and *b*, then the penetration depth of the cylinders is $d = a + b - \|\mathbf{r}\|$. We then translate both nodes *i* and *j* by $^{d}/_{2}\,\mathbf{n}$ (half the penetration), and we translate both nodes *k* and *l* by $-^{d}/_{2}\,\mathbf{n}$.

A collision impulse could also be added to the links to simulate elastic collision. However, since humans don't bounce well, this would be counter-productive for links representing body parts.

Tactile sensing

When the system detects link/link collisions, it stores in each link pointers to the link that hit it, and the object to which that link belongs. The system also computes the kinetic energy of the impact. If the kinetic energy is over threshold, the system registers it as pain. Characters also maintain an overall pain level, which decays exponentially over time.

Low-level character control

All character behavior is ultimately implemented by moving nodes around. One of the advantages of the style of kinematic and dynamic modeling in *Twig* is that this control can be done directly in Cartesian coordinates, without

having to deal with joint angles or performing explicit inverse kinematics and dynamics.

Node control

Nodes are controlled principally by setting their velocity or acceleration or in some cases by directing them to perform a ballistic motion to a set-point. In the latter case, the node automatically moves along a straight line to arrive at the target in a specified amount of time without further need for control. This mode is used principally for limb motions.

Nodes can also be locked in position or told to lock themselves when they come into contact with the ground plane.

Posture Control

Posture is controlled by applying forces directly the nodes of the torso and pelvis, rather than by balancing the body as an inverted pendulum using simulated muscular forces. This makes control simple and stable at the cost of sometimes violating the laws of physics (for example, the current version of the system applies postural forces even when the legs aren't touching the ground). Again, this is adequate for the tasks we're considering, but a more complicated scheme would be necessary for applications in which it was necessary to accurately model balance, tripping, falling, etc.

Posture control consists of a set of simple control loops:

- Standing consists of two control loops
  - A force is applied along the Y (up) axis to the center of the pelvis to hold it at standing height.
  - Forces are applied along the X and Z axes to horizontally align the center node of the pelvis with the midpoint of the feet.
- Sitting up works essentially like standing, except that the center node of the shoulders is controlled so as to place the character's center of mass directly over the midpoint of the two feet. The shoulders are also tilted slightly in the direction of motion when the character is running.
- Orientations are controlled by twisting the pelvis and shoulders. Since the dynamics engine doesn't explicitly support torques, the torque is produced by applying opposite forces to opposite sides of the character.
  - The pelvis rotates to align with the direction of walking
  - The shoulders rotate to align with the gaze direction, subject to the constraint that they not rotate more than 90 degrees relative to the pelvis.

Note that these control loops are simple proportional controllers rather than proportional-derivative controllers (i.e. they have no damping term). They rely on the damping of the nodes themselves to prevent oscillation.

Limb control

The head controller points the "front" of the face toward the current gaze target, or the direction of motion, if there is no gaze target. In the current version of the system, this is an instantaneous motion. This will undoubtedly need to be changed to a smooth motion in the future, but since the current models have no faces, this kind of exaggerated motion is actually useful for cuing the viewer that the character's gaze is shifting.

The arm controller currently supports five actions: swing (used when walking), reach, grapple, hug, and grab. Swinging is implemented by applying impulses to an arm when the opposite foot begins a step. At the level of the limb controller, reaching, grappling and hugging are all implemented by moving the hands directly in front of the shoulders at near-maximum extension. The rest of the reach, hug, and grapple actions are then controlled by higher-level controllers. Grabbing is implemented by creating an invisible, zero-length link from the hand of the designated arm to a designated node on the target object, thus attaching the object to the end of the arm, while allowing it to swing freely as if attached by a ball joint.

Legs are controlled principally by the gait controller (see below).

The system also supports simulated respiration by moving the shoulders up and down in a sinusoid, similar to (Perlin, 1995). Respiration increases with increased walking speed. In the current system, respiration is largely invisible to the viewer because the shoulders are modeled as a single cylinder, however they could be split to make it more apparent.

Hugging. Hugging is implemented by reaching and approaching the target, then joining the hands when the target object makes contact with the character's torso.

"Grappling" Grappling is a kluge that is implemented by waiting until the character closes to within less than an arm length of the target and then engaging reaching, causing the arms to bash into the other character, looking like shoving, punching, or wrestling to the viewer. It also tends to cause pain in the other character, triggering its pain withdrawal reflex, thus making it step back. While insufficient for a fighting game, it's sufficiently realistic for depictions of children fighting.

Gait Control

The gait generator drives the character to walk with a direction and speed chosen by one of the higher-level behaviors. Gait generation is largely kinematic and is closest to the work of Perlin (Perlin, 2003). The gait generator sets the ground-plane velocity of the pelvis to the walk vector, then monitors the extension of the legs. When a leg is sufficiently far behind the pelvis, the gait generator moves the foot node on a ballistic trajectory to a point in front of the pelvis, but in the direction of the walk vector. The constraint handling system moves the knee appropriately and insures that it doesn't bend backward or sideways.

High-level Control

The low level controllers are driven by a set of competing higher-level behaviors, currently sit-down, stand-up, freeze (holds the character immobile), pain withdrawal (a fast, open-loop retreat from a source of pain), and approach (steers the character to a designated object). These behaviors each compute an activation level (a measure of the utility of firing the behavior) and a motor vector (values for the control signals to send to the lower levels). On each update cycle, the highest-activation behavior is chosen and its outputs are routed to the gait generator and arm controller.

Freeze and pain withdrawal are simple behaviors. Approach requires a target from a higher-level behavior. Thus the control system is a hierarchical behavior selection system similar to Blumberg's work on ethologically inspired control (Blumberg, 1996).

Object approach

The approach behavior takes as input a target object, a distance d from the object to stop at, and a direction $\mathbf{d}$ from which to approach it. Approach also takes as input settings for the hug, reach, and grapple controls, which it forwards to the arm control behaviors. It generates a walk vector, $\mathbf{w}$ (a velocity vector for the gait controller), based on an artificial potential field[4]/motor schema (Arkin, 1998) which is the sum of an attraction component, $\mathbf{a}$, in the direction of the target object, and a repulsion component, $\mathbf{r}$, pushing away from any intervening obstacles:

$\mathbf{w} = \mathbf{a} + \mathbf{r}$, where:

$$\mathbf{a} = \mathbf{p}_{\text{target}} - \mathbf{p}_{\text{character}} + d\mathbf{d}$$

$$\mathbf{r} = \sum_{o \neq \text{target}} c \frac{\mathbf{p}_{\text{character}} - \mathbf{p}_o + (\mathbf{p}_{\text{character}} - \mathbf{p}_o) \times \mathbf{e}_Y}{\max(d_{min}, \|\mathbf{p}_{\text{character}} - \mathbf{p}_o\|^2)}$$

Again, $\mathbf{p}_x$ here denotes the position of object x, whereas c and d_{min} are constants tuned to taste, and $\mathbf{e}_Y$ is a unit vector pointing upward. The cross product term produces a curl component to the field that pushes the character around obstacles, making it less likely that they will encounter local minima in the field. To avoid asking the walk system to move too fast, the $\mathbf{a}$ component is also saturated to prevent its magnitude from going over a threshold. To reduce computation time, objects over a threshold distance are ignored when computing $\mathbf{r}$.

Example application

Twig can be run as a server, in which it accepts discrete behavior commands from another system, such as a drama manager. In this case, *Twig* provides a straightforward asynchronous RPC interface. However, *Twig* characters can also be made fully autonomous by adding further behaviors to the hierarchy. One such example is the scene shown in figure 1. Here, a child makes excursions from its parent to explore the environment, and in particular, to play with a ball, but periodically returns to the caregiver to be soothed. This is known as the "safe home base" phenomenon in Attachment Theory (Bowlby, 1969).

The demonstration involves three high-level behaviors: playing with the ball, fighting, and running to hug the parent (attachment). The characters have attention and short-term memory systems that appraise each object in view or in the STM for its salience (interest level), valence (good/bad), and monitoring priority (how much to pay attention to it). The maximal salience object becomes the focus of attention for that update cycle. The behaviors can then react to the focus of attention and change their activation level accordingly.

In parallel, the gaze control system shifts visual attention between the current focus of attention, the target of the approach system (if different), and other objects that have high monitoring priority (the parent and any potential threats).

The result is that the children run after the ball because it's highly valenced. As the small child gets farther from the parent, however, it becomes anxious and the monitoring priority of the parent increases, causing the child to periodically stop and look back to the parent. Eventually, the child's anxiety becomes sufficient for it to abandon the ball and return to hug the parent, which reduces the child's anxiety. Eventually, the child's attention returns to the ball, the child returns to play, and the cycle repeats.

Implementation and performance

Twig is written in C# and runs under XNA 2.0 (Microsoft, 2007). The Verlet integrator runs at a fixed update rate of 60Hz, but the renderer can skip frames if it gets behind. Fortunately, this isn't an issue in practice. A debug build of the scene in figure 1 takes approximately 5ms per frame on a single core of a 1.6MHz notebook machine.[5] Physics and behavior generally take 1-1.5ms range when the characters are interacting and less than 0.5ms when the characters are widely spaced, allowing broad-phase collision detection to prune all tests. Actual rendering is slower, generally around 3.8ms, because the current system isn't optimized to batch drawing operations. This could be improved considerably using hardware instancing.

Failure modes

The simplified physics and control in *Twig* do cause occasional problems. For example, the walking system applies an external force directly to a character's torso, which then pulls the (largely passive) legs along, rather than by simulating muscular forces within the legs and torso. This can

[4] Note that despite the name, the artificial potential fields used in navigation are generally not the gradients of any well-defined potential function.

[5] Timings were done on a ThinkPad X61t, with a 1.6GHz Santa Rosa chipset, 3GB of RAM, and the Mobile Intel 965 Express graphics chipset.

potentially allow a character to violate conservation by pushing the merry-go-round while standing on it.

The system's kinematic simplifications are also sometimes noticeable. Characters are controlled by applying forces in Cartesian coordinates to joints, rather than through manipulation of explicit joint angles. While this makes control much simpler, it also means that a separate set of constraints are necessary to insure that joints don't move into kinematically impossible configurations, such as bending backward. While this is straightforward to do for the knees, it's harder to do for the elbows because of the wider range of motion at the shoulder than the hip. In the current system, the elbows sometimes seem to wiggle unrealistically because, even though they are moving through configurations that are physically possible, they aren't capturing the true dynamics of a human arm.

A final class of issues stems from conflicts within the behavior system itself. For example, if the child runs too fast when trying to hug the parent, it can impact the parent with enough force to cause pain. That triggers a pain withdrawal reflex during the docking phase of hugging. Although this behavior is realistic in the sense that real human children do it from time to time, it has the potential to turn a sentimental scene into slapstick.

Conclusion

Twig is a simple, extensible procedural animation system intended for interactive narrative applications. Although still under development, it provides a proof-of-concept demonstration that the combination of a simplified physics simulation, together with a set of simple control loops, can provide satisfying and believable character movement and interaction. In particular, it shows that while dynamics can help improve the believability of procedural animation, a surprisingly simple dynamics simulation is sufficient, at least for styles and genres that do not require physical realism.

Acknowledgements

I would like to thank Michael Mateas, Andrew Ortony, Magy Seif El-Nasr, and Andrew Stern, for suggestions, and encouragement. I would also like to thank the reviewers for feedback, and particularly for catching a typo in eq1.

References

Arkin, R. (1998). *Behavior-Based Robotics*. Cambridge: MIT Press.

Bates, J. (1994). The Role of Emotion in Believable Agents. *Communications of the ACM*, *37*(7), 122-125.

Blumberg, B. (1996). *Old Tricks, New Dogs: Ethology and Interactive Creatures.* Massachusetts Institute of Technology, Cambridge.

Bowlby, J. (1969). *Attachment and Loss*. New York,: Basic Books.

Hase, K., Miyashita, K., Ok, S., & Arakawa, Y. (2003, May 2003). Human gait simulation with a neuromusculoskeletal model and evolutionary computation. *The Journal of Visualization and Computer Animation*, *14*(2), 73-92.

Havok. (2008). The Havok 5.5 Physics Engine: Havok Inc.

IO Interactive. (2000). Hitman: Codename 47: Eidos Interactive.

Jakobsen, T. (2001). *Advanced Character Physics*. San Jose: CMP Inc.

Kuriyama, S., Kurihara, Y., Irino, Y., & Kaneko, T. (2002). Physiological gait controls with a neural pattern generator. *The Journal of Visualization and Computer Animation*, *13*(2), 107-119.

Microsoft. (2007). XNA Game Studio 2.0: Microsoft Corporation.

Millington, I. (2007). *Game physics engine development*. The Morgan Kaufmann series in interactive 3D technology. Amsterdam ; Boston: Morgan Kaufmann Publishers.

Müller, M., Heidelberger, B., Hennix, M., & Ratcliff, J. (2007). Position based dynamics. *J. Vis. Comun. Image Represent.*, *18*(2), 109-118.

Natural Motion. (2006). Euphoria:core motion synthesis library: Natural Motion, Inc.

Perlin, K. (1995). Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, *1*(1), 5-15.

Perlin, K. (2003). *Unpublished work on bipedal walking*.

Perlin, K., & Goldberg, A. (1996). Improv: A System for Scripting Interactive Actors in Virtual Worlds. *Computer Graphics*, *30*, 205-216. citeseer.ist.psu.edu/perlin96improv.html

Smith, R. (2006). *Open Dynamics Engine v.0.5 User Guide.*

Verlet, L. (1967, July 1967). Computer 'Experiments' on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, *159*(1), 98-103.

Williamson, M. M. (2003). Oscillators and Crank Turning: Exploiting Natural Dynamics with a Humanoid Robot Arm. *Philosophical Transactions of the Royal Society: Mathematical, Physical and Engineering Sciences*, *361*(1811), 2207-2223.