Reactive Motion for an Animated Boxer

Kevin Forbes, Alexander Kolliopoulos

May 3, 2004

1 Introduction

In a round of boxing, a fighter will make decisions and carry out actions to give himself an advantage, by blocking or evading punches while using well placed attacks to defeat an opponent in the ring. A boxer must constantly decide what action will give him the best chance of winning based on the perceived state of the opponent and the boxer's own physical state. These decisions must occur with little time for thought or analysis, using instinct and technique from training. The goal of this project is to capture an approximation to the reactive motion required of a boxer in a ring in real time.

1.1 Previous Work

There has been significant interest in developing behavioural systems for controlling synthetic characters. Tu and Terzopoulos's artificial fishes used a deterministic system built around an intention generator for making decisions [8]. In the same year, Karl Sims presented his evolutionary approach to creating and controlling characters, but they are limited to finding body configurations and control systems to tackle only very simple tasks [7]. In work addressing problem solving at what may be considered a cognitive level, situation calculus has been applied to automatically animate a scene of a merman escaping a shark by using obstacles in the environment to his advantage. Such behaviour is much more human in character, but it is beyond what should be necessary for animating the reactive motion required in a boxing ring.[3].

Beyond decision making, a reactive boxer must be capable of motion in a variety of styles while appearing reasonably realistic. The problem of controlling human-like characters has been addressed in various ways. For example, [2] proposes composing self-rating controllers to manipulate a character within the constraints of a physically simulated system. This requires non-trivial computation to run the simulation, even using shortcuts such as a neural network technique for solving the physics [4]; and good controllers are notoriously difficult to design, even for simple tasks such as locomotion. Approaches using motion capture show more promise for purposes of real time motion. Motion graphs are particularly well suited to producing visually pleasing motion from motion capture data, but given the variety of motions involved in boxing, in addition to making several styles available for each motion, it would be difficult to produce a useful, highly connected motion graph with a reasonable amount of motion capture [5]. Zordan has done work combining motion capture data with physical simulation and inverse kinematics in order to simulate athletics such as boxing and dancing [9]. While these results are impressive, a purely kinematic approach was chosen for this project due to time constraints.

2 System Architecture

The system can be divided into two major components: the reactive behavioural system, and the underlying animation system. These two components were designed to be independent, and although time constraints led to some breaches in modularity, they could be easily separated. Keeping the behavioural code separate from the graphical simulation code affords us a great deal of future adaptability.

2.1 The Animation System

At the lowest level, the animation system transforms a hierarchy of joint matrices, which are used to deform a polygonal mesh using linear blend skinning. These meshes are textured and rendered using OpenGL. By varying the the transformations applied to the joints over time, animation can be achieved.

The most important object type in the animation system is the *actor*. An actor represents one renderable, animate being in the simulation. At the code level, it associates a joint hierarchy and mesh with a material type and a repertoire of prerecorded animation sequences. The association between actors and these resources is many-to-one, which allows for the efficient sharing of resources when multiple actors are simulated.

2.1.1 Data Acquisition and Processing

All of the animations currently used by the system were captured using a Vicon Motion Systems optical motion capture system. A script of actions was prepared which included all of the moves required by the behavioural system. These were performed several times each, in four long takes. The raw capture data was processed using the ViconIQ software package, and fit to a 19 joint skeletal human model. The resulting joint transformation sequences were exported to Maya. [1] Once in Maya, the joint transforms were converted from a Z-up to a Y-up coordinate system, and scaled so that the unit distance was equal to one meter. The animation curves were simplified to reduce the memory footprint of the final result. The processed animation curves were used to construct a Maya skeleton, which was bound to a polygonal model of a boxer. No proper motion retargeting was needed, since the model was constructed to have similar proportions to the the mocap actor. Finally, the animation was manually segmented into individual movements, and the animation curves, skeleton, and mesh were exported using a custom Maya plugin.

2.1.2 Playing Animations

Individual animation curve files are loaded into KeyFrameAnim objects. The skeleton is assumed to be a rigid articulated figure, so only rotation data is loaded for non-root joints. These keyframes are stored as time indexed sequences of quaternions. The root joint is more complicated, since it describes the skeleton's global position and orientation. The simulated boxers' position has 3 degrees of freedom: translation in X and Z, and rotation about the Y axis. The root joint's transformations in these degrees of freedom are modified at load-time to be relative to an initial, identity transformation. This allows for multiple animations to be played in sequence, with the ending root position of each becoming its successor's beginning position. The root joint's remaining degrees of freedom are loaded without any extra processing.

Given a time in milliseconds, a KeyFrameAnim object can be queried for a whole-skeleton pose. Quaternion slerp is used to interpolate between keyframes. When an actor is made to begin playback of an animation, its current pose is stored. Linear interpolation is used to smooth the transition from the actor's initial pose to the animation's keyframed poses. The interval of this interpolation can be changed to favour either higher fidelity to the motion capture data, or smoother transitions.

[1] A web page describing the technical details of the capturing process is at `http://www.dgp.toronto.edu/~alexk/fullmocap.html`

2.1.3 Action Interface

While an actor can be controlled by specifying individual animations for it to play, it is desirable to be able to control it at a higher level. For example, in our database, there are 12 different animations for taking a step forward. While each of these animations perform the same action, they reflect different styles of motion, such as "skilled" or "tired" to various degrees. There are also duplicate motions in the database, which are semantically and stylistically identical, but were culled from different takes. The actor's controller would have to know the names and attributes of all of these similar and redundant animations in order to pick one to carry out its intended actions.

The actor's Repertoire object allows for high-level control. It loads an XML file that contains annotations for each animation file in the database. Each animation file is associated with an action string, and is given any number of named, weighted attributes. For each action type, the repertoire creates an Actionset object. An Actionset object constructs a matrix of the weights, indexed by attribute and animation. Thus each row in the matrix gives a quantitative assessment of an animation with respect to all of the attributes used by animations that perform the same action. Attributes that have no value given in the XML file are assumed to be zero.

As the simulation runs, the controller feeds attributes and values to the actor. These values represent the actor's internal state and can be reset as the actor's state changes. For example, the controller might set the "tired" attribute to zero at the start of the simulation, and then set it to increasingly higher values as the time passes and the actor performs strenuous actions.

At any time, the controller can request that the actor perform an action. The actor checks its Repertoire for an Actionset that matches the requested action string. If no match is found, the actor does nothing. If a match is found, the actor constructs a vector from its current attribute state. This vector is multiplied by the Actionset's matrix, and the resulting vector is taken to be a weighting of the various animations' suitability. Each element of this vector is perturbed by a small random component to break ties. The actor then starts playing the highest-weighted animation.

This scheme allows the controller to direct an actor with an arbitrarily large repertoire using a small, fixed vocabulary of action and attribute names. Additional animations can be added to the actor's repertoire by simply changing the XML file. The quality of actor's performance can be enhanced by filling in the gaps in the permutations of the attributes. For example, if a tired, amateur boxer throws a jab in the current system, the jab animation played will be either tired or amateur, depending upon the boxer's attributes' relative weights. It would be possible, however, to capture more motion data to provide a specialized animation to play in such a situation.

2.2 The Behavioural System

The boxers' mental model is based on that of Tu and Terzopoulos's artificial fishes. This model has been shown to be adequate for reacting to threats immediately and in real time, while carrying out the goals of the creature. Where an artificial fish is interested in avoiding predators, eating, or mating, a boxer is instead concerned with avoiding incoming punches, attacking the opponent, and conserving his energy. When a decision needs to be made, an *intention generator* is used to select an intention given the boxer's perceived state of the world and internal state. This intention is passed to the *behavioural routines*, which further refine the intention to a specific action to be carried out by the boxer. This action is passed to the animation system which acts as the motor controller for the boxer by playing back an appropriate motion captured animation.

2.2.1 Boxer State and Perception

The boxer's internal state determines the style of motion that is used as well as modifying his behaviour. The main internal state parameters, which may be loaded from a boxer profile XML file, determine a boxer's skill, style, and strength. Each of these parameters is tied to styles of motion that a boxer is capable of, with skill selecting skilled or amateurish motion and style selecting defencive or offencive motion. Whether a boxer uses energetic or tired motion is controlled by the boxer's internal energy parameter. Unlike the other parameters, the energy is dynamic and changes as a fight progresses. A boxer's energy will be lowered as he moves, especially by throwing punches, but energy may be recovered by taking a moment to let his head clear. Additionally, a boxer receives an artificial boost to energy when a punch lands to correspond with the motivation that a well placed hit can bring a fighter. Very often one will see a tired boxer launch into a rush of attacks after a critical blow to the opponent.

A boxer has some perception of the state of the world around him. As long as the opponent is within about 60 degrees of the boxer's gaze, he is considered to be visible, and the boxer knows the relative angle to the opponent. When the opponent is not visible, the boxer only knows that the opponent disappeared to the right or left. Further, more detailed motion, such as incoming punches, can only be detected if the joint in interest is within a cone with angle about 45 degrees from the central view vector. Hence, hooks and uppercuts may be missed if they come in from the side or below. It is assumed that the boxer can always tell approximately where he is in the ring based on the portion of the ring visible, so no limitations are made on the perception of ring boundaries.

2.2.2 The Intention Generator

Like artificial fishes, the artificial boxer has an intention generator. However, unlike artificial fishes, the conditions of the boxer's intention generator do not return true or false. Instead, they return a number between 0 and 1, reflecting the boxer's confidence in the assessment of the condition. Thus, the confidence acts as a probability that the boxer will consider the condition to be true. A random number $r \in [0, 1]$ is generated, and if r is less than the confidence, the condition is considered to be true. This adds a stochastic element to the intention generator, but it also presents an opportunity to perform simple learning.

Each condition is associated with an *adjustment*, which is a positive number. When the adjustment is less than 1, it means that the boxer has a lower confidence in the condition than what the condition function actually returns. For adjustments greater than 1, the boxer has a higher confidence than what is returned by the condition function. To apply an adjustment less than 1, the confidence is simply modified by $\hat{c} = \alpha c$, where $\hat{c}$ is the adjusted confidence, c is the value returned by the condition function, and α is the adjustment value for that condition. This has the effect of lowering the confidence, but there can be problems if α becomes too small, in which case the condition nearly always is assessed to be false. It is almost certainly the case that the designer of an artificial boxer will want to specify states in which a condition must evaluate to true, regardless of the ad-

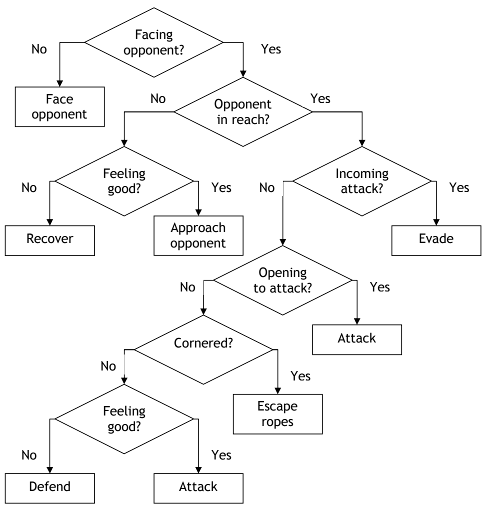

Figure 1: The Intention Generator

justment for that condition. To allow for this, when a condition function returns 1, no adjustment is applied. Adjustments greater than 1 are handled in a similar manner, with $\hat{c} = 1 - \frac{(1-c)}{\alpha}$ for $c \neq 0$.

Figure 1 shows the intention generator tree used for the boxers. Adjustments are learnt by scoring paths taken in the intention generator tree. When an event occurs that is beneficial to a boxer, such as landing a punch, the adjustments for each condition in the path taken to the intention that resulted in the positive behaviour are modified. Specifically, conditions that evaluated to true have their adjustments raised, and those that were false have their adjustments lowered. Hence, in the future, the same successful path will be more likely to be taken. Negative events, such as a blocked punch or taking an opponent's hit are scored similarly. However, some nodes in the path will be scored more often than others, so they should receive a smaller modification to their adjustment. For example, the facing condition will always receive a modification since it is at the root of the tree. To address this, each node up from the last condition before the selected intention will receive half the modification of the node further down the tree in the path.

2.2.3 The Behavioural Routines

Once an intention has been generated, it is passed to the behaviour routines. This simplifies design of the behavioural system of the boxer by separating high level strategies from low level specific action selection. Each of the intentions must be represented with a behaviour routine: face opponent, approach opponent, recover, evade, attack, escape ropes, and defend. This system of selecting actions given intentions presents a method of controlling a boxer that does not require a user to be concerned with selecting a specific action. Instead, a user may supply intentions with key presses, or perhaps mouse gestures, and let the boxer decide how to interpret the intention with the behaviour routines. For our purposes, the behaviour routines are fairly simple. One example is the face opponent behaviour routine, which queries the boxer's perception for the approximate angle to the opponent and then activates either a large or small step to the left or right, depending on what is perceived to be the quickest way to line up the enemy. The attack behaviour routine will stochastically select a punch based on the distance to the opponent and the state of the boxer. No learning is applied at the behaviour routine level, but it is not difficult to imagine a system of informing the behaviour routines of their success or failure and allowing them to modify themselves in a routine dependent fashion to complement the learning in the intention generator. The main barrier to this would be the amount of coding required. Since each behaviour routine functions in a different way, the reinforcement learning would have to be custom designed for each routine.

3 Results

In the final system, two boxers load their profiles from XML files, and one may be controlled by either the user or a reactive intention generator. User control is in the form of keyboard input that corresponds to either direct action commands or intentions. While mouse input has been considered, it would not be particularly useful because most actions must play out to completion before a new action may begin. This is to encourage realistic looking motion, and it prevents intention dithering which occurs even more so in this system than artificial fishes, because of the stochastic element. The only actions that may be interrupted with a new action are the turns, because without this it is very difficult to face the opponent with any accuracy. Even with this, we only allow interruption to occur at tenth of a second intervals to reflect the reaction time of a real boxer. We have experimented with allowing other locomotion animations to be interruptible, but this results in a reactive boxer that twitches constantly, changing its mind between whether it should turn or walk. Hence, the control of the mouse could only be used for gestures corresponding to normal input and perhaps turning angles.

The learning system of the intention generator turns out to be quite sensitive to the reward and punishment scheme used. For a while we had trouble with the boxers discovering that they could stay far enough from each other to be safe and just stand in place recovering energy without fighting. This was corrected by punishing the recover intention when the boxer is not tired, as this will not win a fight.

In testing with a user controlled boxer, we found that a fairly aggressive style is effective, and it is somewhat reassuring that more than once, one of the boxers has discovered this and raised the adjustment on the opening condition significantly. Often this may result in the other boxer getting beaten so badly that he is fairly helpless. In normal conditions, one can observe some characteristics of a real boxing match, with fighters dodging punches and catching each other off guard.

Limitations can be seen in the system of adjusting the intention generator since conditions may only be favoured or disfavoured, making them more or less likely to be true independent of the state of the environment beyond the condition function. This is a strength of the approach for its simplicity and speed, but it also prevents any deeper cognitive learning.

4 Conclusions

We have presented a method of animating and controlling a reactive boxer using motion capture and a dynamic intention generator. Much of the way boxers behave is reliant on the choices of weights for rewards and punishments. However, despite the apparent simplicity of the approach with the intention generator consisting of only six conditions and seven behaviour routines, a wide range of behaviours can be observed. Some interesting directions for future work in reactive motion might be to make modifications to the intention generator's connectivity as in [7], running hundreds of simulations to automatically generate effective intention generators from a selection of conditions. Another possibility is to incorporate more information about the state of the environment as in [1], if the limitations of the huge possible state space inherent in a sport even as simple as boxing can be overcome. One might also consider learning at the behaviour routine level, as has been noted. Indeed, reactive motion is an area with a wide range of possibilities that have yet to be explored. With this project we have found some degree of success in extending methods designed for much simpler creatures to animating reactive boxers.

A Boxing Strategy

Boxers fight favouring one side toward the opponent; a right-handed boxer will approach the opponent with his left side, and a left-handed boxer will do the opposite. In this work, we always assume a right-handed boxer, but technique is the same for a left-handed boxer. Keeping one's side toward the enemy makes it easier to defend oneself as the vulnerable midsection is turned away from any direct punches the opponent might throw, and it gives the left hand a shorter distance to travel to the opponent. A boxer would want to give the weaker arm more reach because it is used defensively for the most part, while the stronger arm is reserved for more powerful punches.

There are essentially four types of punches in boxing: the jab, cross, hook, and uppercut [6]. The jab is the only punch thrown with the left arm, and it is used either defensively to keep the opponent at a distance and off balance or offensively to set up more powerful punches. The cross, also known as a straight right, is typically used as a power punch to keep the opponent back, and it also is the easiest and safest of the right-handed punches to execute. The hook utilizes the right hand as well, but it requires arching and turning of the body which can leave the boxer vulnerable. This risk is made acceptable by the fact that it is often difficult to see an approaching hook, because they come in from the side. Finally, the uppercut is even more risky because of the way it leaves a boxer open to quicker attacks, and it's range is limited. However, a well placed uppercut can be quite powerful, making it dangerous for both boxers involved.

Offence makes up only part of the strategy necessary for a successful boxer. For all but the most powerful boxers, careful defence must play an important role. This includes making use of blocks, raising one's gloves to cover the face while using the elbows to protect the chest and midsection. While blocks are effective at stopping most punches from causing serious damage to a boxer, it can be draining and demoralizing to try to block too many power punches. To throw an opponent off, one must dodge and throw feints and jabs. A feint is a motion that resembles the start of the punch, but without the follow through. This is to make it more difficult for an opponent to read a boxer's motion, so when a real punch does come, the opponent won't be ready for it.

Beyond the general strategies described here, there are notable styles of boxing: grinding, boxing, and punching. A grinder is the most rare of

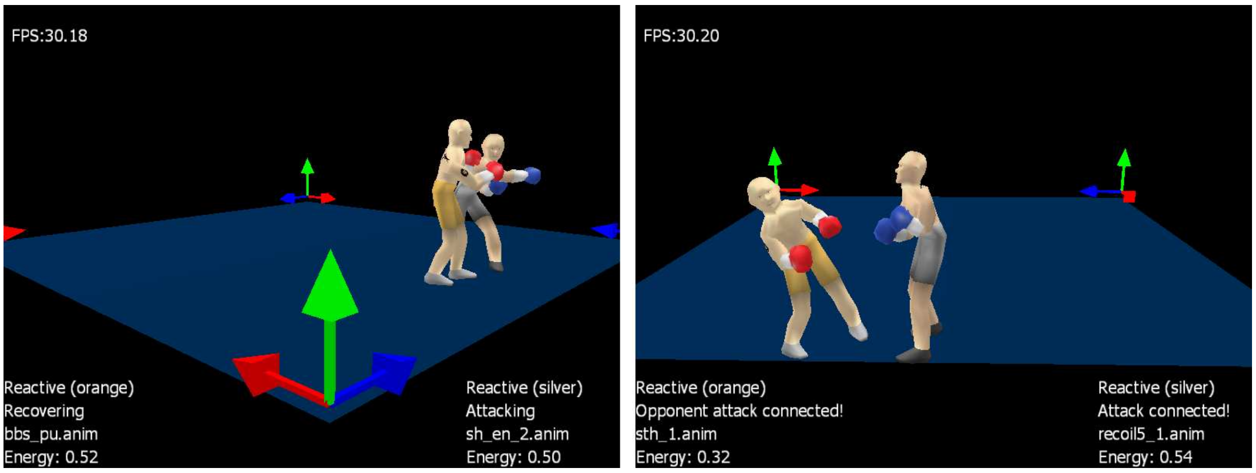

Figure 2: Reactive boxers fighting

the three types of boxers one might see. A grinder would be characterized by playing defensively and chipping away at an opponent. This is especially effective against boxers who are not particularly good at conserving there energy. After tiring themselves out in perhaps several rounds, a grinder would finish off such a boxer. Punchers, on the other hand, rely on their power to throw big punches. While this approach may be careless, when boxers like Mike Tyson in his early career use this style, it can take only one good hit to end a match. The boxing style is well rounded, something of a middle ground between grinders and punchers.

References

[1] Bruce Blumberg, Marc Downie, Yuri Ivanov, Matt Berlin, Michael Patrick Johnson, and Bill Tomlinson. Integrated learning for interactive synthetic characters. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 417–426. ACM Press, 2002.

[2] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 251–260. ACM Press, 2001.

[3] John Funge, Xiaoyuan Tu, and Demetri Terzopoulos. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 29–38. ACM Press/Addison-Wesley Publishing Co., 1999.

[4] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 9–20. ACM Press, 1998.

[5] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 473–482. ACM Press, 2002.

[6] Colin McMillan. Boxing: Blow by blow, 2004. `http://news.bbc.co.uk/sport2/hi/boxing/3372009.stm`.

[7] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM Press, 1994.

[8] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: physics, locomotion, perception, behavior. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 43–50. ACM Press, 1994.

[9] Victor B. Zordan and Jessica K. Hodgins. Motion capture-driven simulations that hit and react. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 89–96. ACM Press, 2002.