ADAPT: The Agent Development and Prototyping Testbed

Alexander Shoulson* Nathan Marshak† Mubbasir Kapadia‡ Norman I. Badler§

University of Pennsylvania, Philadelphia, PA, USA

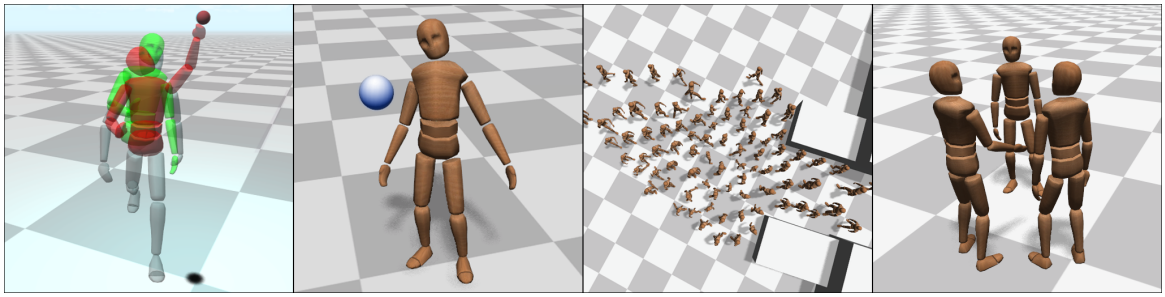

Figure 1: *Demonstrating the capabilities of ADAPT. Visualizing multiple choreographers that blend to produce a pose for the display model; an agent reacting to the impact force of a ball; a crowd of* 100 *agents resolving a bottleneck; three characters engaged in a conversation.*

Abstract

We present ADAPT, a flexible platform for designing and authoring functional, purposeful human characters in a rich virtual environment. Our framework incorporates character animation, navigation, and behavior with modular interchangeable components to produce narrative scenes. Our animation system provides locomotion, reaching, gaze tracking, gesturing, sitting, and reactions to external physical forces, and can easily be extended with more functionality due to a decoupled, modular structure. Additionally, our navigation component allows characters to maneuver through a complex environment with predictive steering for dynamic obstacle avoidance. Finally, our behavior framework allows a user to fully leverage a character's animation and navigation capabilities when authoring both individual decision-making and complex interactions between actors using a centralized, event-driven model.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: Virtual Humans, Character Animation, Behavior Authoring, Crowd Simulation

Links: DL PDF

*e-mail:ashoulson@gmail.com
†e-mail:nmarshak@seas.upenn.edu
‡e-mail:mubbasir.kapadia@gmail.com
§e-mail:badler@seas.upenn.edu

1 Introduction

Animating interacting virtual humans in real-time is a complex undertaking, requiring the solution to numerous tightly coupled problems such as steering, path-finding, full-body character animation (e.g. locomotion, gaze tracking, and reaching), and behavior authoring. This complexity is greatly amplified as we increase the number and degree of sophistication of characters in the environment. Numerous solutions for character animation, navigation, and behavior design exist, but these solutions are often tailored to specific applications, making integration between systems arduous. Integrating multiple character control architectures requires a deep understanding of each controller's design so that they may communicate with one another; otherwise character controllers will conflict at overlapping parts of the body and produce visual artifacts by naïvely overwriting one another. Directly modifying arbitrary character controllers to cooperate with one another and respond to external behavior commands can be costly and time-consuming. Monolithic, feature-rich character animation systems do not commonly support modular access to only a subset of their capabilities, while simpler systems lack control fidelity. Realistically, no sub-task of character control has a "perfect" solution. An ideal character animation system would allow a designer to choose between preferable techniques for producing a particular action or animation, leveraging the wealth of established systems already produced by the character animation research community and interfacing with robust frameworks for behavior and navigation.

We present a modular system that allows for the seamless integration of multiple character animation controllers on the same model, without requiring any controller to drastically change or accommodate any other. Rather than requiring a tightly-coupled set of character controllers, ADAPT uses a system for blending arbitrary poses in a user-authorable dataflow pipeline. Our system couples these animation controllers with an interface for path-finding and steering, as well as a comprehensive behavior authoring structure for authoring both individual decision-making and complex interactions between groups of characters. Our platform generalizes to allow the addition of new character controllers and behavior routines with minimal integration effort. Since controllers do not need to be fundamentally redesigned to work with one another, we avoid the combinatorial effect of having to modify each pre-existing controller to adjust for the change. Our system for character control

contributes to our core goal of providing a platform for experimentation in character animation, navigation, and behavior authoring. We allow researchers to rapidly iterate on character controller designs with visual feedback, compare their results with other established systems on the same model, and use features from other packages to provide the functionality they lack without the need to deeply integrate or reinvent known techniques.

2 Related Work

There exists a wealth of research [Pelechano et al. 2008] in virtual human simulation that separately addresses the problems of character animation, steering and path-finding, and behavior authoring.

Character Animation. Data-driven approaches [Kovar et al. 2002] use motion-capture data to animate a virtual character. Motion data can be manipulated by warping [Witkin and Popovic 1995] or blending [Menardais et al. 2004] to enforce parametric constraints on a recorded action. Interactive control of virtual characters can be achieved by searching through motion clip samples for desired motion as an unsupervised process [Lee et al. 2002], or by extracting descriptive parameters from motion data [Johansen 2009]. Procedural methods are used to solve specific tasks such as reaching, and can leverage empirical data [Liu and Badler 2003], example motions [Feng et al. 2012b], or hierarchical inverse kinematics [Baerlocher and Boulic 2004] for more natural movement. Physically-based approaches [Faloutsos 2002; Yin et al. 2007] derive controllers to simulate character movement in a dynamic environment. We refer to Pettré et. al. [2008] for a more extensive summary of work in these areas.

Steering and Path-finding. For navigation, the environment itself is often described and annotated as a reduction of the displayed geometry to be used in path planning. Probabilistic roadmaps superimpose a stochastic connectivity structure between nodes placed in the maneuverable space [Kavraki et al. 1996]. Navigation meshes [Kallmann 2010] provide a triangulated surface upon which agents can freely maneuver. Steering techniques use reactive behaviors [Reynolds 1999] or social force models [Helbing and Molnar 1995] to perform goal-directed collision avoidance in dynamic environments. Predictive approaches [Paris et al. 2007; van den Berg et al. 2008; Kapadia et al. 2009; Singh et al. 2011a] enable an agent to avoid others by anticipating their movements. Recast [Mononen 2009] provides an open-source solution to generating navigation meshes from arbitrary world geometry by voxelizing the space, and the associated Detour library provides path planning and predictive steering on the produced mesh. Pelechano et. al. [2008] provide a detailed review of additional work in this field.

Behavior Authoring. Animating behaviors in virtual agents has been addressed using multiple diverse approaches, particularly with respect to how behaviors are designed and animated. Early work focuses on imbuing characters with distinct, recognizable personalities using goals and priorities [Loyall 1997] along with scripted actions [Perlin and Goldberg 1996]. Our system makes use of parameterized behavior trees [Shoulson et al. 2011] to coordinate interactions between multiple characters. The problem of managing a character's behavior can be represented with decision networks [Yu and Terzopoulos 2007], cognitive models [Fleischman and Roy 2007], and goal-oriented action planning [Young and Laird 2005; Kapadia et al. 2011]. Very simple agents can also be simulated on a massive scale using GPU processing [Erra et al. 2010].

Multi-Solution Platforms. End-to-end commercial solutions [Massive Software Inc. 2010; Autodesk, Inc. 2012] combine multiple diverse character control modules to accomplish simultaneous tasks on the same character, incorporting navigation, behavior, and/or robust character animation. SteerSuite [Singh et al. 2009] is an open-source platform for developing and evaluating steering algorithms. SmartBody [Shapiro 2011] is an open-source system that combines steering, locomotion, gaze tracking, and reaching. These tasks are accomplished with 15 controllers working in unison to share control of parts of the body. SmartBody's controllers are hierarchically managed [Kallmann and Marsella 2005] where multiple animations, such as gestures, are displayed on a virtual character using a scheduler that divides actions into phases and blends those phases by interpolation. The controllers must either directly communicate and coordinate, or fix cases where their controlled regions of the body overlap and overwrite one another, making the addition of a new controller a process that affects several other software components. SmartBody also provides a navigation system with dynamic obstacle avoidance. Our platform shares some qualities with SmartBody, but also differs in several fundamental ways. While we do provide a number of character controllers for animating a virtual human, our work focuses more on enabling high-level behavioral control of multiple interacting characters, the modularity of these character controllers, and the ease with which a user can introduce a new animation repertoire to the system without disturbing the other controllers already in place.

3 Framework

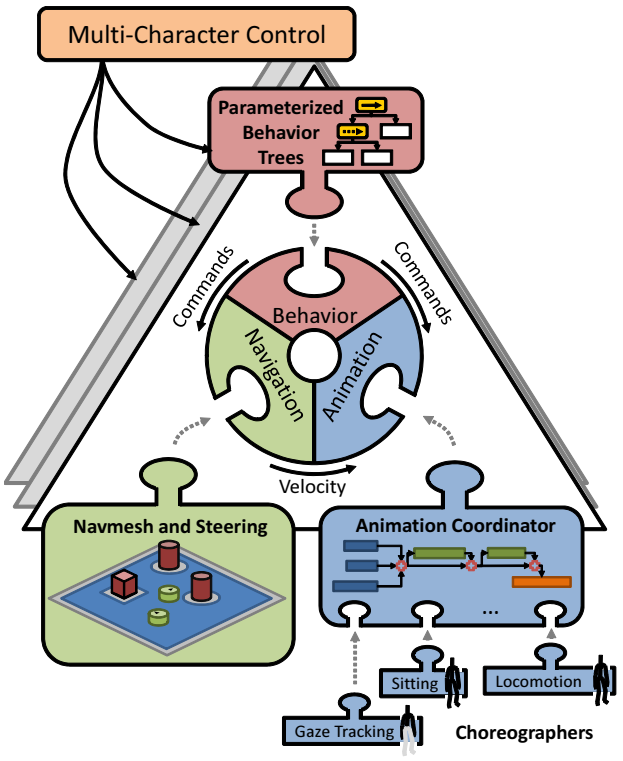

Figure 2: *Overview of ADAPT, illustrating the structure for controlling an individual character and all of the characters in an environment. Every character has a core interface for behavior, navigation, and animation, each of which connects to more specific modular components. Top-level narrative control communicates with each character through the behavior interface.*

ADAPT operates at multiple layers with interchangeable, lightweight components, and we focus on minimizing the amount of communication and interdependency between modules (Figure 2). The animation system performs control tasks such as locomotion, gaze tracking, and reaching as independent modules, called

choreographers, that can share parts of the same character's body without explicitly communicating or negotiating with one another. These modules are managed by a coordinator, which acts as a central point of contact for manipulating the virtual character's pose in real-time. The navigation system performs path-finding with predictive steering and we provide a common interface to allow users to interchange the underlying navigation library without affecting the functionality of the rest of the framework. The behavior level is split into two tiers. Individual behaviors are attached to each character and manipulate that character using the behavior interface, while a centralized control structure orchestrates the behavior of multiple interacting characters in real-time. The ultimate product of our system is a pose for each character at an appropriate position in the environment, produced by the animation coordinator and applied to a rendered virtual character in the scene each frame.

3.1 Full-Body Character Control

Controlling a fully-articulated character is traditionally accomplished using a series of interwoven subcomponents responsible for various parts of the body. Without prior knowledge of other systems, a designer creating a character controller will generally do so with the assumption that no other systems are acting on the rigged model at the same time. If a controller sets the orientation or position of a character's joint, it does so expecting no other controller to overwrite that orientation or position in the current frame. If two controllers conflict and overwrite one another, the constant changes cause visual artifacts such as jitter as the character rapidly shifts between the two settings for its joints. Controllers can be made to share control of a single body either by negotiating with one another, or by dividing the body into sections and controlling those sections alone. However, this requires that the controllers be specifically designed to coordinate, which requires additional effort on either the designer or the user of the control system. The addition of new functionality also becomes more difficult as all of the previous body controllers must be modified to communicate with any new components and share control of the body's joints.

To address this issue, we divide the problem of character animation into a series of isolated, modular components called *choreographers* attached to each character. Each choreographer operates on a *shadow*, which is an invisible clone of the character skeleton, and has unmitigated control to manipulate the skeletal joints of its shadow. Each frame, a choreographer produces an output pose consisting of a snapshot of the position and orientation of each of the joints in its private shadow. A *coordinator* receives the shadow poses from each choreographer and performs a weighted blend to produce a final pose that is applied to the display model for that frame. Since each choreographer has its own model to manipulate without interruption, choreographers do not need to communicate with one another in order to share control of the body or prevent overwriting one another. This allows a single structure, the coordinator, to manage the indirect interactions between choreographers using a simple, straightforward, and highly authorable process centered around blending the shadows produced by each choreographer. This system is discussed in more detail in Section 4.

3.2 Steering and Path-finding

We use a navigation mesh approach for steering and path-finding with dynamic obstacle avoidance. Each display model is controlled by a point-mass system, which sets the root positions (usually the hips) of the display model and each shadow every frame. We use a common interface for navigation with basic commands such as setting a goal position in the world. Character choreographers do not directly communicate with the navigation layer. Instead, choreographers are made aware of the position and velocity of the character's root, and will react to that movement on a frame-by-frame basis. A character's orientation can follow several different rules, such as facing forward while walking, or facing in an arbitrary direction, and we handle this functionality outside of the navigation system itself. ADAPT supports both the Unity3D built-in navigation system and the Recast/Detour library [Mononen 2009] for path-finding and predictive goal-directed collision avoidance, and users can easily experiment with alternate solutions, such as navigation graphs.

3.3 Behavior

ADAPT is designed to accommodate varying degrees of behavior control for its virtual characters by providing a diverse set of choreographers and navigation capabilities. Each character has a capability interface with commands like `ReachFor()`, `GoTo()`, and `GazeAt()` that take straightforward parameters like positions in space and send messages to that character's navigation and animation components. To invoke these capabilities, we use Parameterized Behavior Trees (PBTs) [Shoulson et al. 2011], which present a method for authoring character behaviors that emphasizes simplicity without sacrificing expressiveness. Having a single, flat interface for a character's action repertoire simplifies the task of behavior authoring, with well-described and defined tasks that a character can perform. One advantage of the PBT formalism is that they accommodate authoring behavior for multiple actors in one centralized structure. For example, a conversation between two characters can be designed in a single data structure that dispatches commands to both characters to take turns playing sounds or gestural animations. For very specific coordination of characters, this approach can be preferable over traditional behavior models where characters are authored in isolation and interactions between characters are designed in terms of stimuli and responses to triggers. ADAPT also generalizes across traditional or experimental new ways of modeling behavior to cover cases where PBTs are not the most appropriate. The behavior system is discussed in more detail in section 5.

4 Shadows in Full-Body Character Animation

Model rendering systems describe a virtual human as a skinned mesh with a hierarchical skeletal structure underneath. The movement of the body is determined by altering the position and orientation of each joint in the skeleton "rig", which in turn affects the position and orientation of that joint's children in the hierarchy. General character controllers are systems designed to manipulate the character by setting the positions and orientations of that character's joints, either via animations or procedurally with physical models or inverse kinematics. We address the problem of coordination between these controllers by allocating each character controller its own private character model, a replica of the skeleton or a subset of the skeleton of the character being controlled. Our modular controllers, called choreographers, act exactly the same way as traditional character controllers, but do so on private copies of the actual rendered character model. These skeleton clones (shadows), match the skeletal hierarchy, bone lengths, and initial orientations of the final rendered character (display model), but have no visual component in the scene. This is illustrated and described in figure 3. The general process of our character animation system has two interleaving steps. First, each choreographer manipulates its personal shadow and outputs a snapshot (called a shadow pose) describing the position and orientation of that shadow's joints at that time step. Then, we use a centralized controller to blend the shadow pose snapshots into a final pose for the rendered character. For clarity, note that "shadow" refers to the invisible articulated skeleton allocated to each choreographer to manipulate, while a "shadow

pose" is a serialized snapshot containing the joint positions and orientations for a shadow at a particular point in time.

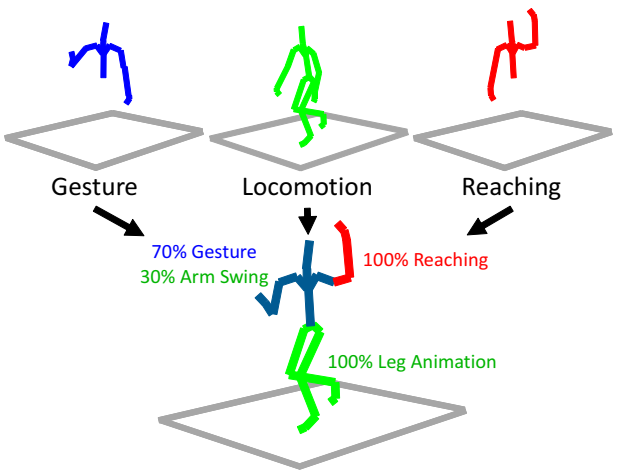

Figure 3: *Blending multiple character shadows to produce a final output skeleton pose. As an example, we combine the pose of the locomotion choreographer (green, full-body) during a walk cycle with the reaching choreographer (red, upper-body) extending the left arm towards a point above the character's head, and the gesture choreographer (blue, upper-body) playing a waving animation. The generated poses are projected, either wholly or partially, on different sections of the displayed body during any particular frame. The partial blend is represented with a mix of colors in the RGB space.*

4.1 Choreographers

The shadow pose of a character at time t is given by $\mathbf{P}_t \in \mathbb{R}^{4\times|J|}$. where $\mathbf{P}_t^j$ where is the configuration of the j^{th} joint at time t. A choreographer is a function $C(\mathbf{P}_t) \longrightarrow \mathbf{P}_{t+1}$ which produces the next pose by changing the configuration of the shadow joints for that time step. Using these definitions, we define two classes of choreographers:

Generators. Generating choreographers produce their own shadow pose each frame, requiring no external pose data to do so. Each frame, the input shadow pose $\mathbf{P}_t$ for a generator C is the pose $\mathbf{P}_{t-1}$ generated by that same choreographer in the previous frame. For example, a sitting choreographer requires no external input or data from other choreographers in order to play the animations for a character sitting and standing, and so its shadow's pose is left untouched between frames. This is the default configuration for a choreographer.

Transformers. Transforming choreographers expect an input shadow pose, to which they apply an offset each frame. Each frame, the input shadow pose $\mathbf{P}_t$ to a transformer C is an external shadow pose $\mathbf{P}'_{t+1}$ from another choreographer C', computed for that frame. The coordinator sets its shadow's pose to $\mathbf{P}'_{t+1}$ and applies an offset to the given pose during its execution, to produces a new pose $\mathbf{P}_{t+1}$. For example, before executing, the reach choreographer's shadow is set to the pose of a previously-updated choreographer's shadow (say, the locomotion choreographer with swinging arms and torso movement). The reach choreographer then solves the reach position from the base of the arm based on the torso position it was given, and overwrites its shadow's arm and wrist joints to produce a new pose. Without an input shadow, the reach choreographer would not be aware of other choreographers moving the torso, and would not be able to accommodate different torso positions when solving a reaching problem. Note that this is accomplished without the choreographers directly communicating or even being fully aware of one another. A transforming choreographer can receive an input pose, or blend of input poses, from any choreographer that has already been updated in the current frame.

4.2 The Coordinator

During runtime, our system produces a pose for the display model each frame, given the character choreographers available. This is a task overseen by the coordinator. The coordinator is responsible for maintaining each choreographer, organizing the sequence in which each choreographer performs its computation each frame, and reconciling the shadow poses that each choreographer produces by sending them between choreographers and/or blending them together. The coordinator's final product each frame is a sequence of weighted blends of each active choreographer's shadow pose. We compute this product using the *pose dataflow graph*, which dictates the order of updates and the flow of shadow poses between choreographers. Generators pass data to transformers, which can then pass their data to other transformers, until a final shadow pose is produced, blended with others, and applied to the display model.

Blending is accomplished at certain points in the pose dataflow graph denoted by *blend nodes*, which take two or more input shadows and produce a weighted blend of their transforms. If the weights sum to a value greater than 1, they are automatically normalized.

$$B(\{(\mathbf{P}_i, w_i) : i = 1..n)\}) \longrightarrow \mathbf{P}' \quad (1)$$

Designing a dataflow graph is a straightforward process of dictating which nodes pass their output to which other nodes in the pipeline, and the graph can be modified with minimal effort. The dataflow graph for a character is specified by the user during the design and authoring process, connecting choreographers with blend nodes and one another. The weights involved in blending are bound to edges in the graph and then controlled at runtime by commands from the behavior system. The order of the pose dataflow graph roughly dictates the priority of choreographers over one another. Choreographers closer to the final output node in the graph have the authority to overwrite poses produced earlier in the graph, unless bypassed by the blending system. Changing the order of nodes in the dataflow graph will affect these priorites, and so we generally design the graph so that choreographers controlling more parts of the body precede those controlling fewer.

Blended poses are calculated on a per-joint basis using each joint's position vector and orientation quaternion. The weighted average we produce accommodates cases where parts of a shadow's skeleton have been pruned or filtered from the blend (such as an upper-body shadow missing the character's legs). The blend function produces a new shadow pose that can be passed to other transformers, or be applied to the display model's skeleton. Taking a linear weighted average of vectors is a solved problem, but such is not the case with the problem of quickly averaging $n > 2$ weighted quaternions. We discuss the techniques with which we experimented, and the final calculation method we decided to use in Appendix A. In addition, Feng et. al. [2012a] provide a detailed review of more sophisticated motion blending techniques than our linear approach.

Figure 4 illustrates a sample dataflow graph, incorporating generating and transmuting choreographers, as well as four blend nodes. Three generating choreographers (blue) begin the pipeline. The gesture choreographer affects only the upper body, with no skeleton information for the lower body. Increasing the value of the gesture weight w_g places this choreographer in control of the torso, head,

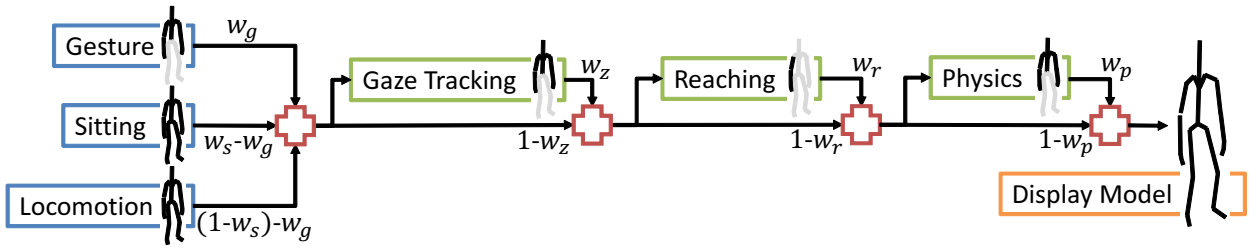

Figure 4: *A sample dataflow graph we designed for evaluating ADAPT. Generating choreographers appear in blue, transmuting choreographers appear in green, and blend nodes appear as red crosses. The final display model node is highlighted in orange. The sitting weight w_s, gesture weight w_g, gaze weight w_z, reach weight w_r, and physical reaction weight w_p are all values between some very small positive ϵ and $1 - \epsilon$.*

and arms. The sitting and locomotion choreographers can affect the entire body, and the user controls them by raising and lowering the sitting weight w_s. If w_g is set to $1 - \epsilon$, the upper body will be overridden by the gesture choreographer, but since the gesture choreographer's shadow has no legs, the lower body will still be controlled by either the sitting or locomotion choreographer as determined by the value of w_s. The first red blend node combines the three produced poses and sends the weighted average pose to the gaze tracker. The gaze tracking choreographer receives an input shadow pose, and applies an offset to the upper body to achieve a desired gaze target and produce a new shadow pose. The second blend node can bypass the gaze tracker if the gaze weight w_z is set to a low value (ϵ). The reach and physical reaction choreographers receive input and can be bypassed in a similar way. The final result is sent and applied to joints of the display model, and rendered on screen. The dataflow graph accommodates the addition of new choreographers in a generalizable fashion, allowing a user to insert new nodes and blend between the poses they create. Rather than designing animation modules to explicitly negotiate, the coordinator seamlessly fades control of parts of the body between arbitrary choreographers in an authorable pipeline.

4.3 Using Choreographers and the Coordinator

The dataflow graph, once designed, does not need to be changed during runtime or to accommodate additional characters. Instead, the coordinator provides a simple interface comprising messages and exposed blend weights for character animation. Messages are commands (e.g., `SitDown()`) relayed by the coordinator to its choreographers, making the coordinator a single point of contact for character control, as illustrated in Figure 2. In addition to messages, the weights used for blending the choreographers at each blend node in the dataflow graph are exposed, allowing external systems to dictate which choreographer is active and in control of the body (or a segment of the body) at a given point of time. For example, in Figure 4, lowering w_s will transfer control of the body to the locomotion choreographer, while raising its value will give influence to the sitting choreographer. Both choreographers are still manipulating their shadows each update, but only one choreographer's shadow pose is displayed on the body at a given time, with smooth fading transitions between the two where necessary.

For gesturing, we raise w_g, which takes control of the arms and torso away from both the locomotion and sitting choreographers and stops the walking animation's arm swing. Given sole control, the gesture choreographer plays an animation on the upper body, and then is faded back out to allow the walking arm-swing to resume. Since the gesture choreographer's shadow skeleton has no leg bones, it never overrides the sitting or locomotion choreographer, so the lower body will still be sitting or walking while the upper body gesture plays. All weight changes are smoothed over several frames to prevent jitter and transition artifacts. Note that the controllers are never in direct communication to negotiate this exchange of body control. The division of roles between the coordinator and choreographers centralizes character control to a single externally-facing character interface, while leaving the details of character animation distributed across modular components are isolated from one another and can be easily updated or replaced.

Shadow Pose Post-Processing. Since shadow poses are serializations of a character's joints, additional nodes can be added to the pose dataflow graph to manipulate shadows as they are transferred between choreographer nodes or blend nodes. For instance, special filter nodes can be added to constrain the body position of a shadow pose, preventing joints from reaching beyond a comfortable range by clamping angles, or preventing self-collisions by using bounding volumes. Nodes can be designed to broadcast messages based on a shadow's pose, such as notifying the behavior system when a shadow is in an unbalanced position, or has extended its reach to a certain distance. The interface for adding new kinds of nodes to a pose dataflow graph is highly extensible. This affords the user another opportunity to quickly add functionality to a coordinator without directly modifying any choreographers.

4.4 Example Choreographers

ADAPT provides a diverse array of character choreographers for animating a fully articulated, expressive virtual character. Some of these choreographers were developed specifically for ADAPT, while others were off-the-shelf solutions used to highlight the ease of integration with the shadow framework. ADAPT is designed to "trick" a well-behaved character control system into operating on a dedicated shadow model rather than the display model of the character, and so the process of modifying an off-the-shelf character control library to a character choreographer often requires modifying only a few lines of code. Since shadows replicate the structure and functionality of a regular character model, no additional considerations are required once the choreographer has been retargeted to the shadow. Note that the choreographers presented here are largely baseline examples. The focus of ADAPT is to allow a user to add additional choreographers, experiment with new techniques, and easily exchange generic choreographers with more specialized alternatives.

Locomotion. ADAPT uses a semi-procedural motion-blending locomotion system for walking and running released as a C# library with the Unity3D engine [Johansen 2009]. The system takes in animation data, analyzes those animations, and procedurally blends them according to the velocity and orientation of the virtual character. We produced satisfactory results on our test model using five

motion capture animation clips. Additionally, the user can annotate the character model to indicate the character's legs and feet, which allows the locomotion library to use inverse kinematics for foot placement on uneven surfaces. We extended this library to work with the ADAPT shadow system, with some minor improvements.

Gaze Tracking. We use a simple IK-based system for attention control. The user defines a subset of the upper body joint hierarchy which is controlled by the gaze tracker, and can additionally specify joint rotation constraints and delayed reaction speeds for more realistic results. These parameters can be defined as functions of the characters velocity or pose, to produce more varied results. For instance, a running character may not be permitted to rotate its torso as far as a character standing still. Integrating the gaze tracker into ADAPT required minimal changes to the existing library.

Upper Body Gesture Animations. We dedicate a shadow with just the upper body skeleton to playing animations such as hand gestures. We can play motion clips on various parts of the body to blend animations with other procedural components.

Sitting and Standing. The sitting choreographer maintains a simple state machine for whether the character is sitting and standing, and plays the appropriate transition animations when it receives a command to change state. This choreographer acts as an alternative to the locomotion choreographer when operating on the lower body, but can be smoothly overridden by choreographers acting on the upper body, such as the gaze tracker.

Reaching. We implemented a simple reaching control system based on Cyclic Coordinate Descent (CCD). We extended the algorithm to dampen the maximum angular velocity per frame, include rotational constraints on the joints, and apply relaxation forces in the iteration step. During each iteration of CCD (100 per frame), we clamp the rotation angles to lie within the maximum extension range, and gently push the joints back towards a desired "comfortable" angle for the character's physiology. These limitations and relaxation forces are based on an empirical model for reach control based on human muscle strength [Slonneger et al. 2011]. This produces more realistic reach poses than naïve CCD, and requires no input data animations. The character can reach for an arbitrary point in space, or will try to do so if the point is out of range.

Physical Reaction. By allocating an upper-body choreographer with a simple ragdoll, we can display physical reactions to external forces. Once an impact is detected, we apply the character's last pose to the shadow skeleton, and then release the ragdoll and allow it to buckle in response to the applied force. By quickly fading in and out of the reeling ragdoll, we can display a physically plausible response and create the illusion of recovery without requiring any springs or actuators on the ragdoll's joints.

SmartBody Integration. To access its locomotion and procedural reaching capabilities, we integrated the ICT SmartBody framework into our platform, using SmartBody's Unity interface and some modifications. Since our model's skeleton hierarchy differed from that of the default SmartBody characters, sample animations had to be retargeted to use on our model. Additionally, our animation interface needed to interact with SmartBody using BML. Since our coordinator is already designed to relay messages from the behavior system, changing those messages to a BML format was a straightforward conversion. Overall, the SmartBody choreographer blends naturally with other choreographers we have in the ADAPT framework, though SmartBody has other features that we do not currently exploit. This process demonstrates the efficacy of integrating other available libraries and/or commercial solutions.

5 Behavior

The navigation and shadow-based character animation system provides a number of capability functions, including:

Commands	Description
`ReachFor(target)`	Activates the reaching choreographer, and reaches towards a position.
`GazeAt(target)`	Activates the gaze choreographer, and gazes at a position.
`GoTo(target)`	Begins navigating the character to a position.
`Gesture(name)`	Activates the gesture choreographer for the duration of an animation.
`SitDown()`	Activates the sitting choreographer and sits the character down.
`StandUp()`	Stands the character up and then disables the sitting choreographer.

Passing an empty target position will end that task, stopping the gaze, reach, or navigation. The locomotion choreographer will automatically react to the character's velocity, and move the legs and arms to compensate if the character should be turning, walking, side-stepping, backpedaling, or running. Note that only sitting and navigating are mutually exclusive. All other commands can be performed simultaneously without visual artifacts.

5.1 Adding a New Behavior Capability

Adding a new behavior capability with a motion component, such as climbing or throwing an object, requires a choreographer capable of producing that motion. Choreographers can be designed to perform animation tasks based on animation data, procedural techniques, or physically-driven models. Since choreographers operate on their own private copies of the character's skeleton, they can be designed in isolation and integrated into the system separately. Once the choreographer is developed, the process of adding a new behavior capability to take advantage of the choreographer requires two steps. First, the choreographer must be authored into the pose dataflow graph, either as a generating or transforming node, with appropriate connections to blend nodes and other choreographers. Next, the behavior interface can be extended with new functions that either modify the blend weights relevant to the new choreographer, or pass messages to that choreographer by relaying them through the coordinator. The sophistication of character choreographers varies, but the process of integrating a functioning choreographer into the behavior and animation pipeline for a character is authorable and generalizable.

5.2 Multi-Character Interactions

Using this behavior repertoire, we can produce more sophisticated actions as characters interact with one another and the environment. Authoring complex behaviors requires an expressive and flexible behavior authoring structure granting the behavior designer reasonable control over the characters in the environment. To accomplish this task, we use parameterized behavior trees (PBTs). PBTs are an extension of the behavior tree formalism that allow behavior trees to manage and transmit data within their hierarchical structure without the use of a blackboard. A useful advantage of PBTs is the fact that they can simultaneously control multiple characters in a single reusable structure called an *event*. Events are pre-authored behavior trees that sit uninitialized in a library until invoked at runtime. When instantiated, an event takes one or more actors as parameters, and is temporarily granted exclusive control over those

characters' actions. While in control, an event treats these characters as limbs of the same entity, dispatching commands for agents to navigate towards and interact with one another. Once the event ends, control is yielded to the characters' own individual decision processes, which can also be designed using PBTs or with some other technique. Events are a convenient formalism to use for interactions with a high degree of interchange and turn-taking, such as conversations. A conversation event can be authored as a simple sequential and/or stochastic sequence of commands directing agents to face one another and take turns playing gesture animations or exchanging physical objects. ADAPT provides a fully-featured scheduler for managing and updating both the personal behavior trees belonging to each character and higher-level event behavior trees encompassing multiple characters.

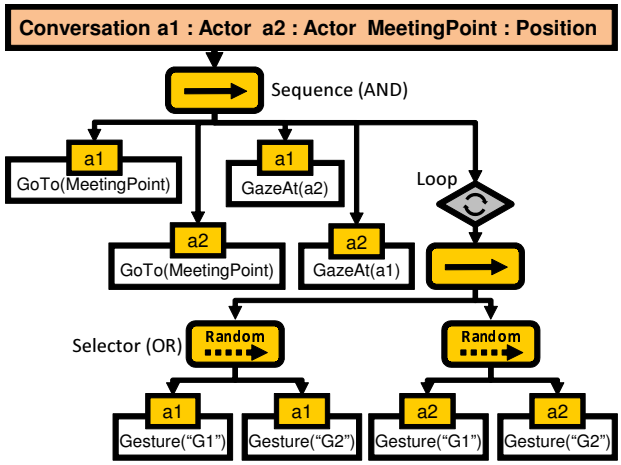

Figure 5: *A simple conversation PBT event controlling two characters,* `a1` *and* `a1`*, with one additional* `MeetingPoint` *parameter.*

Figure 5 illustrates a sample behavior tree event conducting two characters through a conversation using our action repertoire. The characters, `a1` and `a2`, are passed as parameters to the tree, along with the meeting position. Using our action interface, the tree directs the two characters to approach one another at the specified point, face each other, and alternatively play randomly selected gesture animations. The gesturing phase lasts for an arbitrary duration determined by the configuration of the loop node in the tree. After the loop node terminates, the event ends, reporting success, and the two characters return to their autonomous behaviors. Note that this tree can be reused at any time for any two characters and any two locations in the environment in which to stand. This framework can be exploited to create highly sophisticated interactions involving crowds of agents, and its graphical, hierarchical nature makes subtrees easier to describe and encapsulate.

6 Results

We demonstrate the features of ADAPT in isolation, as well as a final scene showcasing animation, navigation, and behavior working together to produce a narrative sequence (Figure 1). Using our system, we can create a character that can simultaneously reach, gaze, walk, and play gesture animations, as well as activate other functionality like sitting and physically reacting to external forces. ADAPT characters can intelligently maneuver an environment avoiding both static obstacles and one another. These features are used for authoring sequences like exchanging an object between actors, wandering while talking on a phone, and multiple characters holding a conversation.

Multi-Actor Simulation. The concluding narrative sequence shown in the video is simulated using several reusable authored events, which are activated using spatial and temporal triggers. Events once active, can be successfully executed or interrupted by other triggers due to dynamic events, or user input. This produces a rich interactive simulation where virtual characters can be directed with a high degree of fidelity, without sacrificing autonomy or burdening the user with authoring complexity.

In the beginning, an event ensues where a character is given a phone and converses while wandering through the scene, gazing at objects of interest. The phone conversation event successfully completes and the character hands back the phone. Spotting nearby friends invokes a conversation, which is an extension of the event illustrated in Figure 5. The conversation is interrupted when a ball is thrown at one of the characters. The culprit flees from the scene of the crime, triggering a chasing event where the group runs after the child. The chase fails as the child is able to escape through a crossing crowd of characters, which are participating in a group event to navigate to the theater and find a free chair to sit. We illustrate some of the trees used for this sequence in greater detail in a supplemental document.

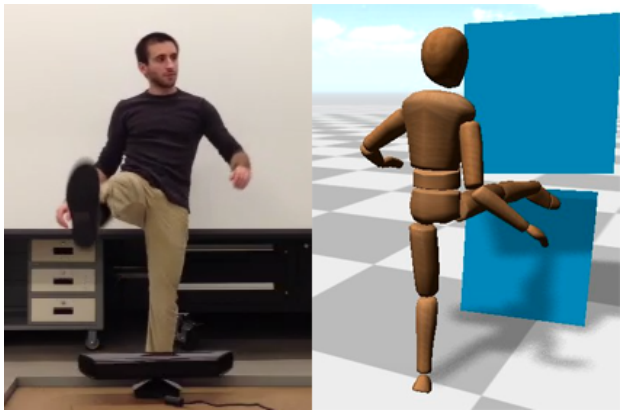

Figure 6: *Controlling a character in ADAPT and physically interacting with the environment using the Kinect.*

Adding a Kinect Choreographer. As an example of our system's extensibility, we created an additional choreographer to interface with the Microsoft Kinect and control a character with gesture input. To do so, we allocated a full-body choreographer to the input of the Kinect, applying the captured skeleton from the Kinect's framework directly to the joints of the dedicated shadow. This is demonstrated in Figure 6. Blending this choreographer with others allowed us to expand the character's agency in the world. When the character stands idle, we give full upper and lower body control to the Kinect input. When the user wishes to make the character move, we blend the legs of the locomotion choreographer on top of the Kinect input, displaying appropriate walking or running animations and foot placement while still giving the Kinect control of the upper body. This is a feasible compromise for allowing a user to retain correct leg animation when exploring a virtual environment larger than the Kinect's capture area. The process of interfacing the Kinect skeleton input with a new choreographer and a blending coordinator was very fast and straightforward.

Performance. ADAPT supports approximately 150 agents with full fidelity at interactive frame rates. Figure 7 displays the update frequency for the animation and navigation system (for our scenes, the computational cost of behavior was negligible). This varies with the complexity of the choreographers active on each character. The ADAPT animation interface and the pose dataflow graph has little impact on performance, and the blend operation is linear in num-

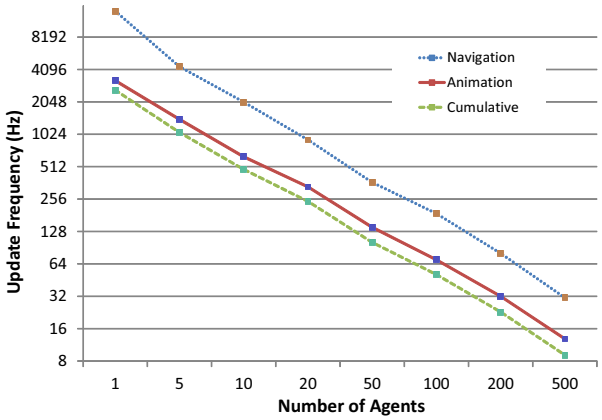

Figure 7: *Update frequency for the character animation and navigation components in ADAPT.*

ber of choreographers. Each joint in a shadow is serialized with 7 4-byte float values, making each shadow 28 bytes per joint. For 26 bones, the shadow of a full-body character choreographer has a memory footprint of 728 bytes. For 200 characters, the maximum memory overhead due to shadows is less than 1 MB. In practice, however, most choreographers use reduced skeletons with only a limb or just the upper body, making the actual footprint much lower for an average character.

Separating character animation into discrete modules and blending their produced poses as a post-processing effect also affords the system unique advantages with respect to dynamic level-of-detail (LOD) control. Since no choreographer is architecturally dependent on any other, controllers can be activated and deactivated arbitrarily. Deactivated controllers can be smoothly faded out of control at any time, and their nodes in the dataflow graph can be bypassed using the already-available blend weights. This drastically reduces the number of computed poses, and conserves processing resources needed for background characters that do not require a full repertoire of actions. The system retains the ability to re-activate those choreographers at any time if a specific complex action is suddenly required. Since choreographers are not tightly coupled, no choreographer needs to be made aware of the fact that any other choreographer has been disabled for LOD purposes.

7 Conclusions

ADAPT is a modular, flexible platform which provides a comprehensive feature set for animation, navigation, and behavior tools needed for end-to-end simulation development. By allowing a user to independently incorporate a new animation choreographer or steering system, and make those components immediately accessible to the behavior level without modifying other existing systems, characters can very easily be expanded with new capabilities and functionality. We are releasing ADAPT as an open-source project not only to provide animation, steering, and behavior package to community, but to provide a platform that is designed to allow users to tailor the system to fit their own personal needs, to rapidly iterate on experimental designs, and to compare their results against other established techniques. The library, assets, and documentation are available at `http://cg.cis.upenn.edu/ADAPT`

7.1 Limitations and Complications

The ADAPT platform has some surmountable issues that arise from blending poses as a post-process. Solutions to these complications either already exist in the system or could be introduced with future work.

Cross-Choreographer State Awareness. At times, choreographers may need to be aware of major state changes in the character's pose caused by another choreographer. For example, we may wish to restrict the degree to which the character can rotate its torso for gaze tracking while the character is running. We accomplish this using the message broadcast system integrated into the coordinator. When a character reaches a certain speed, the locomotion choreographer can broadcast to all other choreographers that the character is in an `IsRunning` state. The gaze tracking choreographer can receive this message and restrict its maximum torso rotation accordingly. This allows choreographers to cooperate without being explicitly aware of one another, and is a more extensible paradigm than deep integration of controllers.

Foot-Placement Artifacts. Interpolation between arbitrary poses generally produces smooth results in our system, with the exception of blends that linearly translate the position of a character's feet. This situation arises with our sitting choreographer, where the placement of a character's feet while standing may not coincide with the foot placement in the transition animation between standing and sitting. A linear blend here results in an unrealistic sliding of the foot despite ground contact. This issue could be easily solved with a slightly more robust locomotion system that allowed arbitrary foot placement, so that we could use a special case to adjust the feet to step to the proper position before blending from the locomotion to the sitting choreographer. Since each choreographer is aware of which parts of the body it uses, and how it wants to pose each joint, this solution could be generalized for transitioning between any choreographers that use the lower body of the character.

7.2 Future Work

Moving forward, we will continue to expand the animation and authoring capabilities supported by ADAPT. For example, the design of the pose dataflow graph is one possible avenue for improvement. Currently, a designer manually organizes the choreographer and blend nodes in the structure of the coordinator. While this is a conceptually simple task because the dataflow graph is so easy to visualize, we have yet to develop a scripting or graphical interface to make the process more accessible to a completely untrained user. More importantly, however, we believe the process of authoring a dataflow graph can be completely automated based on which parts of the body each choreographer uses.

Another main development effort is the production of more capable choreographers for use in the ADAPT framework. We would like to develop or incorporate a better locomotion system with the ability to control an autonomous character with footstep-level precision [Singh et al. 2011b]. One major advantage in this effort is the ability to directly integrate other developed systems into the ADAPT framework and seamlessly blend them with the rest of our choreographers, as we have done with the SmartBody package. In addition to choreographers described here, we want our platform to provide an array of options for different kinds of motor skills, including jumping, climbing, and carrying objects with weight.

Finally, we are also interested in improving the virtual environment and developing extensible ways for characters to interact with the environment on a behavioral level. To ease the authoring burden, we are currently creating an interface similar to smart objects for annotating the environment and describing the ways that characters

can interact with it. We are particularly interested in extending the ADAPT platform to develop solutions for the automated scheduling of events to follow global narrative arcs. All of these improvements will allow us to apply our platform to other areas research, as ADAPT is uniquely suited for producing the next generation of narrative-driven simulations.

Acknowledgments

We acknowledge Dr. Ari Shapiro and Dr. Ben Sunshine-Hill for their discussions and contributions to the ADAPT project. The research reported in this document was performed in connection with Contract Numbers W911NF-07-1-0216 and W911NF-10-2-0016 with the U.S. Army Research Laboratory. The views and conclusions contained in this document are those of the authors and should not be interpreted as presenting the official policies or position, either expressed or implied, of the U.S. Army Research Laboratory, or the U.S. Government unless so designated by other authorized documents. Citation of manufacturers or trade names does not constitute an official endorsement or approval of the use thereof. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation heron.

Appendix A: On Quaternion Blending

While vectors can be easily averaged using addition and scalar multiplication in $\mathbb{R}^3$, interpolating between quaternions is not as simple due to the spherical surface of the unit-quaternion space. For our coordinator to be able to blend shadow poses between choreographers, we require a fast method for blending the quaternions describing the rotations of the character's joints in the unit-quaternion manifold. Our ultimate goal is to develop a function $\text{Blend}((q_1, w_1), (q_2, w_2), \ldots, (q_n, w_n)) = q$ taking n weighted unit quaternions and producing an average quaternion q. To create this function, we experimented with a number of different techniques, and felt it valuable to document our efforts here.

Slerp. Spherical linear interpolation (Slerp) [Shoemake 1985] is a constant-time operation for interpolating between two unit quaternions q_1 and q_2, using an interpolation weight w. Slerp is defined equivalently as follows:

$$\begin{aligned} \text{Slerp}(q_1, q_2; w) &= q_1(q_1^{-1}q_2)^w && (2)\\ &= q_2(q_2^{-1}q_1)^{1-w} && (3)\\ &= (q_1q_2^{-1})^{1-w}q_2 && (4)\\ &= (q_2q_1^{-1})^w q_1 && (5) \end{aligned}$$

Slerp has the ideal properties of being a closed form solution, and generally taking the shortest path between two quaternions. Because of this, we considered using a chain of Slerp operations. For instance, with three quaternions q_1, q_2, and q_3 and two blend weights w_0, w_2, we could perform:

$$\begin{aligned} &\text{Blend}_{SLERP}(q_1, q_2, q_3; , w_1, w_2)\\ &\quad = \text{Slerp}(\text{Slerp}(q_1, q_2; w_1), q_3; w_2) \end{aligned} \tag{6}$$

Ultimately, we abandoned the idea primarily because operation would not be commutative in all cases, which could create unexpected and confusing results in the authoring process, especially as the number of input quaternions grew.

Angular Velocities. Another alternative is to treat each of the character's joint as a ball-and-socket joint with three rotational degrees of freedom. This allows to compute the angular velocities applies to each joint between frames, and to average those velocities when blending between each choreographer's produced motions. This had two problems. First, the process did not properly handle blending into static poses from dynamic ones. Second, this essentially converted each joint quaternion into a set of Euler angles, which suffer from the problem of gimbal lock.

Iterative Solutions. Multiple iterative solutions exist for constrained interpolation. Pennec [1998], Johnson [2003], and Buss et. al. [2001] all describe iterative techniques for finding the weighted average of quaternions on the spherical surface. While these provide accurate results and generally blend over the shortest distance, we looked for a solution with a closed form.

Because our blending is spread out over numerous frames, our quaternion blending function is usually only interpolating between very short distances. As a result, we experienced success with a naïve algorithm.

Data: Unit quaternions $q_1, \ldots, q_n$
Data: Normalized weights $w_1, \ldots, w_n$
Result: An average quaternion q
$q = [0, 0, 0, 0]$;
for $i = 1 \rightarrow n$ **do**
  if $i > 1$ **and** $q_1 \cdot q_i < 0$ **then**
    // negate every element of the quaternion;
    $q_i = [-q_i^0, -q_i^1, -q_i^2, -q_i^3]$;
  end
  for $j = 0 \rightarrow 3$ **do**
    $q^j += q_i^j * w_i$
  end
end
return $\text{Normalize}(q)$
Algorithm 1: The computation of $\text{Blend}((q_1, w_1), \ldots, (q_n, w_n))$

So long as the blend distances are short, we can naively treat the quaternion space as $\mathbb{R}^4$, and take a weighted average of each element of the quaternion. This will work provided the quaternion is normalized after the calculation. To ensure that we generally take the shortest distance between angles, we pick an arbitrary quaternion q_p and calculate the dot product of that quaternion with each other quaternion $q_{i \neq p}$. If the dot product is negative, we negate each term in q_i. Note that we negate each term rather than inverting the quaternion. This, again, is a quick approximation for blending over short distances. In practice, this quick method produces good visual results, and is computationally cheaper than multiple slerps or an iterative solution. For blends across more significant distances, we could opt for a more complicated solution, but so far have seen no need to do so.

References

AUTODESK, INC., 2012. Autodesk gameware - artificial intelligence middleware for games.

BAERLOCHER, P., AND BOULIC, R. 2004. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *Vis. Comput. 20*, 6 (Aug.), 402–417.

BUSS, S. R., AND FILLMORE, J. P. 2001. Spherical averages and applications to spherical splines and interpolation. *ACM Trans. Graph. 20*, 2 (Apr.), 95–126.

ERRA, U., FROLA, B., AND SCARANO, V. 2010. Behavert: a gpu-based library for autonomous characters. In *Motion in Games*, MIG'10, 194–205.

FALOUTSOS, P. 2002. *Composable Controllers for Physics-Based Character Animation*. PhD thesis, University of Toronto.

FENG, A. W., HUANG, Y., KALLMANN, M., AND SHAPIRO, A. 2012. An analysis of motion blending techniques. In *The Fifth International Conference on Motion in Games*.

FENG, A. W., XU, Y., AND SHAPIRO, A. 2012. An example-based motion synthesis technique for locomotion and object manipulation. I3D, 95–102.

FLEISCHMAN, M., AND ROY, D. 2007. Representing intentions in a cognitive model of language acquisition: Effects of phrase structure on situated verb learning. In *AAAI '07*, AAAI, 7–12.

HELBING, D., AND MOLNAR, P. 1995. Social force model for pedestrian dynamics. *PHYSICAL REVIEW E 51*, 42–82.

JOHANSEN, R. S. 2009. *Automated Semi-Procedural Animation for Character Locomotion*. Master's thesis, Aarhus University.

JOHNSON, M. P. 2003. *Exploiting Quaternions to Support Expressive Interactive Character Motion*. PhD thesis, Massachusetts Institute of Technology.

KALLMANN, M., AND MARSELLA, S. 2005. Lncs '05. ch. Hierarchical motion controllers for real-time autonomous virtual humans, 253–265.

KALLMANN, M. 2010. Shortest paths with arbitrary clearance from navigation meshes. In *Proceedings of the Eurographics / SIGGRAPH Symposium on Computer Animation (SCA)*.

KAPADIA, M., SINGH, S., HEWLETT, W., AND FALOUTSOS, P. 2009. Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '09, 215–223.

KAPADIA, M., SINGH, S., REINMAN, G., AND FALOUTSOS, P. 2011. A behavior-authoring framework for multiactor simulations. *Computer Graphics and Applications, IEEE 31*, 6 (nov.-dec.), 45 –55.

KAVRAKI, L., SVESTKA, P., LATOMBE, J.-C., AND OVERMARS, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE 12*, 4 (aug), 566 –580.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. SIGGRAPH, 473–482.

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. *ACM TOG 21*, 3, 491–500.

LIU, Y., AND BADLER, N. I. 2003. Real-time reach planning for animated characters using hardware acceleration. CASA, 86–93.

LOYALL, A. B. 1997. *Believable Agents: Building Interactive Personalities*. PhD thesis, Carnegie Mellon University.

MASSIVE SOFTWARE INC., 2010. Massive: Simulating life. www.massivesofware.com.

MENARDAIS, S., MULTON, F., KULPA, R., AND ARNALDI, B. 2004. Motion blending for real-time animation while accounting for the environment. CGI, 156–159.

MONONEN, M., 2009. Recast/Detour navigation library.

PARIS, S., PETTR, J., AND DONIKIAN, S. 2007. Pedestrian reactive navigation for crowd simulation: a predictive approach. *Computer Graphics Forum 26*, 3, 665–674.

PELECHANO, N., ALLBECK, J. M., AND BADLER, N. I. 2008. *Virtual Crowds: Methods, Simulation, and Control*. Synthesis Lectures on Computer Graphics and Animation.

PENNEC, X. 1998. Computing the Mean of Geometric Features Application to the Mean Rotation. Tech. Rep. RR-3371, INRIA, Mar.

PERLIN, K., AND GOLDBERG, A. 1996. Improv: a system for scripting interactive actors in virtual worlds. SIGGRAPH, 205–216.

PETTRÉ, J., KALLMANN, M., AND LIN, M. C. 2008. Motion planning and autonomy for virtual humans. In *ACM SIGGRAPH 2008 classes*, ACM, New York, NY, USA, SIGGRAPH '08, 42:1–42:31.

REYNOLDS, C., 1999. Steering behaviors for autonomous characters.

SHAPIRO, A. 2011. Building a character animation system. MIG, 98–109.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph. 19*, 3 (July), 245–254.

SHOULSON, A., GARCIA, F., JONES, M., MEAD, R., AND BADLER, N. I. 2011. Parameterizing Behavior Trees. In *Proceedings of the 4th International Conference on Motion in Games (MIG '11)*, Springer, 144–155.

SINGH, S., KAPADIA, M., FALOUTSOS, P., AND REINMAN, G. 2009. An open framework for developing, evaluating, and sharing steering algorithms. In *Proceedings of the 2nd International Workshop on Motion in Games*, Springer-Verlag, Berlin, Heidelberg, MIG '09, 158–169.

SINGH, S., KAPADIA, M., HEWLETT, B., REINMAN, G., AND FALOUTSOS, P. 2011. A modular framework for adaptive agent-based steering. In *Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '11, 141–150 PAGE@9.

SINGH, S., KAPADIA, M., REINMAN, G., AND FALOUTSOS, P. 2011. Footstep navigation for dynamic crowds. *Computer Animation and Virtual Worlds 22*, 2-3, 151–158.

SLONNEGER, D., CROOP, M., CYTRYN, J., JR., J. T. K., RABBITZ, R., HALPERN, E., AND BADLER, N. I. 2011. Human model reaching, grasping, looking and sitting using smart objects international symposium on digital human modeling. Proc. International Ergonomic Association Digital Human Modeling.

VAN DEN BERG, J., LIN, M. C., AND MANOCHA, D. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In *ICRA*, IEEE, 1928–1935.

WITKIN, A., AND POPOVIC, Z. 1995. Motion warping. SIGGRAPH, 105–108.

YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. Simbicon: simple biped locomotion control. *ACM TOG 26*, 3.

YOUNG, R. M., AND LAIRD, J. E., Eds. 2005. Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA, AAAI Press.

YU, Q., AND TERZOPOULOS, D. 2007. A decision network framework for the behavioral animation of virtual humans. SCA, 119–128.