ADAPT: The Agent Development and Prototyping Testbed

Alexander Shoulson, Nathan Marshak, Mubbasir Kapadia, and Norman I. Badler

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. The 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. The 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.

1 Introduction

Animating interacting virtual humans in real-time is a complex undertaking, requiring the solution to 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 amplified as we increase the number and 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 controllers requires a deep understanding of each controller's design and combinatorial communication between each other to avoid conflicts that lead to motion artifacts. This is further exacerbated when using hybrid control architectures (e.g., motion capture data, inverse kinematics, physically based animation) in a single system. 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, leveraging the wealth of established systems already produced by the character animation research community and interface 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 tightly coupling a fixed set of character controllers, ADAPT uses a system for blending arbitrary poses in a user-authorable dataflow pipeline. Our system combines 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. New control capabilities can be easily integrated into ADAPT without being aware of, or requiring any changes in exisiting controllers. Our system provides 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.

This paper makes the following contributions:

- An extensible, scalable platform for animating crowds of autonomous virtual characters in sophisticated 3D virtual worlds.
- Automatic integration of diverse controllers, facilitating data-driven, physically based, or IK based solutions to simultaneously operate on overlapping subsets of the character's body without the need for any conflict resolution.
- A graphical, hierarchical behavior authoring system empowering authors to design and orchestrate fine-grained character movements, as well as complex, coordinated, and dynamic multi-character interactions.

- *All authors are with the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 19104.*
- *A. Shoulson: ashoulson@gmail.com*
- *N. Marshak: nmarshak@seas.upenn.edu*
- *M. Kapadia: mubbasir.kapadia@gmail.com*
- *N. Badler: badler@seas.upenn.edu*

Deliverable. We provide ADAPT: an open-source library that comines a suite of modular character controllers including data-driven locomotion, procedural reaching, gesturing, and physical reactions, with integrated navigation, and event-centric behavior authoring for multi-actor interactions to create an extensible end-to-end crowd authoring tool for research.

2 Related Work

There exists a wealth of research [1], [2], [3] that separately addresses the problems of character animation, steering and path-finding, and behavior authoring, with many open challenges [4] that need to be addressed in an effort to arrive at a common standard for simulating autonomous virtual humans for the next generation of interactive virtual world applications.

Character Animation. Data-driven approaches [5], [6] use motion-capture data to animate a virtual character. Motion clips can be manipulated and concatenated by using warping [7], blending [8], [9], layering [10], or planning [11], [12] to enforce parametric constraints on recorded actions. Interactive control of virtual characters can be achieved by searching through motion clip samples for desired motion as an unsupervised process [13], or by extracting descriptive parameters from motion data [14]. Procedural methods are used to solve specific tasks such as reaching, and can leverage empirical data [15], example motions [16], or hierarchical inverse kinematics [17] for more natural movement. Physically-based approaches [18], [19] derive controllers to simulate character movement in a dynamic environment. We refer to Pettré et. al. [20] 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 [21]. Navigation meshes [22] provide a triangulated surface upon which agents can freely maneuver. Potential Fields [23] generate a global field for the entire landscape where the potential gradient is contingent upon the presence of obstacles and distance to goal, but is prone to local minima. Dynamic potential fields [24] have been used to integrate global navigation with moving obstacles and people, efficiently solving the motion of large crowds without the need for explicit collision avoidance.

Steering techniques use reactive behaviors [25] or social force models [26], [27] to perform goal-directed collision avoidance in dynamic environments. Predictive approaches [28], [29] and local perception fields [30], [31] enable an agent to avoid others by anticipating their movements, while more complex scenarios such as group interactions and deadlocks are solved using hybrid techniques [32], space-time planning [33], or by externalizing steering logic [34]. Data-driven steering [35], [36] focuses on generating local-space samples from observations of real people which are used to create databases, or serve as training data to learn computational models which are queried to emulate real-human behavior. Recast [37] 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. [1] 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 [38] along with scripted actions [39]. Our system makes use of parameterized behavior trees [40] to coordinate interactions between multiple characters. The problem of managing a character's behavior can be represented with decision networks [41], cognitive models [42], goal-oriented action planning [43], [44], or via learning [45]. Very simple agents can also be simulated on a massive scale using GPU processing [46]. Recent work [47], [48], [49] proposes an event-centric authoring paradigm to facilitate multi-actor interactions with contextual awareness based on agent type and event location.

Multi-Solution Platforms. End-to-end commercial solutions [50], [51] combine multiple diverse character control modules to accomplish simultaneous tasks on the same character, incorporting navigation, behavior, and/or robust character animation. SteerSuite [52] is an open-source platform for developing and evaluating steering algorithms. SmartBody [53] 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 [54] 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. 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

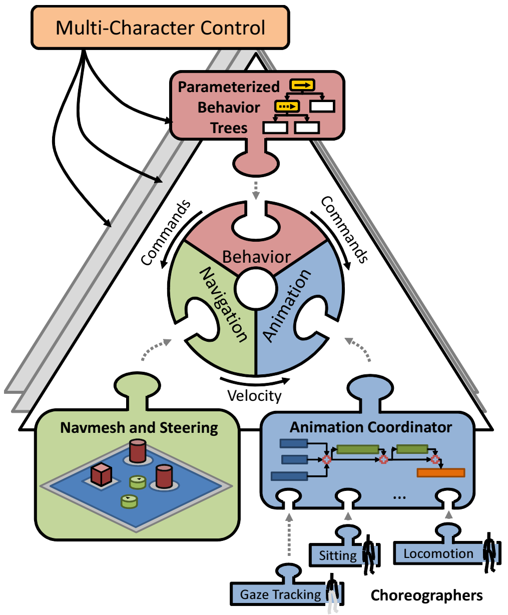


Fig. 1. Overview of ADAPT framework.

ADAPT operates at multiple layers with interchangeable, lightweight components, and minimal communication between modules. 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 replace 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. Figure 1 provides an illustration of the framework from an architectural integration perspective (the framework appears more hierarchical from a behavior control perspective, as illustrated in Figure 6).

3.1 Full-Body Character Control

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. 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 [37] for path-finding and predictive goal-directed collision avoidance, and users can supplement their own preferred solution.

3.3 Behavior

ADAPT accommodates varying degrees of behavior control for its virtual characters by providing a diverse set of choreographers and navigation capabilities. Each character has capabilities 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) [40], 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. The behavior system is discussed in more detail in section 5.

4 Shadows in Full-Body Character Animation

General character controllers animate a virtual character using pre-recorded motions, 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. Figure 2 illustrates a two-step blend process. First we combine the pose of the locomotion choreographer (green, full-body) during a walk cycle with the gesture choreographer (blue, upper-body) playing a waving animation, and then we apply the arm of the reaching choreographer (red, upper-body) full blend weight, safely overwriting the previous step for the joints of the left arm. The partial blend is represented with a mix of colors in the RGB space. Blend ordering is discussed in greater detail in Section 4.2.

Character animation 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 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.

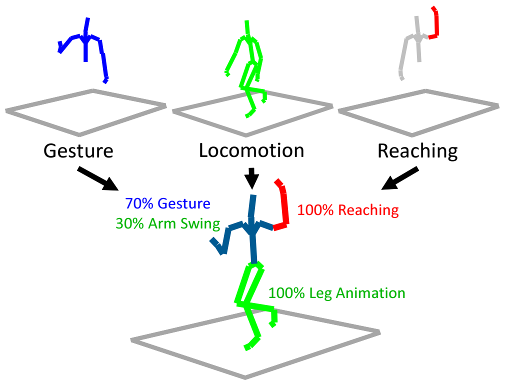


Fig. 2. Blending multiple character shadows to produce a final output skeleton pose.

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, 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 produce a new pose $\mathbf{P}_{t+1}$. For example, before executing, the reach choreographer's shadow is set to the output 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. This allows the reach choreographer to accommodate multiple torso configurations 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 other choreographer(s).

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. 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.

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

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 authoring process. 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. 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 blend function produces a new shadow pose that can be passed to other transformers, or 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 (supplemental). In addition, Feng et. al. [55] provide a detailed review of more sophisticated motion blending techniques than our linear approach.

Figure 3 illustrates a sample dataflow graph. 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, 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.

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 1. 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.

As an example, when the coordinator receives a gesture command, it raises w_g (in Figure 3), 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

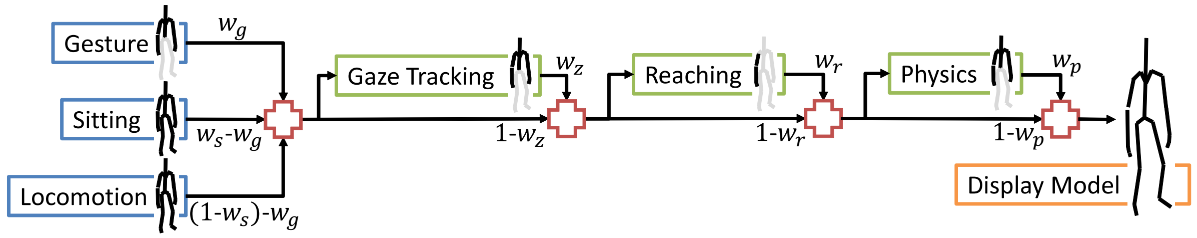


Fig. 3. A sample dataflow graph we designed for evaluating ADAPT. Generating choreographers appear in blue, transmuting choreographers appear in green, and blend nodes appear as red crosses. The final display model node is highlighted in orange. The sitting weight w_s, gesture weight w_g, gaze weight w_z, reach weight w_r, and physical reaction weight w_p are all values between some very small positive ϵ and $1 - \epsilon$.

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. 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.

Cross-Choreographer Functionality. Conflicts between choreographers are resolved using the inherent priority of the pose dataflow graph – every transformer has the option of overwriting any joint it receives as input. Choreographers are unaware of whether or not their output poses are overwritten later or blended out of the final pose. Unless a particular choreographer is the last dataflow node to affect all of the joints it requires, it cannot make any guarantees about the resulting pose it produces. For example, the reach choreographer cannot guarantee a successful reach if its arm joints are partially overwitten by the gesture choreographer. This is why the reach controller appears towards the end of the sample dataflow graph in Figure 3, giving it high overwrite priority. There are two additional ways to address priority and guarantees at runtime. First, the message passing system can be used to tell a choreographer when it is in full control of a part of the body, allowing that choreographer to give accurate feedback or request more control. Second, the pose dataflow graph can be rearranged dynamically to adjust which choreographers have priority over one another when needed. In practice, priority is handled by a few special cases in the behavior system, but the technique could be developed further.

4.4 Example Choreographers

ADAPT provides a number of diverse 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 converting an off-the-shelf character control library into a character choreographer is straightforward. Since shadows replicate the structure and functionality of a regular character model, no additional considerations are required once the choreographer has been allocated a 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 [14]. 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. The user annotates 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 small 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.

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 [56]. 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.

4.5 Automatic Coordinator Derivation

Since a coordinator may be responsible for any number of choreographers, each affecting an arbitrary number of contiguous joints, it is important for a generation system to understand the relationships between joints and the layering nature of ADAPT choreographers. We have designed a simple heuristic by which the hierarchical order of choreographer updates can be automatically generated. The result of our analysis is an ordering of choreographers to use in conjunction with two-input blend nodes for generating a pose dataflow graph. This is achieved by starting the graph with choreographers that affect broader parts of the body, and placing narrower, more focused choreographers later in the sequence. Since choreographers deeper in the pose dataflow graph have higher priority over the parts of the body for which they're responsible, we assume that choreographers that control fewer parts of the body have a more specialized purpose requiring more explicit control in the final generated pose. A choreographer for locomotion might swing the character's arms as the model walks, but the target constraints of a reach choreographer are priortized by placing it later in the dataflow graph.

For a choreographer C, we compute: (1) its maximum depth C_d, defined as the maximal distance between any joint affected and the root (hips) of the model and, (2) its coverage C_v which is the number of joints affected. The specificity of a choreographer $S_C = \frac{C_d}{C_v}$ is lower for choreographers that affect a high number of joints close to the hips (torso), and higher for choreographers that affect a small number of joints farther away from the root, such as by controlling the extremities. Sorting the choreographers by ascending order of specificity prioritizes more specialized choreographers, giving them more exclusive access to the joints they focus on controlling. Figure 4 illustrates the depth and coverage of four choregraphers, with the automatically generated pose dataflow graph shown in Figure 5 (arranging the choreographers left-to-right by ascending specificity).

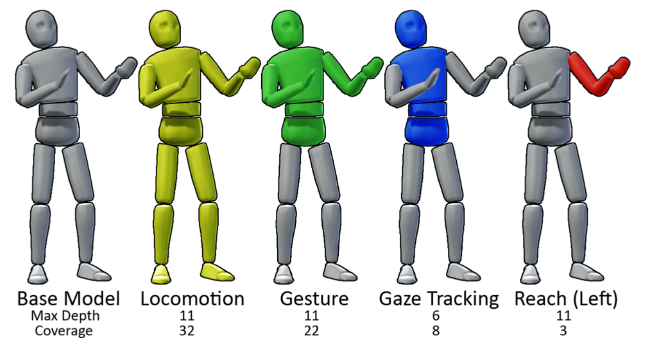


Fig. 4. Coverage analysis of four choreographers on a model with 32 total joints.

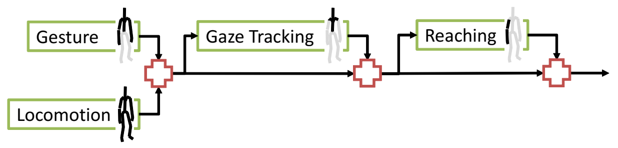


Fig. 5. The automatically generated pose dataflow graph from a four-choreographer coverage analysis.

This is a simple approach that may not apply to every combination of choreographers, but provides a useful first-pass analysis to offer a suggestion to an author designing a new coordinator for a set of unfamiliar choreographers. By design, choreographers are already aware of the joints they affect on the body, making computing C_d and C_v possible without any modification to the ADAPT system. However, other heuristics with additional information from the choreographers could be designed to capture specificity or other elements of good dataflow graph design. This automated technique accelerates the dataflow graph authoring process, which occurs infrequently enough to be tweaked by a human author without consuming prohibitive amounts of time.

5 Character Behavior

One of the most fundamental problems in interactive character animation is converting simple commands like "reach for that object", or cooperative directives like "engage in a conversation" into a series of complicated joint actuations on one or more articulated bodies. ADAPT accomplishes this task with a hierarchy of abstractions known as the *ADAPT Character Stack*, illustrated in Figure 6. The stack is split into four main tiers: Behavior, Actor, Body, and Animation. The higher two levels ("Behavior" and "Actor") of the stack are designed for use by comparitively untrained authors, while lower levels levels offer more fine-grain control fidelity at the expense of simplicity, and can be accessed by expert authors to ensure very specific constraints on the character's movements.

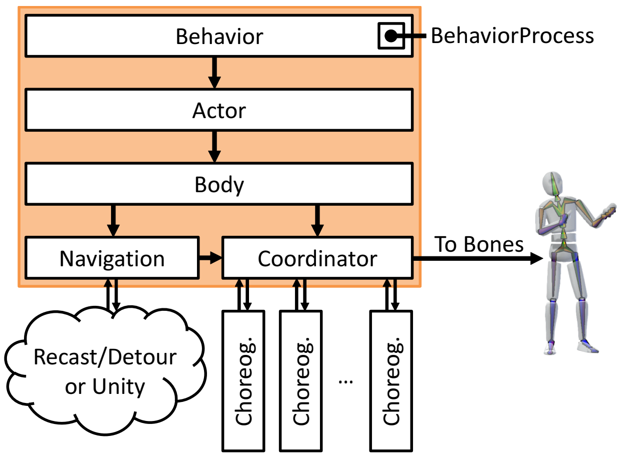


Fig. 6. The ADAPT Character Stack.

Commands from each layer of the stack are filtered, converted, and distributed to subcomponents, starting as behavior invocations, translating to messages sent to the navigation or animation system, and finally converting into joint angles and blend weights used for posing the character on a frame-by-frame basis. Each layer in the character stack provides a different entry point for technical control over the character. The "Behavioral" layers offer interfaces for controlling the character at a high level, suitable for invocation by behavior trees and Smart Objects [57].

Animation (Navigation and Coordinator). This layer provides the lowest-level external access to the character's animation. A component accessing this part of the character stack is concerned with sending messages directly to choreographers (such as to change the reaching target position), or modifying blend weights to adjust the influence of a choreographer.

Body. This layer converts abstract commands like `ReachFor()` into a series of messages passed to the reach choreographer and coordinator to set the reach target and raise the blend weight for the reaching pose. The Body layer is created to encapsulate the pose dataflow graph for a particular character, and assigns more semantic meaning to the blend weights for each blend node in the graph. An ADAPT character's list of capabilities is discussed in more detail in Section 5.1.

Actor. This layer abstracts commands in the Body layer. However, unlike the Body layer, the commands in the Actor layer will keep track of the duration of a task, and report success or failure. A call to `ReachFor()` in the Body layer will return instantly and begin the reaching process, whereas a call to `ReachFor()` in the Actor layer will begin the reach process and then block until the reach has succeeded or failed. Commands in the Actor layer are also designed to respond to a termination signal for scheduling, as described in Section 6.1.1. This layer of abstraction is necessary for controlling a character's behavior with behavior trees.

Behavior. This layer contains more sophisticated, contextual commands comprising multiple sequential calls to the Actor layer, such as playing a series of gestures to convey approval in a conversation. The Behavior layer also contains the character's personal behavior tree, and a BehaviorProcess node responsible for scheduling multi-character actions using the ADAPT behavior scheduler described in Section 6.1.1. Unless involved in a multi-actor event, the Behavior layer is responsible for directing the character's goals, and external calls to the Behavior layer are usually concerned with suspending or re-activating a character's autonomy.

5.1 Body Capabilities

The navigation and shadow-based character animation system provides a number of capabilities, enumerated below. 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.

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.

Adding a New Body 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. 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, and/or pass messages to that choreographer by relaying them through the coordinator. The sophistication of character choreographers varies, but ADAPT is specifically designed for integrating new choreographers into the character behavior and animation pipeline.

6 Character Interactions

Using a character's body capability 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.

6.1 Characters Interacting with Each Other

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 participants. The 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 are also designed using PBTs. 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.

Figure 7 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

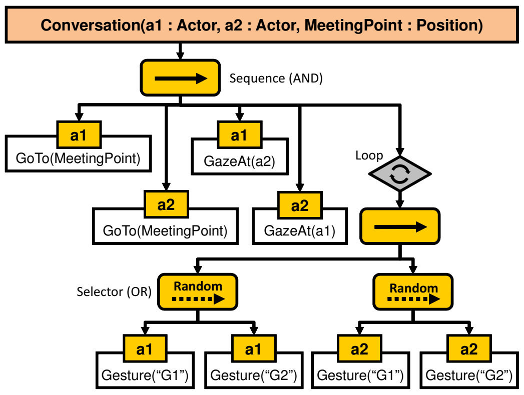


Fig. 7. A simple conversation PBT controlling two characters, `a1` and `a1`, with a `MeetingPoint` parameter.

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.

6.1.1 *The ADAPT Behavior Scheduler*

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. Four basic principles in behavior design enable the scheduler to work effectively for orchestrating the behavior in an ADAPT environment.

PBT Clock. PBTs in ADAPT operate on periodic clock ticks, where each tree refreshes itself, evaluates its current state, sends messages through the character stack, and transitions to its next node if necessary. The ADAPT scheduler keeps track of all of the active trees in the environment, both personal trees for individual characters, and event trees for multi-character interactions, and ticks them 30 times per second.

Character Suspension. Each character in ADAPT owns a "BehaviorProcess" object in its behavior layer, which maintains that character's state with regards to autonomy. A character will not receive ticks to its personal tree if it has been suspended and placed under the control of a multi-character event tree.

Behavior Termination. PBTs in ADAPT are designed to respond to a termination signal. Termination signals can come at any time, and instruct a tree to interrupt its current action and end. The result of a termination signal can last multiple PBT clock ticks, so that poses such as reaching can be smoothly faded out before the tree reports that it has completed termination.

Event Priority. Multi-character event trees are assigned a priority value, where all character personal trees have minimal priority. When a character receives an event with a priority higher than its current tree, the scheduler terminates the current tree and suspends the character until all involved characters are ready. The event then begins ticking and dispatching commands to them. Once the event terminates, the characters return from being suspended.

These four concepts allow very direct control over groups of characters in the environment, with smooth transitions between drastically different tasks. Since trees can be cleanly terminated at any point in their execution, groups of characters involved in wandering or conversing with one another in an environment could very quickly activate a new, higher priority event tree to respond quickly to an event such as a loud noise or a fire. Each tree operates as its own cooperatively multithreaded process with no direct inter-communication. Since a character is only ever controlled by one tree at a time, this presents an opportunity for parallelization in future work.

6.2 Characters Interacting with the Environment

Using the same four principles for the behavior scheduler allows us to more easily implement smart objects into ADAPT to allow characters to interact with the environment. A smart object's affordances can be encoded sequentially in a manner similar to that of a behavior tree. Smart objects receive ticks from the scheduler clock, and will block until reporting either success or failure. For example, when a character wants to sit in a chair, the behavior tree invokes the chair's "sit" affordance with the character as a parameter. From that point, the tree will divert any ticks it receives from the scheduler clock to the smart object, which temporarily takes control of the character and directs it to approach and sit properly on the chair. Once the smart object chair determines that the character has succeeded in sitting down, it will report success to the behavior tree responsible for the character's behavior, so that the tree can move on to other actions. Parallel nodes in the tree can be used to synchronize actions, allowing a character to gesture or gaze at a target while still approaching and sitting. Smart objects represent the primary means of interaction with the environment, and are useful for a wide array of other interaction tasks.

Figure 8 illustrates a simple state machine example for a chair smart object that instructs a character to approach and sit on it in the proper orientation.

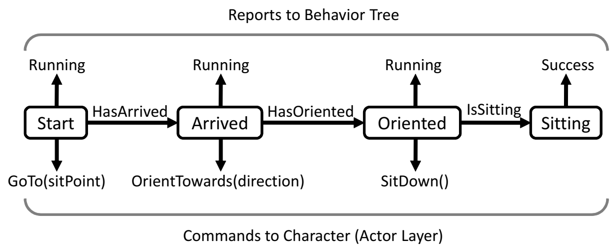


Fig. 8. The state machine for a chair smart object's "sit" affordance. The object transitions between states after evaluating predicates on the state of the character, periodically sending commands to the actor and reporting a status (running, success, or failed) to the behavior tree controlling the affordance.

7 Results

We demonstrate the ADAPT features in isolation, as well as a final scene showcasing animation, navigation, and behavior working together to produce a narrative sequence (Figure 1). 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 and dynamic obstacles. These features are used for authoring sequences like exchanging objects between actors, wandering while talking on a phone, and multiple characters holding a conversation.

7.1 Choreographer Extensibility

Adding a Kinect Choreographer. We created an additional choreographer to interface with the Microsoft Kinect and control a character with gesture input. We allocated a 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 9(a). 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 allows 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 took minimal effort.

Jump Choreographer. Figure 9(b) illustrates the integration of a simple jump choreographer which uses a pre-recorded motion clip that is blended onto the final display model when a jump is triggered. Jump motion is applied to the whole body and supersedes the locomotion choreographer when active. The root motion during a jump is procedurally computed to make the jump choreographer robust to varied environment configurations with differing jump distances.

Locomotion with Footstep Constraints. The previously described locomotion controller precomputes parameters such as root displacement and speed for motions in the database which is used for blending between motions to follow a given velocity command. In order to facilitate control policies with footstep-level precision [33], we extend the parameter space to include the relative transform of the current support foot based on the work in [58]. To offset the problem of insufficient coverage in the motion database, we use IK on the legs to enforce foot constraints. This allows our animated characters to naturally place their feet to solve particularly challenging locomotion scenarios as illustrated in Figure 9(d).

7.2 Computational Performance

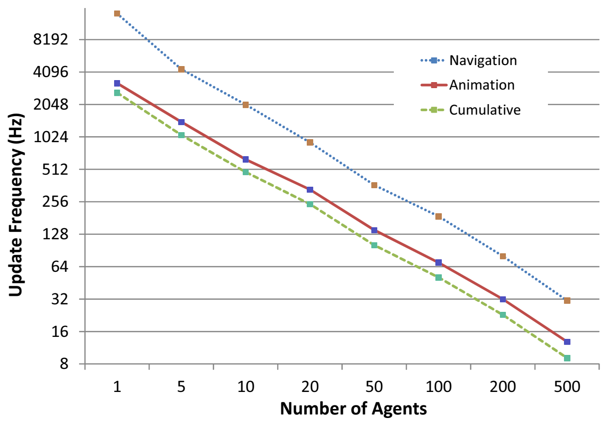


Fig. 10. Update frequency for the character animation and navigation components in ADAPT.

ADAPT supports approximately 150 agents with full fidelity at interactive frame rates. Figure 10 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,

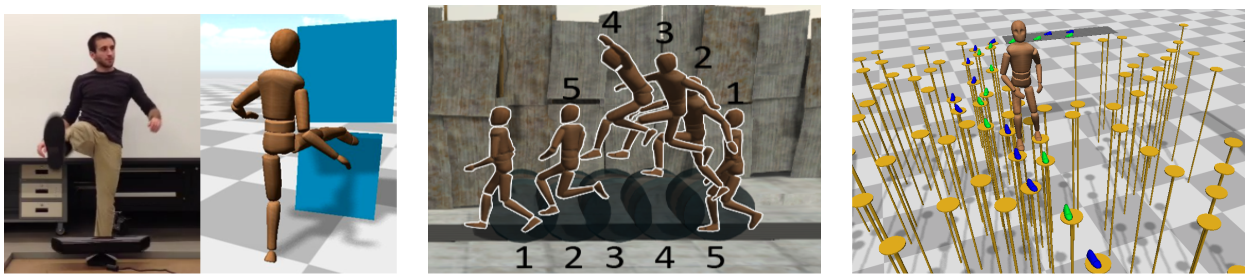


Fig. 9. (a) Controlling a character in ADAPT and physically interacting with the environment using the Kinect. (b) A time-lapse view of the jump choreographer avoiding a moving obstacle. (c) Solving a challenging locomotion problem with footstep-level precision.

and the blend operation is linear in number 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.

7.3 Multi-Actor Simulations

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. Trigger management is handled by the ADAPT Behavior Scheduler described in Section 6.1.1. For our demo, most triggers were scheduled temporally, and can be replaced by a more dynamic event dispatcher. 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 7. The conversation is interrupted when a ball is thrown at one of the characters. The culprit flees from the characters, 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 Appendix B (supplemental).

7.4 Open Source Framework

We have released the software framework as an open source library. The platform has seen interest from multiple academic and industrial institutions, and is already being used for a wide variety of applications including development of narrative-driven interactive virtual worlds, enabling sound perception in virtual humans, the use of machine learning for crowd simulation, high-fidelity character animation, multi-character navigation, simulating culturally accurate virtual populaces, and the development of serious games for education and data collection. The platform can be downloaded as a Unity package at `http://cg.cis.upenn.edu/ADAPT/`.

8 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. Our framework enables, for the first time, seamless integration of diverse, modular, and easily extensible controllers for animating autonomous virtual humans in complex 3D environments. This system can be used to orchestrate sophisticated character movements as well as complex multi-actor interactions for narrative-driven interactive virtual worlds using a graphical behavior authoring paradigm. All of this is enabled by a simple, powerful, and accessible core platform.

Limitations and Complications. Choreographers may sometimes 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.

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 can be resolved by using better blending schemes or by using a more robust locomotion system, as described in Section 7.1.

Future Work. Moving forward, we will continue to expand the animation and authoring capabilities supported by ADAPT. In addition to choreographers described here, we want our platform to provide an array of options for different kinds of motor skills, including climbing, and carrying objects with weight, as well as provide multi-modal sensory capabilities such as agent hearing. 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 have developed 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.

References

[1] N. Pelechano, J. M. Allbeck, and N. I. Badler, *Virtual Crowds: Methods, Simulation, and Control*, ser. Synthesis Lectures on Computer Graphics and Animation, 2008.

[2] D. Thalmann and S. R. Musse, *Crowd Simulation, Second Edition*. Springer, 2013.

[3] M. Kapadia and N. I. Badler, "Navigation and steering for autonomous virtual humans," *Wiley Interdisciplinary Reviews: Cognitive Science*, 2013.

[4] M. Kapadia, A. Shoulson, C. D. Boatright, P. Huang, F. Durupinar, and N. I. Badler, "What's next? the new era of autonomous virtual humans," in *MIG*, 2012, pp. 170–181.

[5] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," ser. SIGGRAPH, 2002, pp. 473–482.

[6] O. Arikan and D. A. Forsyth, "Interactive motion generation from examples," *ACM TOG*, vol. 21, no. 3, pp. 483–490, July 2002.

[7] A. Witkin and Z. Popovic, "Motion warping," ser. SIGGRAPH, 1995, pp. 105–108.

[8] S. Menardais, F. Multon, R. Kulpa, and B. Arnaldi, "Motion blending for real-time animation while accounting for the environment," ser. CGI, 2004, pp. 156–159.

[9] J. Min and J. Chai, "Motion graphs++: a compact generative model for semantic motion analysis and synthesis," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 153:1–153:12, Nov. 2012.

[10] M. Dontcheva, G. Yngve, and Z. Popović, "Layered acting for character animation," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 409–416, July 2003.

[11] A. Shapiro, M. Kallmann, and P. Faloutsos, "Interactive motion correction and object manipulation," in *Interactive 3D graphics and games*, ser. I3D '07. ACM, 2007, pp. 137–144.

[12] Y. Huang, M. Mahmudi, and M. Kallmann, "Planning humanlike actions in blending spaces," in *Intelligent Robots and Systems (IROS)*, 2011.

[13] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," *ACM TOG*, vol. 21, no. 3, pp. 491–500, 2002.

[14] R. S. Johansen, "Automated semi-procedural animation for character locomotion," Master's thesis, Aarhus University, 2009.

[15] Y. Liu and N. I. Badler, "Real-time reach planning for animated characters using hardware acceleration," ser. CASA, 2003, pp. 86–93.

[16] A. W. Feng, Y. Xu, and A. Shapiro, "An example-based motion synthesis technique for locomotion and object manipulation," ser. I3D, 2012, pp. 95–102.

[17] P. Baerlocher and R. Boulic, “An inverse kinematics architecture enforcing an arbitrary number of strict priority levels,” *Vis. Comput.*, vol. 20, no. 6, pp. 402–417, Aug. 2004.

[18] P. Faloutsos, M. van de Panne, and D. Terzopoulos, “Composable controllers for physics-based character animation,” ser. ACM SIGGRAPH, 2001, pp. 251–260.

[19] K. Yin, K. Loken, and M. van de Panne, “Simbicon: simple biped locomotion control,” *ACM TOG*, vol. 26, no. 3, 2007.

[20] J. Pettré, M. Kallmann, and M. C. Lin, “Motion planning and autonomy for virtual humans,” in *ACM SIGGRAPH classes*, 2008, pp. 42:1–42:31.

[21] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE-RAS*, vol. 12, pp. 566 –580, 1996.

[22] M. Kallmann, “Shortest paths with arbitrary clearance from navigation meshes,” in *Eurographics/SIGGRAPH SCA*, 2010.

[23] C. Warren, “Global path planning using artificial potential fields,” in *IEEE-RAS*, 1989, pp. 316–321 vol.1.

[24] A. Treuille, S. Cooper, and Z. Popović, “Continuum crowds,” in *ACM SIGGRAPH*, ser. SIGGRAPH ’06, 2006, pp. 1160–1168.

[25] C. Reynolds, “Steering behaviors for autonomous characters,” 1999.

[26] D. Helbing and P. Molnar, “Social force model for pedestrian dynamics,” *PHYSICAL REVIEW E*, vol. 51, pp. 42–82, 1995.

[27] N. Pelechano, J. M. Allbeck, and N. I. Badler, “Controlling individual agents in high-density crowd simulation,” in *ACM SIGGRAPH/Eurographics SCA*, 2007, pp. 99–108.

[28] S. Paris, J. Pettr, and S. Donikian, “Pedestrian reactive navigation for crowd simulation: a predictive approach,” *Computer Graphics Forum*, vol. 26, no. 3, pp. 665–674, 2007.

[29] J. van den Berg, M. C. Lin, and D. Manocha, “Reciprocal velocity obstacles for real-time multi-agent navigation,” in *ICRA*. IEEE, 2008, pp. 1928–1935.

[30] M. Kapadia, S. Singh, W. Hewlett, and P. Faloutsos, “Egocentric Affordance Fields in Pedestrian Steering,” in *Interactive 3D graphics and games*, ser. I3D ’09. ACM, 2009, pp. 215–223.

[31] M. Kapadia, S. Singh, W. Hewlett, G. Reinman, and P. Faloutsos, “Parallelized egocentric fields for autonomous navigation,” *The Visual Computer*, pp. 1–19.

[32] S. Singh, M. Kapadia, B. Hewlett, G. Reinman, and P. Faloutsos, “A modular framework for adaptive agent-based steering,” in *ACM I3D*, 2011, pp. 141–150.

[33] S. Singh, M. Kapadia, G. Reinman, and P. Faloutsos, “Footstep navigation for dynamic crowds,” *Computer Animation and Virtual Worlds*, vol. 22, no. 2-3, pp. 151–158, 2011.

[34] M. Schuerman, S. Singh, M. Kapadia, and P. Faloutsos, “Situation agents: agent-based externalized steering logic,” *Comput. Animat. Virtual Worlds*, vol. 21, pp. 267–276, May 2010.

[35] A. Lerner, Y. Chrysanthou, and D. Lischinski, “Crowds by example,” *CGF*, vol. 26, no. 3, pp. 655–664, 2007.

[36] K. H. Lee, M. G. Choi, Q. Hong, and J. Lee, “Group behavior from video: a data-driven approach to crowd simulation,” in *ACM SIGGRAPH/Eurographics SCA*, 2007, pp. 109–118.

[37] M. Mononen, “Recast/Detour navigation library,” 2009. [Online]. Available: http://code.google.com/p/recastnavigation/

[38] A. B. Loyall, “Believable agents: Building interactive personalities,” Ph.D. dissertation, Carnegie Mellon University, 1997.

[39] K. Perlin and A. Goldberg, “Improv: a system for scripting interactive actors in virtual worlds,” ser. SIGGRAPH, 1996, pp. 205–216.

[40] A. Shoulson, F. Garcia, M. Jones, R. Mead, and N. I. Badler, “Parameterizing Behavior Trees,” in *Motion in Games*. Springer, 2011, pp. 144–155.

[41] Q. Yu and D. Terzopoulos, “A decision network framework for the behavioral animation of virtual humans,” ser. SCA, 2007, pp. 119–128.

[42] M. Fleischman and D. Roy, “Representing intentions in a cognitive model of language acquisition: Effects of phrase structure on situated verb learning,” in *AAAI ’07*, pp. 7–12.

[43] J. Orkin, “Agent architecture considerations for real-time planning in games,” in *Interactive Digital Entertainment Conference*. AAAI Press, 2005, pp. 105–110.

[44] M. Kapadia, S. Singh, G. Reinman, and P. Faloutsos, “A Behavior-Authoring Framework for Multiactor Simulations,” *IEEE CGA*, vol. 31, no. 6, pp. 45 –55, 2011.

[45] W. Li and J. M. Allbeck, “The virtual apprentice,” in *IVA*, 2012, pp. 15–27.

[46] U. Erra, B. Frola, and V. Scarano, “BehaveRT: a GPU-based library for autonomous characters,” in *Motion in Games*, ser. MIG’10, 2010, pp. 194–205.

[47] C. Stocker, L. Sun, P. Huang, W. Qin, J. M. Allbeck, and N. I. Badler, “Smart events and primed agents,” in *Intelligent virtual agents*, ser. IVA’10. Springer-Verlag, 2010, pp. 15–27.

[48] A. Shoulson and N. I. Badler, “Event-centric control for background agents,” in *International conference on Interactive Digital Storytelling*, ser. ICIDS, 2011, pp. 193–198.

[49] L. Sun, A. Shoulson, P. Huang, N. Nelson, W. Qin, A. Nenkova, and N. I. Badler, “Animating synthetic dyadic conversations with variations based on context and agent attributes,” *Comput. Animat. Virtual Worlds*, vol. 23, no. 1, pp. 17–32, Feb. 2012.

[50] Massive Software Inc., “Massive: Simulating life,” 2010, www.massivesofware.com.

[51] Autodesk, Inc., “Autodesk gameware - artificial intelligence middleware for games,” 2012. [Online]. Available: http://gameware.autodesk.com/

[52] S. Singh, M. Kapadia, P. Faloutsos, and G. Reinman, “An open framework for developing, evaluating, and sharing steering algorithms,” in *MIG*, 2009, pp. 158–169.

[53] A. Shapiro, “Building a character animation system,” ser. MIG, 2011, pp. 98–109.

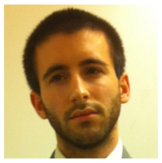
[54] M. Kallmann and S. Marsella, “Lncs ’05,” 2005, ch. Hierarchical motion controllers for real-time autonomous virtual humans, pp. 253–265.

[55] A. W. Feng, Y. Huang, M. Kallmann, and A. Shapiro, “An analysis of motion blending techniques,” in *MIG*, 2012.

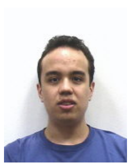
[56] D. Slonneger, M. Croop, J. Cytryn, J. T. K. Jr., R. Rabbitz, E. Halpern, and N. I. Badler, “Human model reaching, grasping, looking and sitting using smart objects international symposium on digital human modeling,” ser. Proc. International Ergonomic Association Digital Human Modeling, 2011.

[57] M. Kallmann and D. Thalmann, “Direct 3d interaction with smart objects,” in *ACM symposium on Virtual reality software and technology*, ser. VRST ’99, 1999, pp. 124–130.

[58] B. J. H. van Basten, P. W. A. M. Peeters, and A. Egges, “The step space: example-based footprint-driven motion synthesis,” *CAVW*, vol. 21, no. 34, pp. 433–441, May 2010.



Alexander Shoulson is Ph.D. student at the University of Pennsylvania, supervised by Norman I. Badler. His research focuses on interactive narrative, character animation, and behavior authoring for virtual humans.



Nathan Marshak is a Master’s student in Computer Graphics and Game Technology at the University of Pennsylvania. He graduated with a BSE in Computer Science from the University of Pennsylvania in 2013. His current interests include animation, control, and shape deformation.

Mubbasir Kapadia is an Associate Research Scientist at Disney Research Zurich. His research focuses on the simulation of functional, purposeful autonomous virtual humans. Kapadia has a PhD in Computer Science from the University of California, Los Angeles.

Norman I. Badler is the Rachleff Professor of Computer and Information Science at the University of Pennsylvania. His research involves developing software for human and group behavior modeling and animation. He is the founding Director of the SIG Center for Computer Graphics and the Center for Human