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) [1] 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:

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

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:

$$\begin{aligned} &\text{Blend}_{SLERP}(q_1, q_2, q_3; , w_1, w_2)\\ &\quad = \text{Slerp}(\text{Slerp}(q_1, q_2; w_1), q_3; w_2) \end{aligned} \quad (5)$$

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 [2], Johnson [3], and Buss et. al. [4] 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.

APPENDIX B: EXAMPLE BEHAVIOR TREES

These trees used in our demonstration are as follows:

- *RandomGesture* is a parameterized subtree that takes in a single actor and instructs that actor to play a random gesture. Trees like these can use different categories of gestures, such as greeting gestures or gestures with different emotions.
- After sending a navigation command, the *Wait-Arrive* tree will wait for an actor to arrive at its destination by blocking the tree until arrival occurs. This is a convenient and reusable subtree for a number of similar functions.

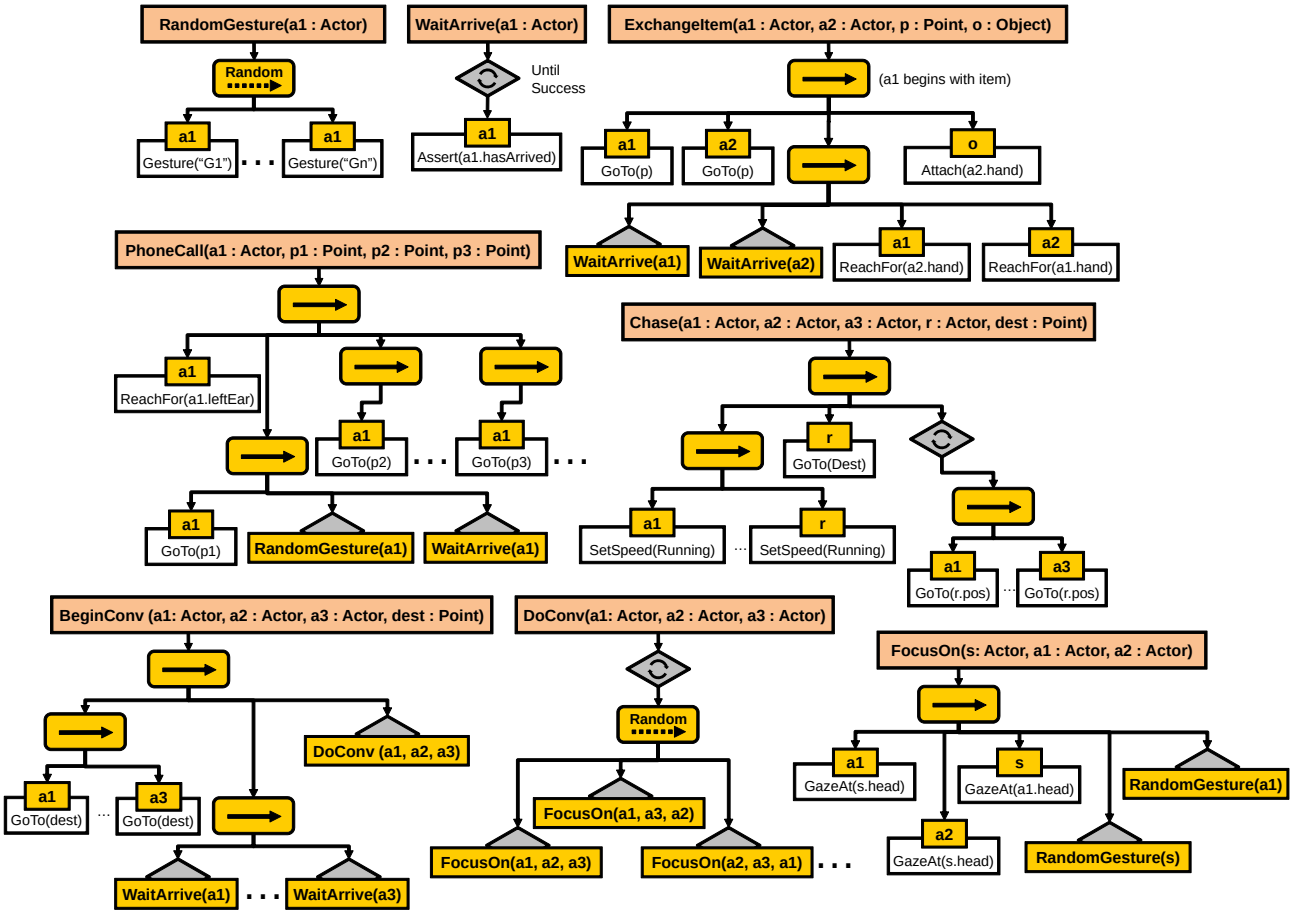

Fig. 1. Example behavior trees used for the ADAPT demo.

- The *ExchangeItem* tree instructs two actors to approach a single point and exchange an item (that actor a1 begins by carrying). Note that the actual tree used in our demo included time offsets for more natural motion, and instructed the agents to look at one another and at each other's hands as the item was exchanged. The WaitArrive tree is invoked as a subtree node, passing the appropriate parameters from the parent tree to the child.
- Once the protagonist is given the phone, the *PhoneCall* event tree executes instructing him to move through three waypoints and play random gestures. The actual implemented tree includes a short sequence at the end for gazing at the runner in the distance.
- The reusable *FocusOn* tree, designed for three characters, designates one character as the speaker. The two other characters gaze at the speaker, while the speaker chooses one to gaze at in return. The speaker gestures, and then the character the speaker is addressing makes a response gesture.
- A conversation is simply divided into a series of FocusOn subtrees with different rearrangements of the parameters. This is handled by the *DoConv* event tree, which randomizes the parameter permutations.
- The *BeginConv* tree occurs at the beginning of a conversation, and handles the characters approaching a central point before conversing.
- The chase scene is accomplished with the *Chase* event tree. The characters increase their movement speed, then the runner departs for a given destination, and the chasing characters set their navigation target to the runner's position.

REFERENCES

[1] K. Shoemake, "Animating rotation with quaternion curves," *ACM SIGGRAPH*, vol. 19, no. 3, pp. 245–254, July 1985.
[2] X. Pennec, "Computing the Mean of Geometric Features Application to the Mean Rotation," INRIA, Tech. Rep. RR-3371, 1998.
[3] M. P. Johnson, "Exploiting quaternions to support expressive interactive character motion," Ph.D. dissertation, MIT, 2003.
[4] S. R. Buss and J. P. Fillmore, "Spherical averages and applications to spherical splines and interpolation," *ACM Trans. Graph.*, vol. 20, no. 2, pp. 95–126, Apr. 2001.