ANIMACIÓN POR ORDENADOR

Tema 7

Animación para videojuegos.
Mezcla de movimientos.
Máquinas de estados.

CONTENTS

1. Animation in videogames: *Blending & Sequencing*
2. Animation playback
3. Animation blending
4. State machines
5. Character controller

BLENDING & SEQUENCING

- Now that we understand how a character rig works and how to manipulate animation data, we can edit and play back simple animation

- The subject of blending and sequencing encompasses a higher level of animation playback, involving constructing the final pose out of a combination of various inputs

BLENDING & SEQUENCING

- Most areas of computer animation have been pioneered by the research and special effects industries
- Blending and sequencing, however, is one area where video games have made a lot of real progress in this area towards achieving interactively controllable and AI characters in complex environments…
- The special effects industry is using some game related technology more and more (battle scenes in Lord of the Rings…)

ANIMATION PLAYBACK

POSES

- A pose is an array of values that maps to a rig
- If the rig contains only simple independent DOFs, the pose can just be an array of floats
- If the rig contains quaternions or other complex coupled DOFs, they may require special handling by higher level code
- Therefore, for generality, we will assume that a pose contains both an array of M≥0 floats and an additional array of N≥0 quaternions

$$\Phi = \left[\phi_0 \ldots \phi_{M-1} \quad \mathbf{q}_0 \ldots \mathbf{q}_{N-1}\right]$$

ANIMATION CLIP

- *AnimationClip* stores an array of channels for a particular animation (or it could store the data as an array of poses…)
- This should be treated as constant data, especially in situations where multiple animating characters may simultaneously need to access the animation (at different time values)
- For playback, animation is accessed as a pose. Evaluation requires looping through each channel.

```
class AnimationClip {
  void Evaluate(float time,Pose &p);
}
```

ANIMATION PLAYER

- We need something that ‘plays’ an animation. We will call it an animation *player*
- At it’s simplest, an animation player would store a AnimationClip*, Rig*, and a float time
- As an active component, it would require some sort of Update() function
- This update would increment the time, evaluate the animation, and then pose the rig

ANIMATION PLAYER

```
class AnimationPlayer {
    float Time;
    AnimationClip *Anim;
    Pose P;
public:
    const Pose &GetPose();
    void Play(AnimationClip &clip);
    void Update();
};
```

ANIMATION PLAYER

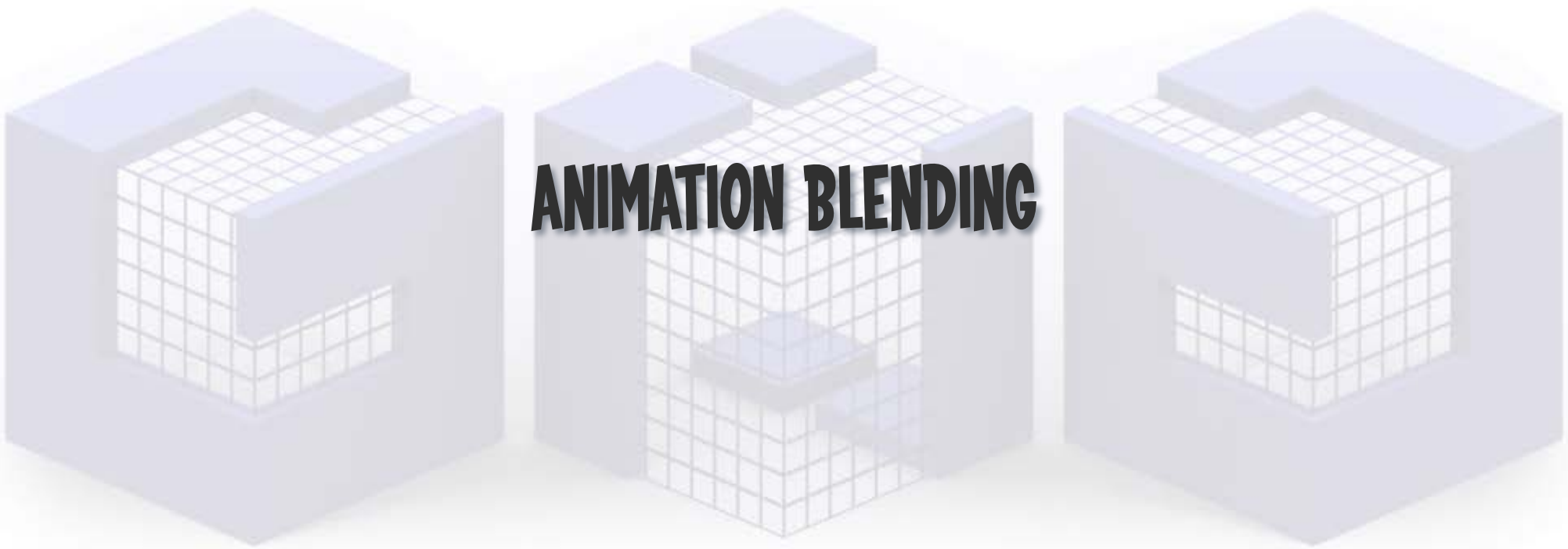
A simple player just needs to increment the Time and access the new pose once per frame

The first question that comes up though, is what to do when it gets to the end of the animation clip?

- Loop back to start
- Hold on last frame
- Deactivate itself… (return 0 pose?)
- Send a message…

ANIMATION PLAYER

- The animation player is a basic component of an animation blending & sequencing system
- Many of these might ultimately be combined to get the final blended pose. This is why we only want it to output a pose
- By the way, remember the issue of sequential access for keyframes? The animation player should ultimately be responsible for tracking the current keyframe array (although the details could be pushed down to a specific class for dealing with that)

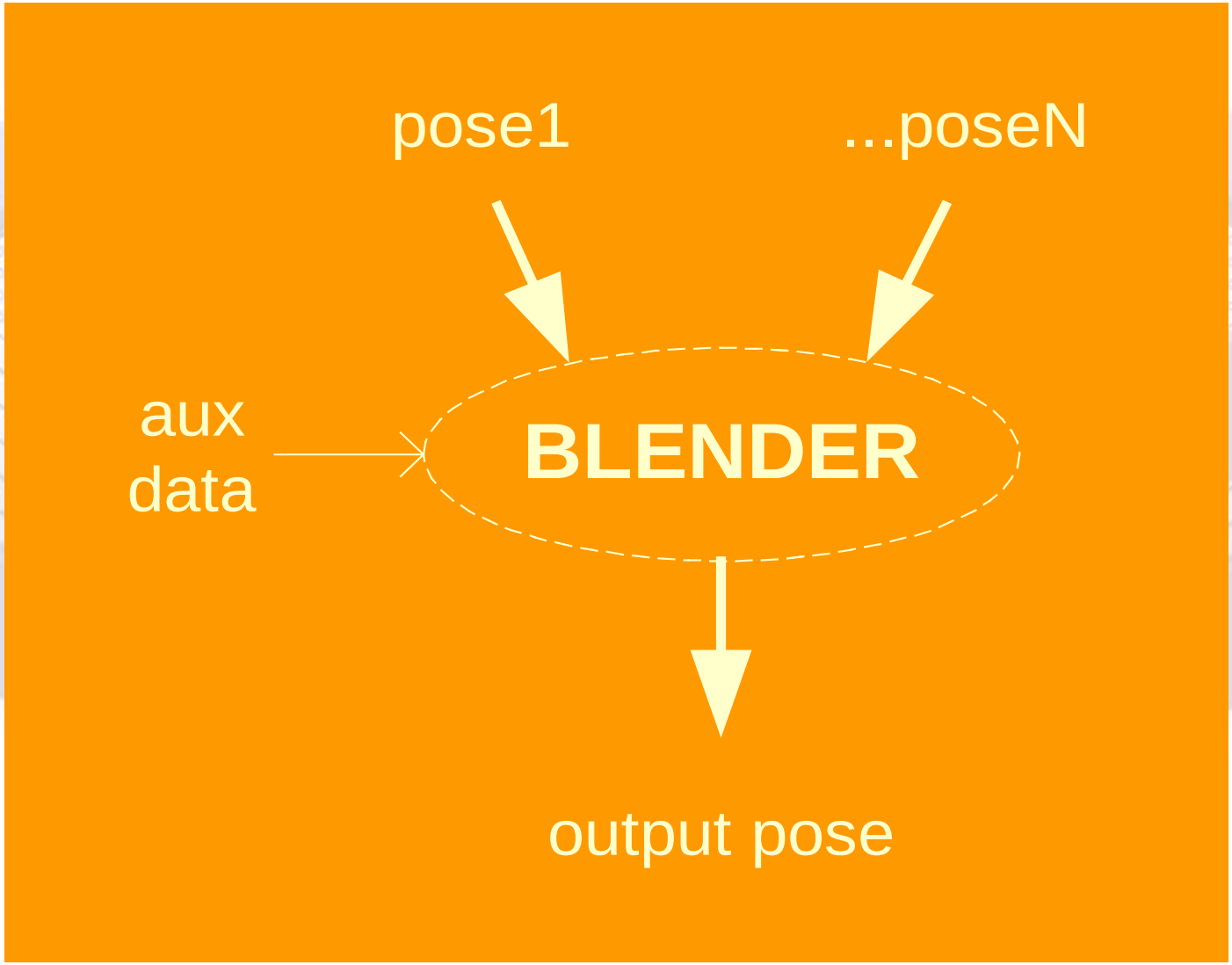


ANIMATION BLENDING

BLENDING OVERVIEW

- We can define blending operations that affect poses
- A blend operation takes one or more poses as input and generates one pose as output
- In addition, it may take some auxiliary data as input (control parameters, etc.)

GENERIC BLEND OPERATION

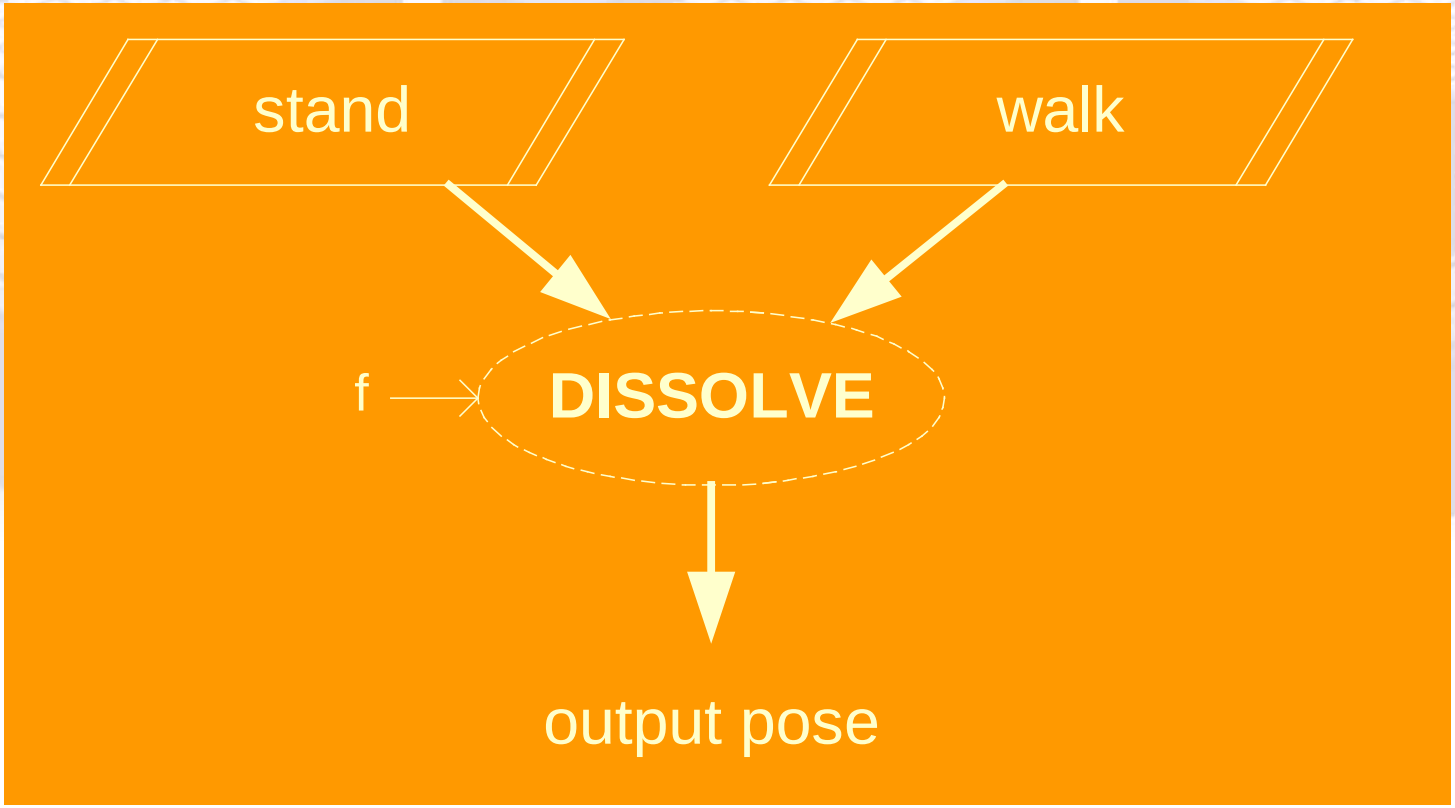


CROSS DISSOLVE

- Perhaps the most common and useful pose blend operation is the ‘cross dissolve’
- Also known as: Lerp (linear interpolation), blend, dissolve…
- The cross dissolve blender takes two poses as input and an additional float as the blend factor (0…1)

CROSS DISSOLVE: STAND TO WALK

Consider a situation where we want a character to blend from a stand animation to a walk animation



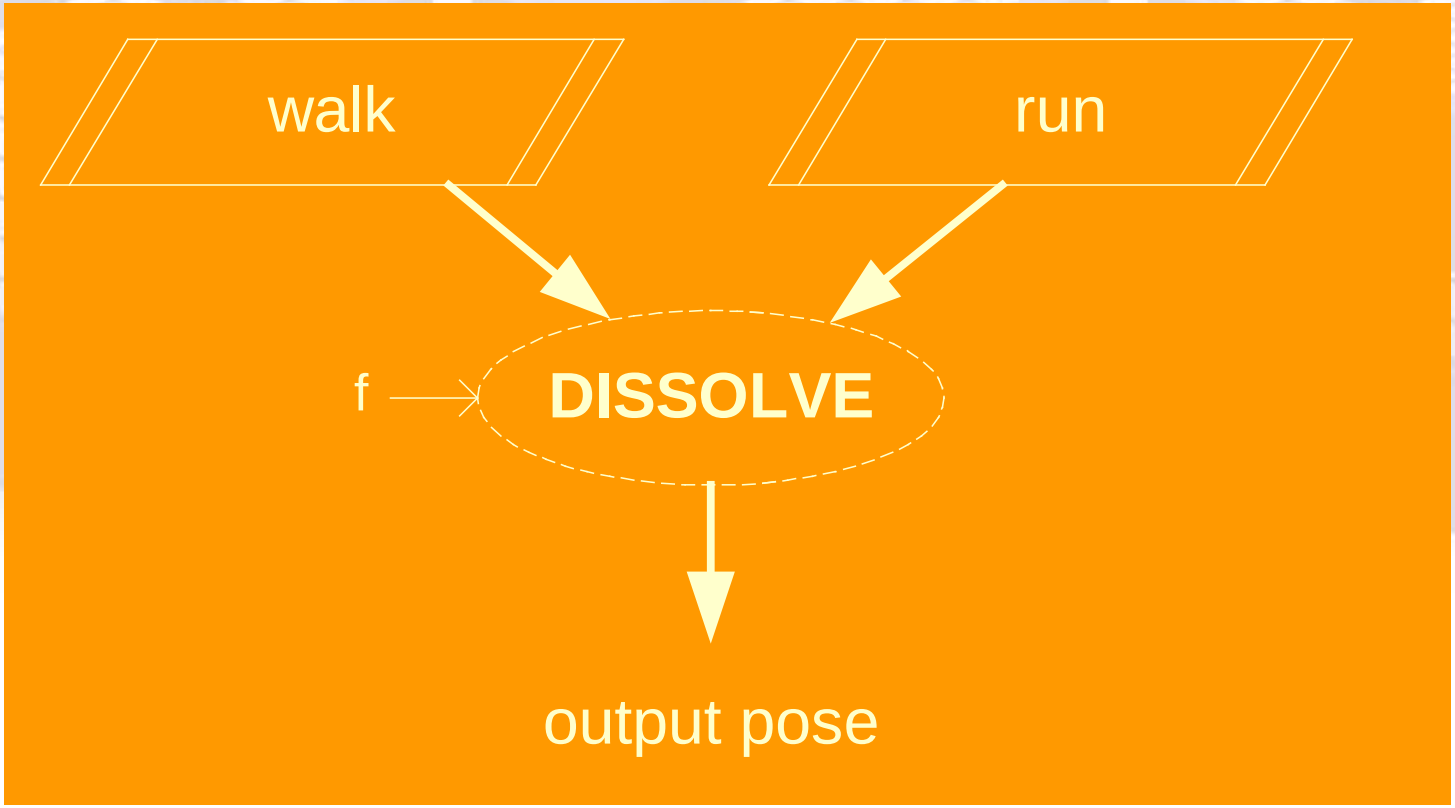
CROSS DISSOLVE: STAND TO WALK

- We could have two independent animations playing (stand & walk) and then gradually ramp the ‘t’ value from 0 to 1
- If the transition is reasonably quick (say <0.5 second), it might look OK

- Note: this is just a simple example of a dissolve and not necessarily the best way to make a character start walking…

CROSS DISOLVE: WALK TO RUN

Blending from a walk to a run requires some additional consideration…



CROSS DISOLVE: WALK TO RUN

- Lets say that we have a walk and a run animation
- Each animation is meant to play as a loop and contains one full gait cycle
- They are set up so the character is essentially moving in place, as on a treadmill
- Let's assume that the duration of the walk animation is d_{walk} seconds and the run is d_{run} seconds
- Let's also assume that the velocity of the walk is v_{walk} and run is v_{run} (these represent the speed that the character is supposed to be moving forward, but keep in mind, the animation itself is in place)

CROSS DISOLVE: WALK TO RUN

- We want to make sure that the walk and run are in phase when we blend between them
- One could animate them in a consistent way so that the two clips both start at the same phase
- But, let's assume they aren't in sync…
- Instead, we'll just store an offset for each clip that marks some synchronization point (say at the time when the left foot hits the ground)
- We'll call these offsets o_{walk} and o_{run}

CROSS DISOLVE: WALK TO RUN

- Let's assume that f is our dissolve factor (0…1) where f=0 implies walking and f=1 implies running
- The resulting velocity that the character should move is simply:

$$v'=Lerp(f,v_{walk},v_{run})$$

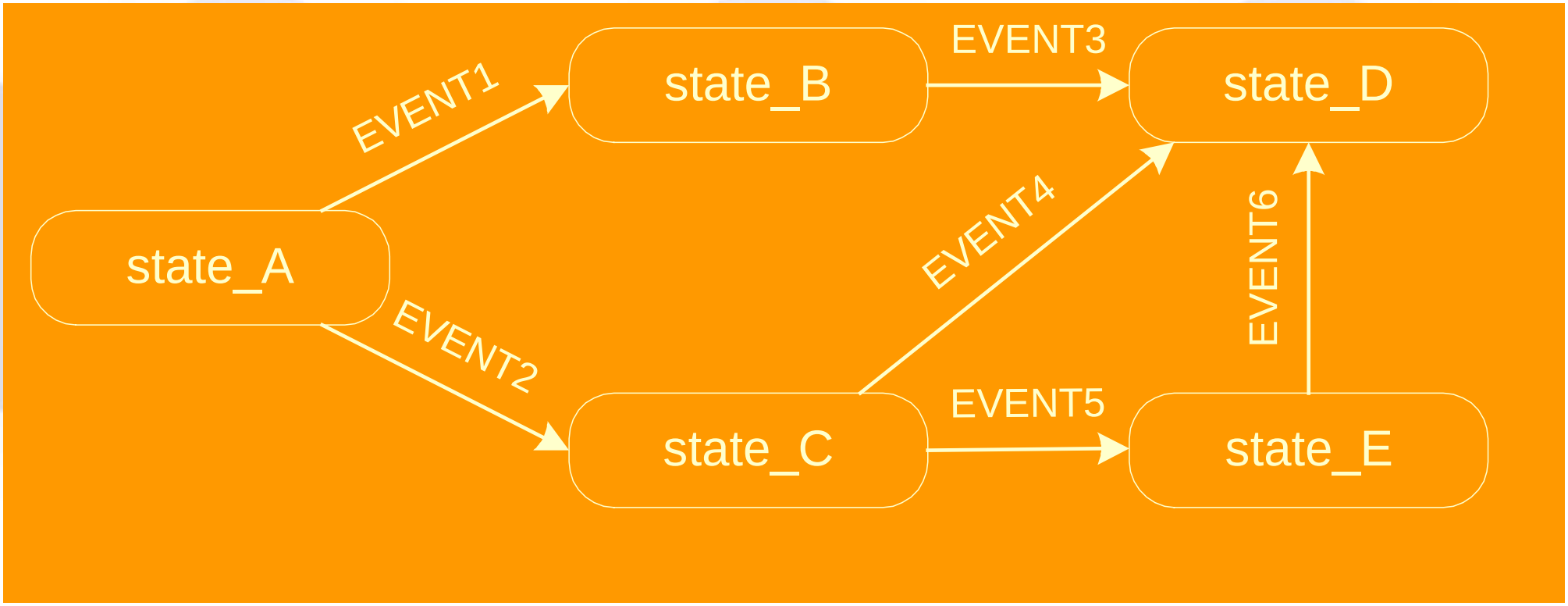
- To get the animations to stay in phase, however, we need to adjust the speeds that they are playing back
- This means that when we're halfway between walk and run, the walk will need to be sped up and the run will need to be slowed down

ANIMATION STATE MACHINES

STATE MACHINES

- Blending is great for combining a few motions, but it does not address the issue of sequencing different animations over time
- For this, we will use a state machine
- We will define the state machine as a connected graph of states and transitions
- At any time, exactly one of the states is the current state
- Transitions are assumed to happen instantaneously

STATE MACHINES



STATE MACHINES

- In the context of animation sequencing, we think of states as representing individual animation clips and transitions being triggered by some sort of event
- An event might come from some internal logic or some external input (button press…)

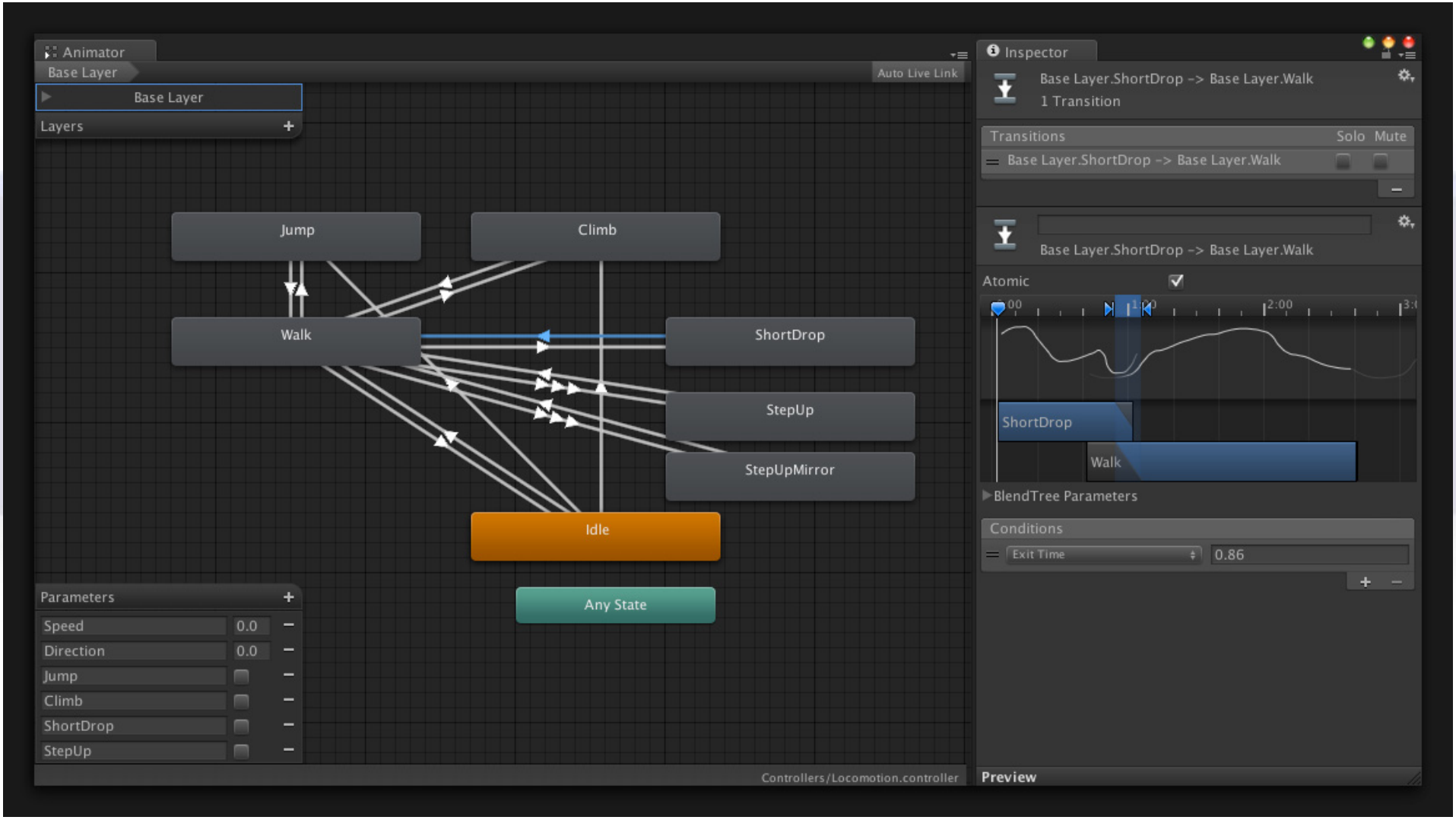
STATE MACHINE EXTENSIONS

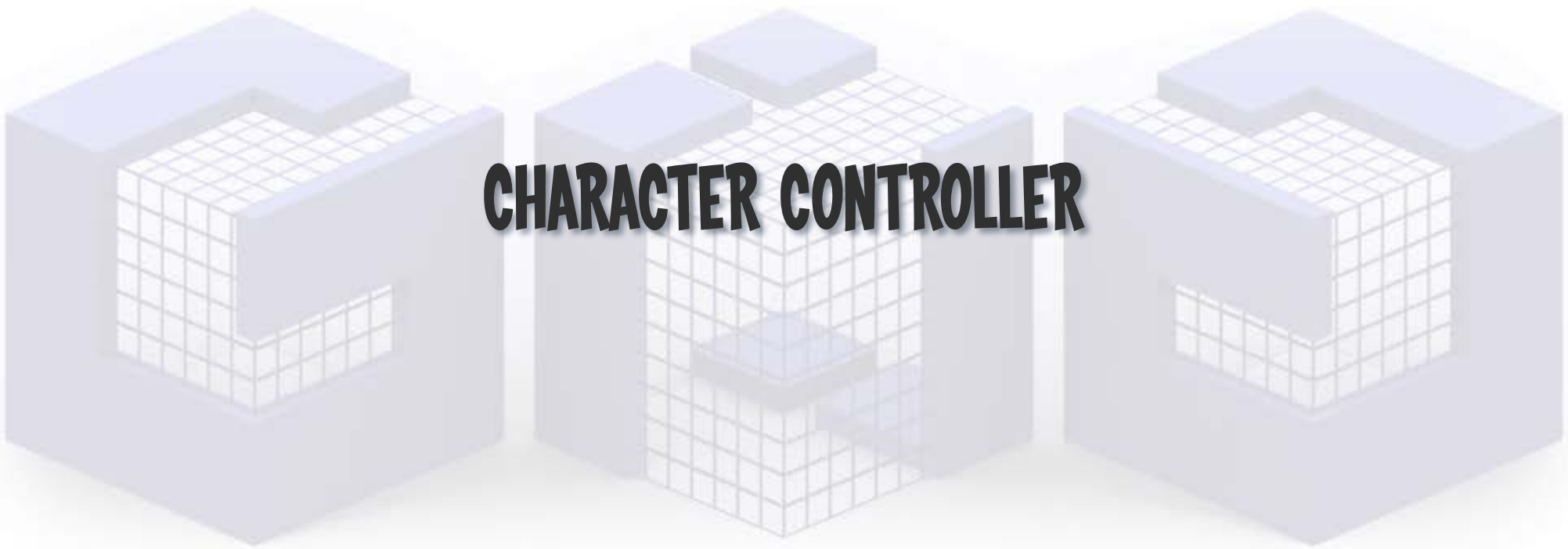
- Global entry transitions
- Event masking
- Fancy states
- Modifiers
- Logic in states
- Combining blenders & state machines
- State machines within state machines

CREATING STATE MACHINES

- Typing in text
- Graphical state machine editor
- Automated state machine generation (motion graphing)

STATE MACHINES IN UNITY 3D





CHARACTER CONTROLLER

CHARACTER CONTROLLER

- When we want an interactive character to move around through a complex environment, we need something to be responsible for the overall placement of the character
- We call this the character *mover*
- We can think of the mover as a matrix that positions the character's root

CHARACTER MOVER

- Usually, we think of the mover matrix as being on the ground right below the character's center
- The mover sits perfectly still when the character isn't moving and generally moves at a smooth constant rate as the character walks
- The character's root translation would be animated relative to the mover

CHARACTER MOVER

- The mover might be coded up to do some simple accelerations, decelerations, turning, and collision detection with the ground and walls
- Depending on the speed that the mover is moving, we might select blend to an appropriate gait animation

CHARACTER MOVER

- Sometimes, we want the character to do more complex moves, such as a dive roll to the right
- In this situation, we might want to explicitly animate what the mover should do
- This data can be written out with the animation and stored as additional channel data (3 translations, 3 rotations)
- These extra channels can be blended like any other channel, and then finally added to the mover when we pose the rig