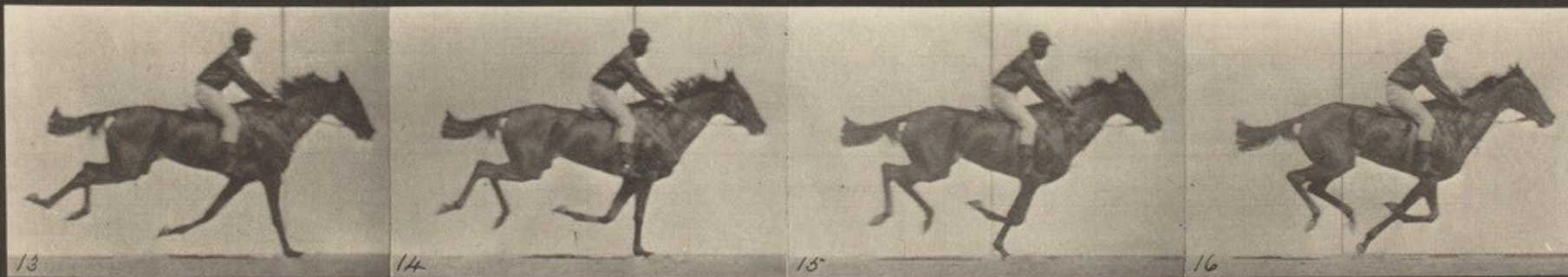
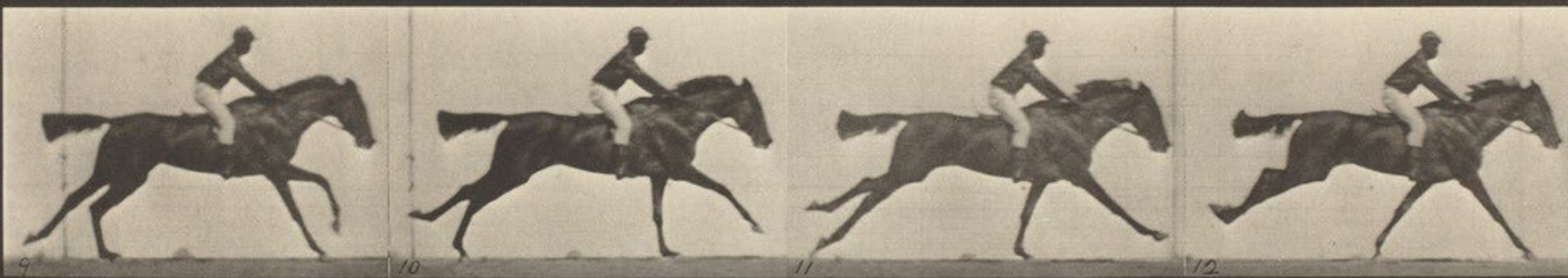
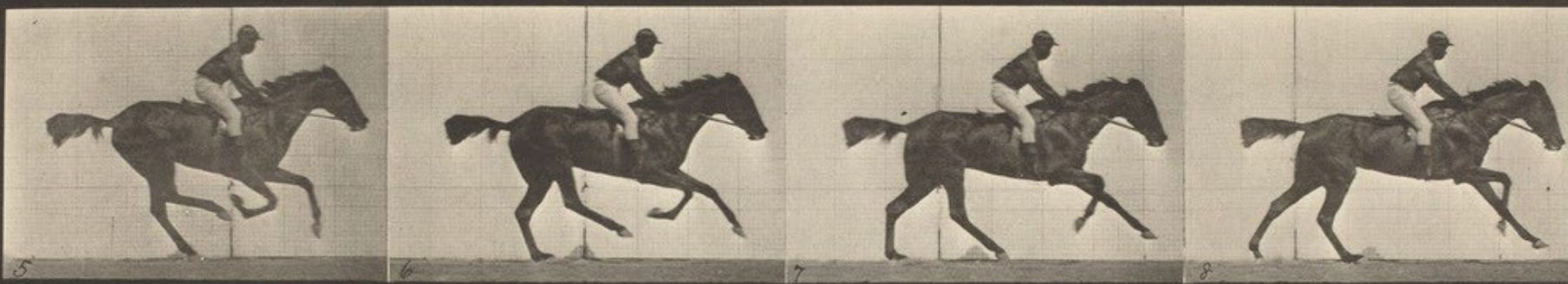
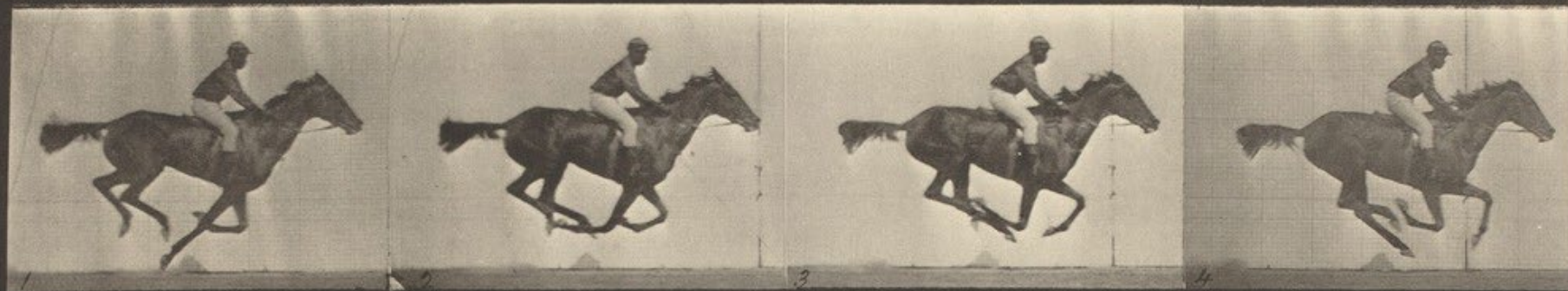
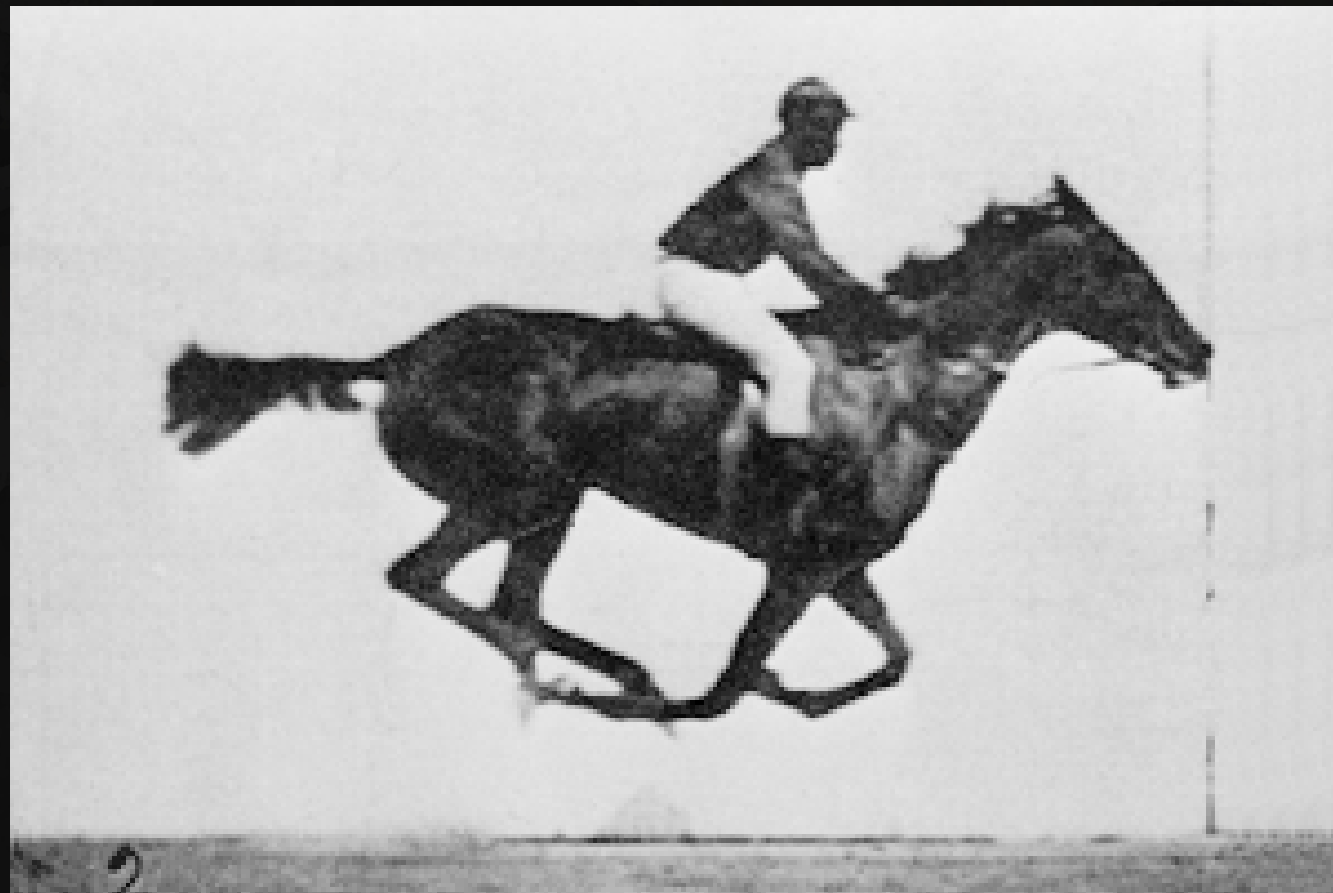


**animation**

# animation

- successive drawings or renderings to create an illusion of **movement** when shown in sequence
- communicating movement
- video games: movement of characters, objects, camera



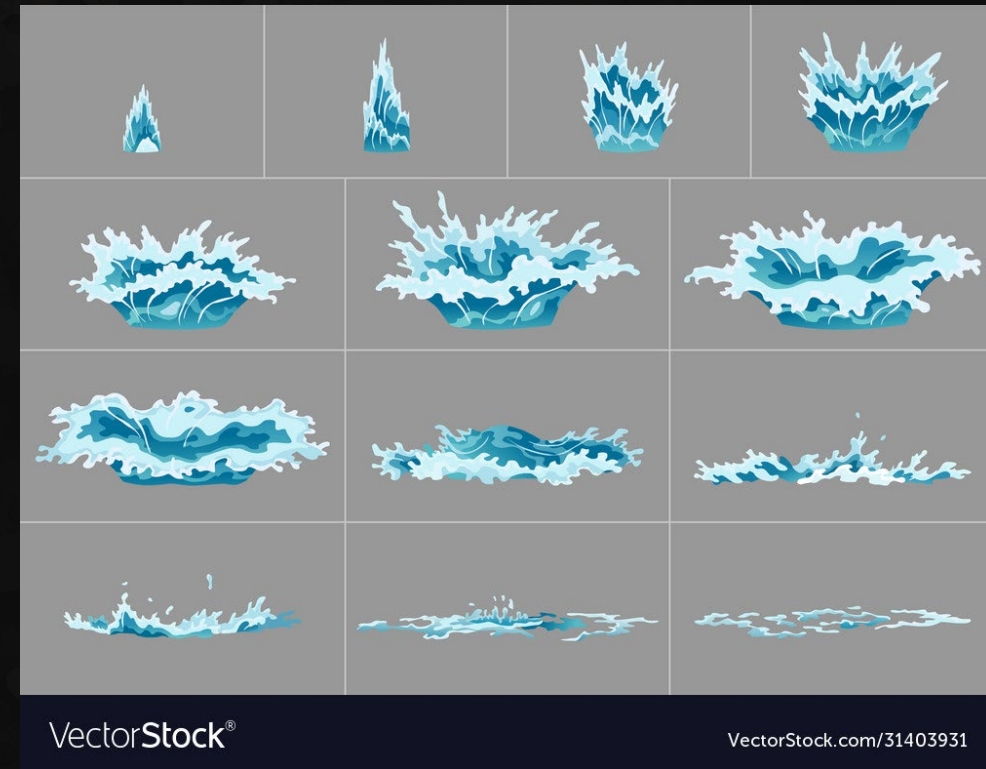


# animation: framerate

- speed at which frames are rendered for an animation
- film: standard 24 fps
- tv: 12 fps, animating on 2s
- video games: 30, 60, 120+ fps
- appear smooth at 60 fps

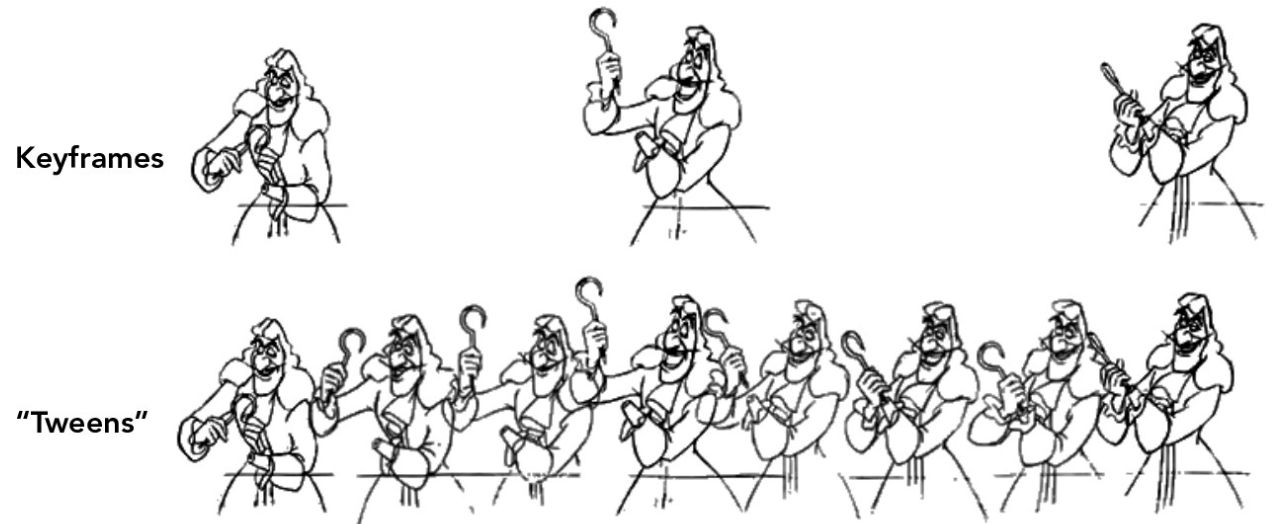
# constructing frames

- representing motion with frames
- **by hand**: drawing or posing
- **interpolation**: generate frames between poses
- **motion capture**: map real world motion to virtual models
- **procedural**: mathematically generated



# key frame animation

- key frames: start, end, important middle steps
- in-between frames constructed with interpolation

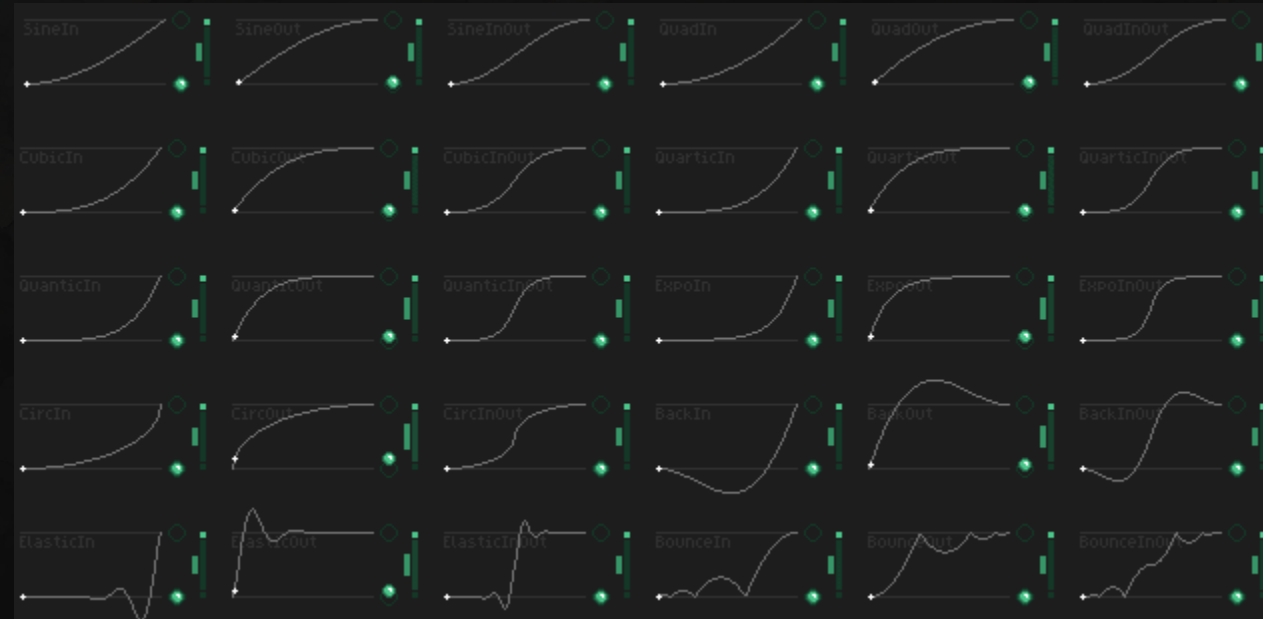


Animator (e.g. lead animator) creates keyframes

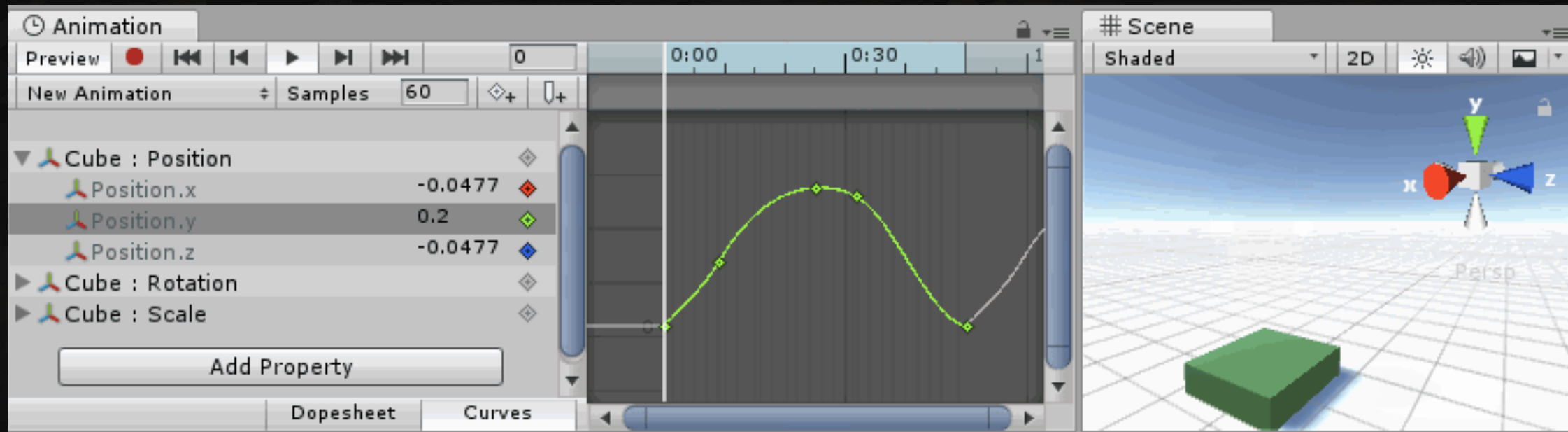
Assistant (person or computer) creates in-between frames ("tweening")

# (computational) tweening

- generating frames in-between key frames
  - adjust numerical values, such as position, scale, rotation
- easing: acceleration / deceleration over time



# unity: tweening with timeline



# unity: tweening in code

- `Vector3.Lerp(Vector3 a, Vector3 b, float t)`
  - linearly interpolates between points a and b by the interpolant t
- `Vector3.Slerp(Vector3 a, Vector3 b, float t)`
  - spherically interpolates between a and b by the amount t

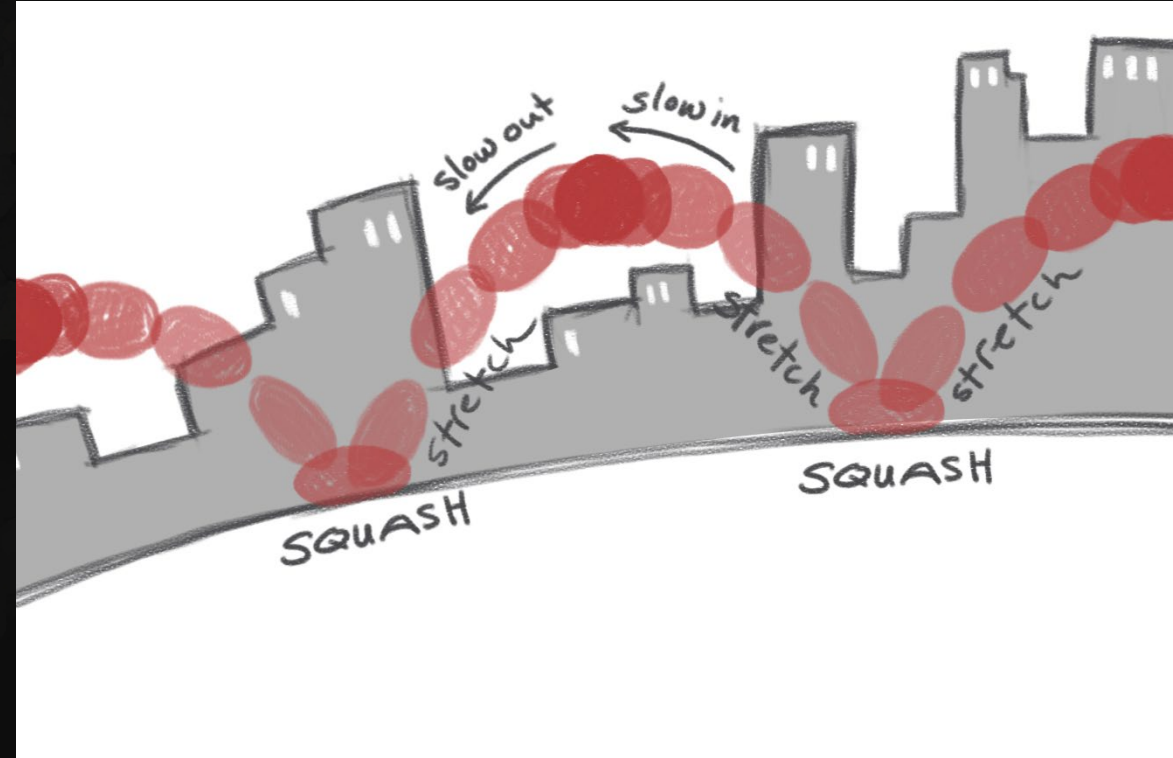
# Disney's 12 principles of animation

Ollie Johnston and Frank Thomas, 1981

1. squash and stretch
2. anticipation
3. staging
4. straight ahead action and pose to pose
5. follow-through and overlapping action
6. slow in and slow out
7. arc motion
8. secondary action
9. timing
10. exaggeration
11. solid drawing
12. appeal

# 1. squash and stretch

- give sense of weight and flexibility
- visualize impact of gravity
- e.g., stretch vertically for jump, squash at start and end
- maintain consistent volume

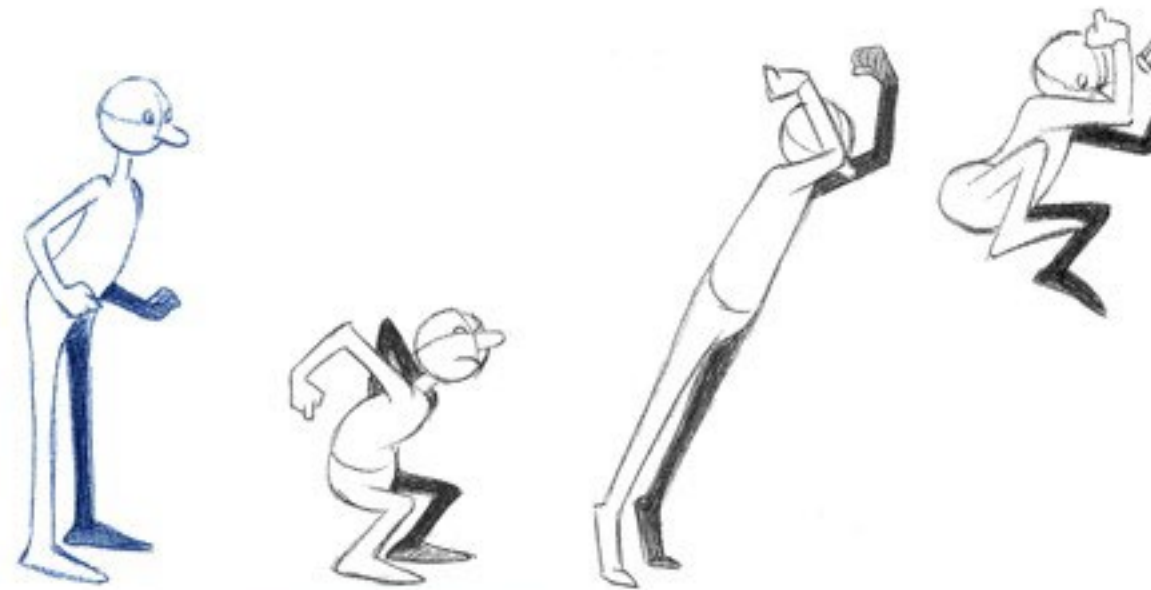


# squash and stretch example



## 2. anticipation

- give clues that a big “action” is about to happen
- e.g., deep breath before the plunge or windup for attack



ANTICIPATION

# anticipation example



# anticipation example



### 3. staging

- every movement, every pose, should convey a clear intention
- direct players attention towards most important element
- e.g., freeze the frame on character should tell how they are feeling



# staging example



## 4. straight ahead action and pose to pose

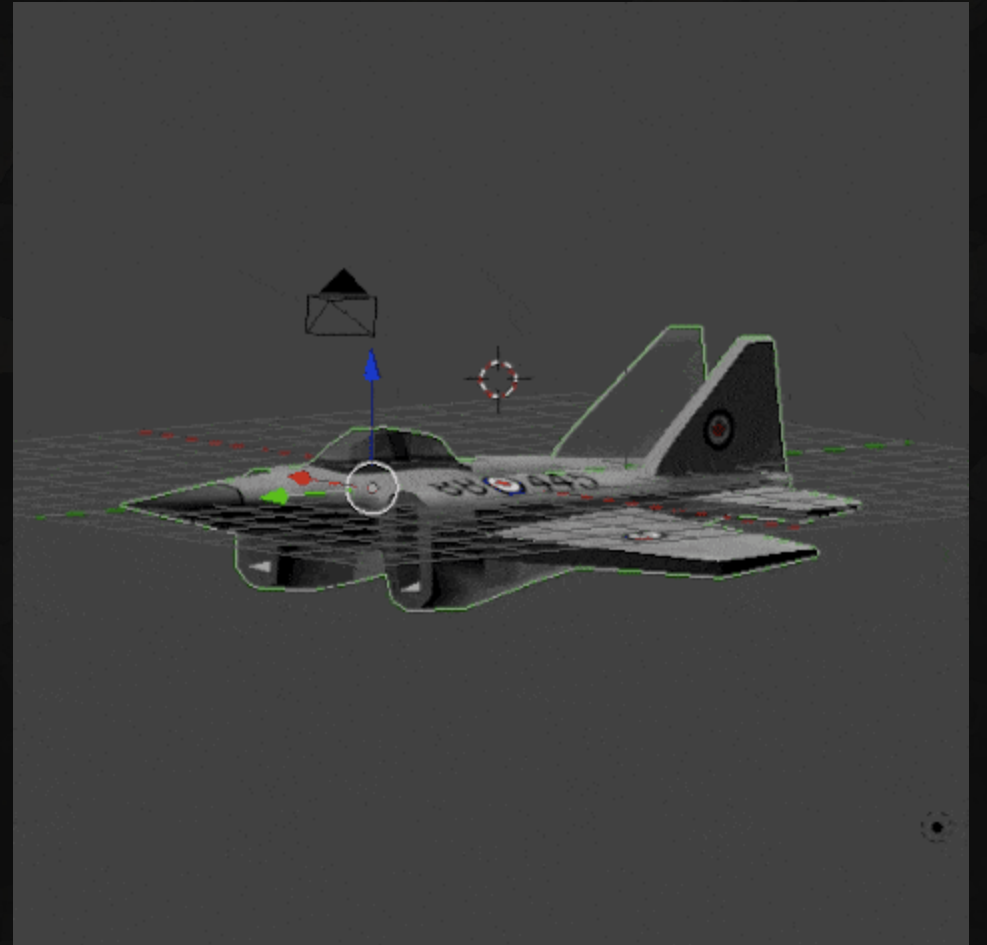
- straight ahead: create frame-by-frame from start to finish
  - fluid, realistic movements
- pose to pose: tweening
  - create begin and end frame and a few frames in between
  - increase dramatic effect of the motion

# straight ahead action example



# pose to pose example

- key frames = poses



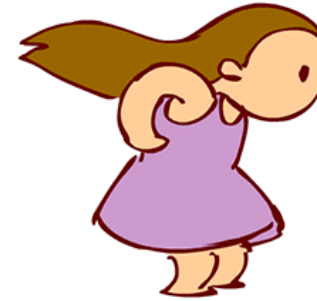
## 5. follow-through and overlapping action

- not everything comes to a stop at the same time
  - e.g., hair, hanging objects, weapons continue slight movements after character stops

with



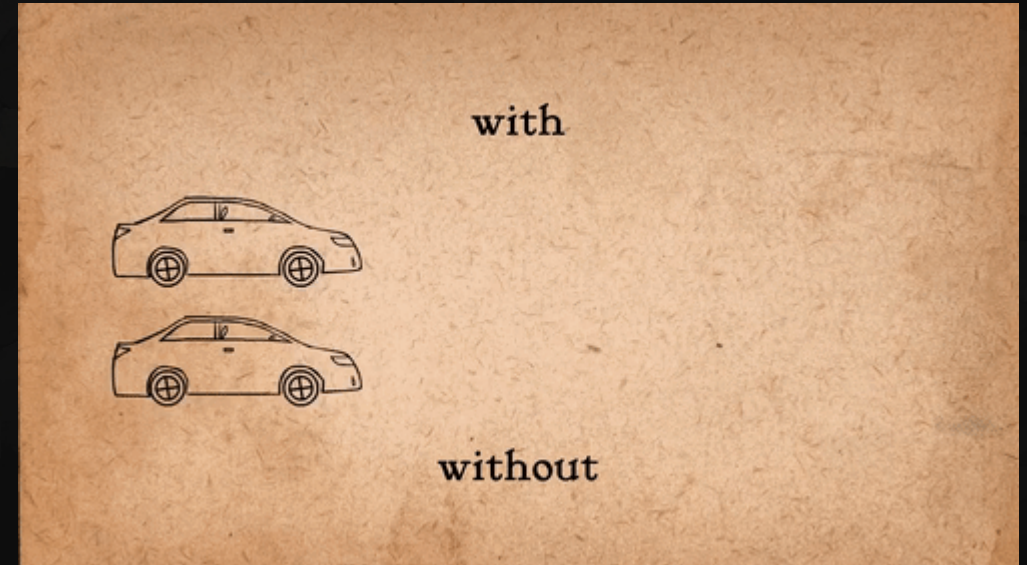
without





## 6. slow-in and slow-out

- ease-in and ease-out effect
- e.g., a pendulum or a car
- adjustable parameter when tweening

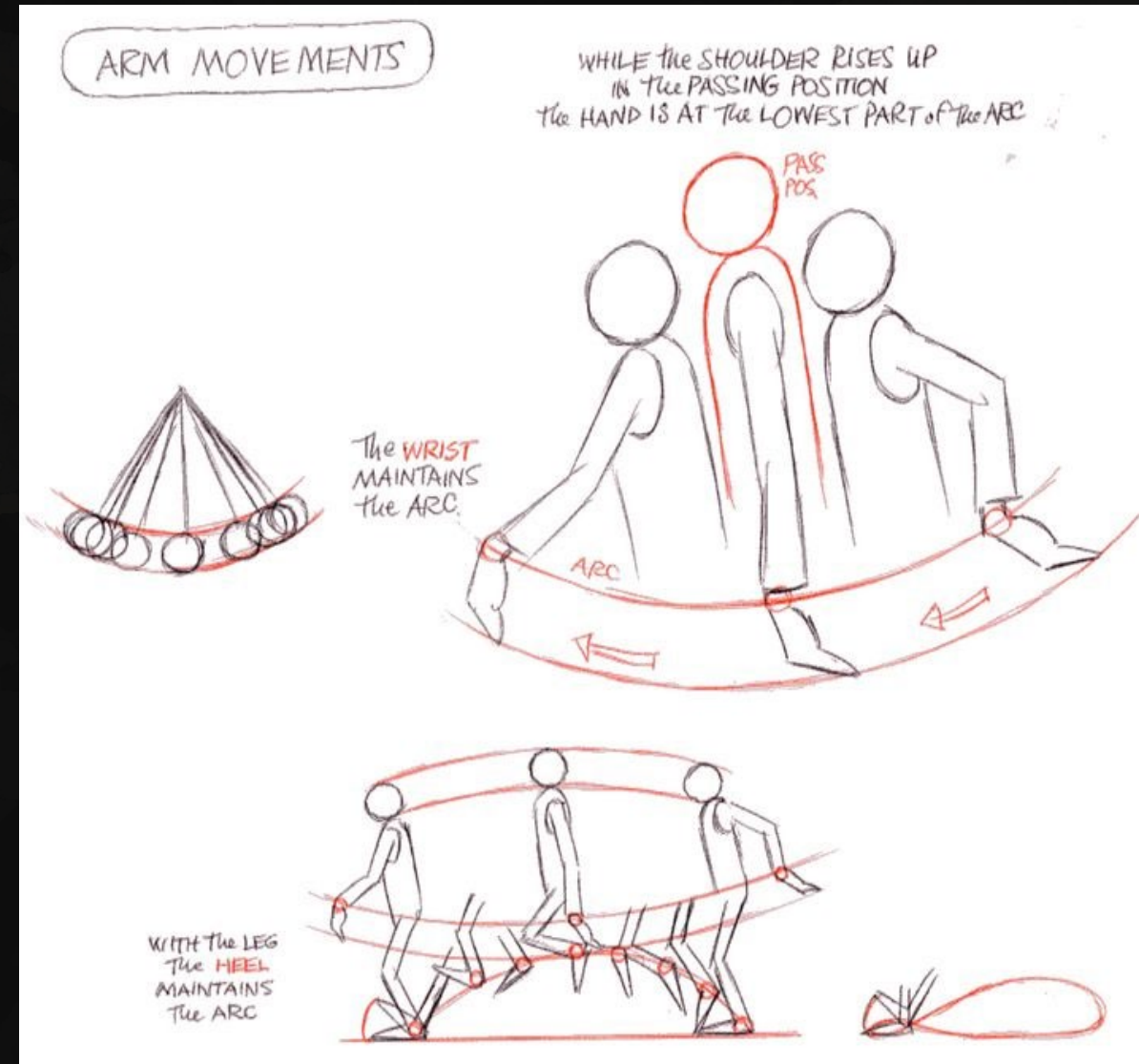


# slow-in / slow-out example



# 7. arc motion

- almost every element including humans move in circular paths rather than straight lines
- when a head moves or an arm is turned it always has an arc motion
- speed of the arc is important

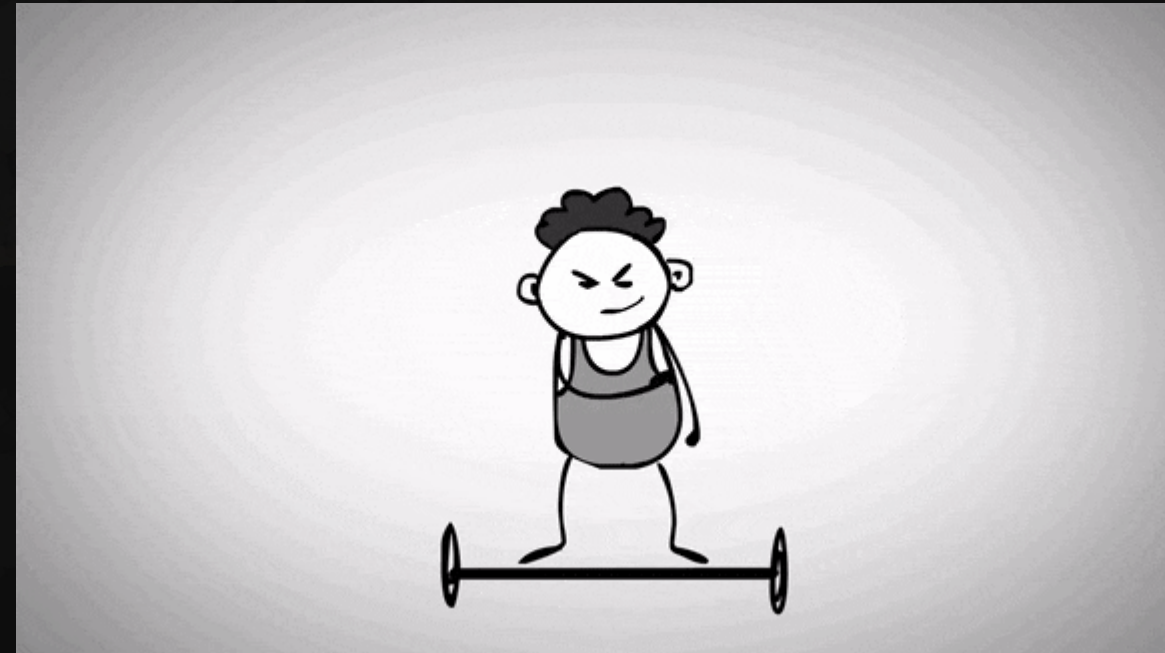


# arc motion example



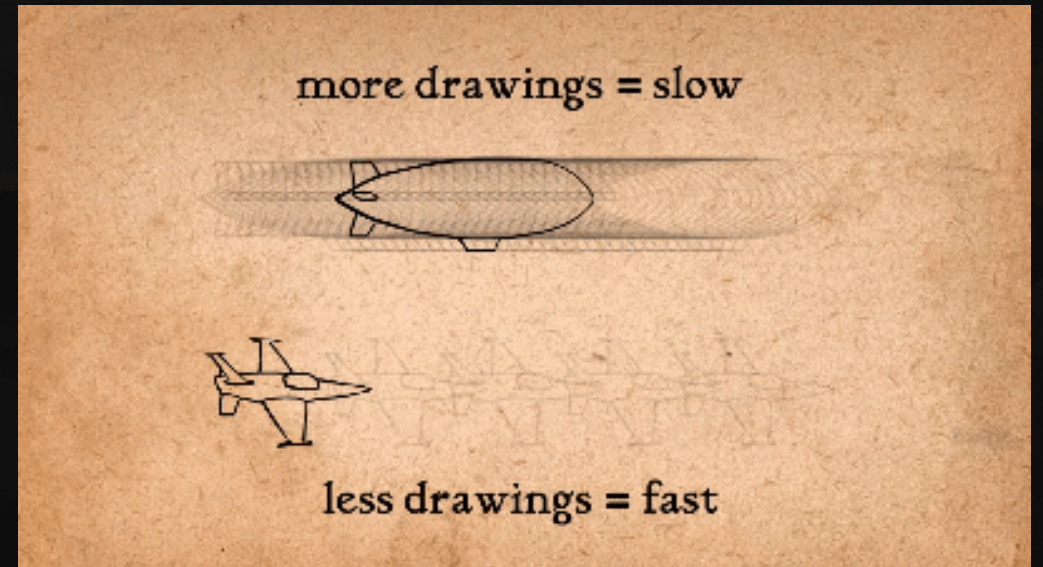
## 8. secondary action

- all movements added to support the main/primary action are called secondary actions
- improves personality and adds dimension to characters and objects



# 9. timing

- frame's arrival is vital to animation
- slower or faster can ruin experience
- delay or lack there of can help make animations more engaging



# 10. exaggeration

- too much realism can sometimes ruin the overall animation and the scene
- spice things up with surreal elements
- e.g., facial distortion, character movement
- push the limits of your characters





# 11. solid drawings

- forms presented in 2D, strive to look 3D
- understanding of volume essential





## 12. appealing

- create images that will be interesting and compelling to audiences
- believable performance vs robotic and unnatural
- element that causes players to believe in character they are interacting with

# principles of game animation

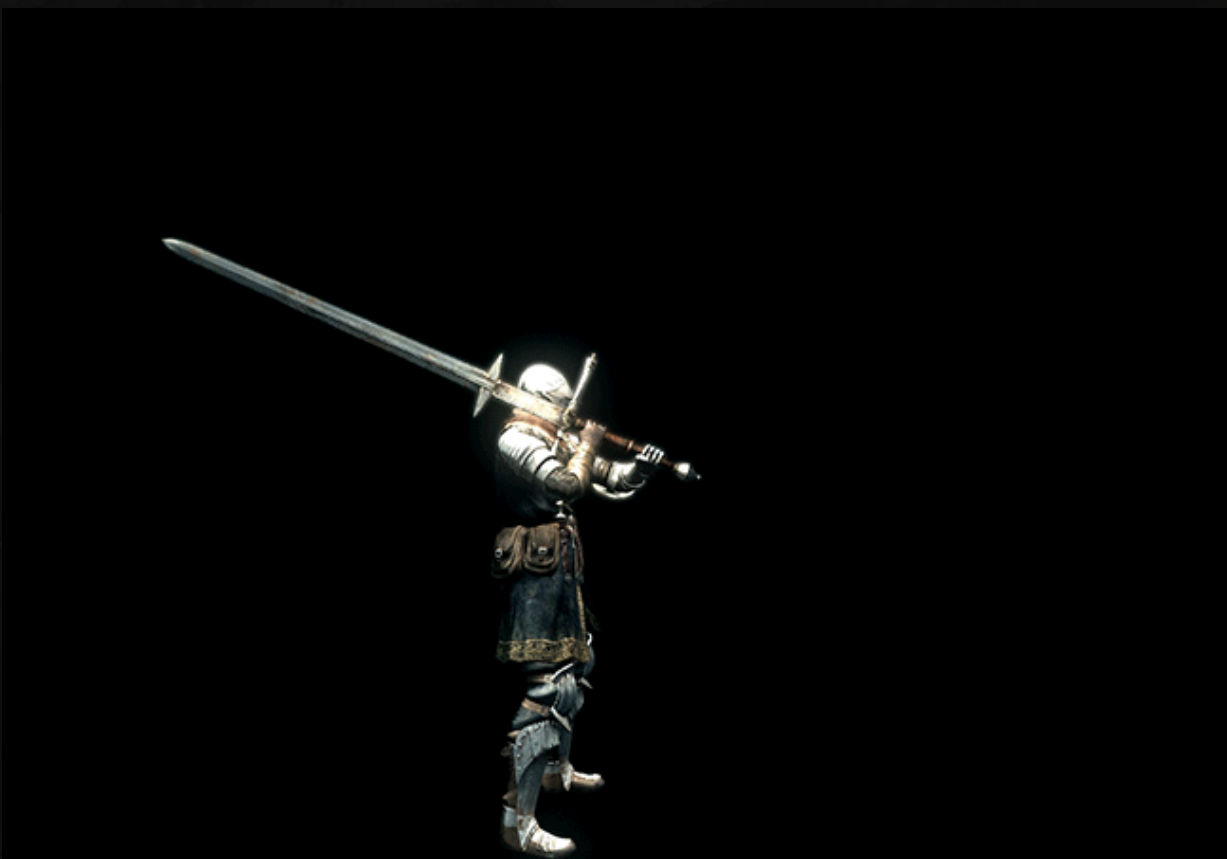
Jonathan Cooper, 2019

- unique to game animation
- complement 12 principles of animation, not replace
- principles: feel, fluidity, readability, context, and elegance

# feel

- interactivity separates video game animation from other traditional forms
- function > form
- response
  - time between player input and corresponding animation
  - vary depending on action, e.g. strong attacks are slower
- inertia and momentum
  - inertia provides sense of feel and fun, i.e., juiciness
  - e.g., time it takes to change direction varies depending on speed and mass of character.





# fluidity

- games use short animations played in a sequence
- need to consider how animations flow together
- blending animations across multiple frames
  - e.g., idle to run
  - Blend Trees in Unity
- include small transitional animations to improve quality
  - e.g., animations for a character changing in different direction
  - number of animations quickly increases



# fluidity

- seamless cycles, e.g. “loops”
  - animations that loop back on themselves
  - end (including momentum) should perfectly match the start
- settling
  - avoid large movements ending abruptly
  - long tail at end of animation that is interruptible
  - achieved by copying desired end pose to end of animation but ensuring that some elements (e.g., limbs) arrive at final position at different times
- blend animations to avoid “pops”

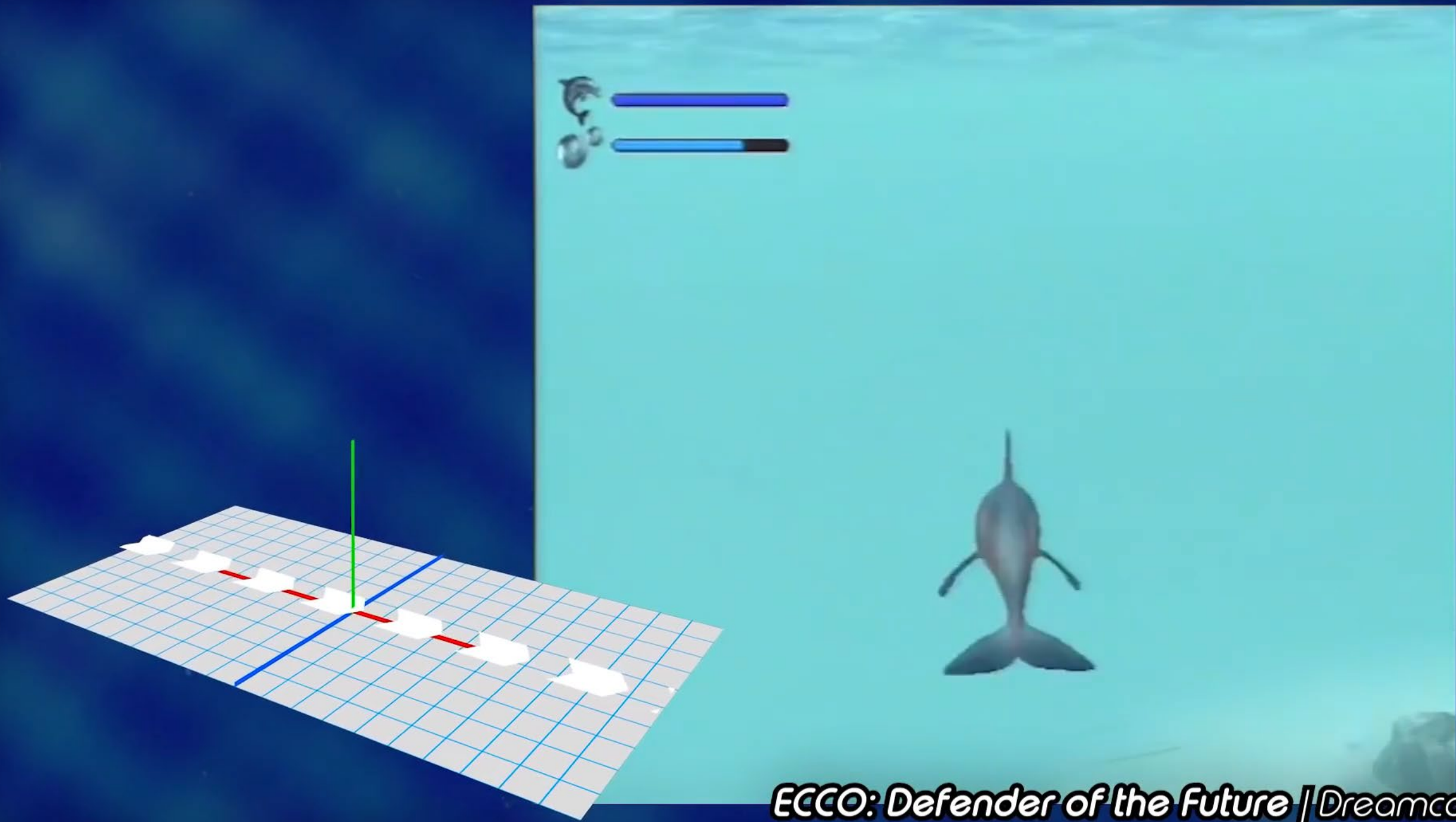


# settling



# readability

- unlike non-interactive forms of media, 3D video game animations may be viewed from all angles
- avoid keeping movement all in one axis
- consider camera angles used in game



*ECCO: Defender of the Future | Dreamcast*



# context

- in video games, animation need to occur in many different situations
- distinction
  - give unique qualities to player character and important NPCs
  - e.g., variety of run cycles depending on environment
  - level of distinction depends on how much control animator has over context
- repetition
  - for animations that are cycles, avoid distinction
  - e.g., one step noticeably longer than others for walking animation



# elegance

- animations flow in and out of one another, triggered within underlying systems affected by player input
- “bang for your buck”
- simplicity of design
  - concentrate on essential elements of animation
  - no more design than is required
  - reuse and sharing
  - layering and combining animations
  - blending



# 2D vs 3D

- 2D: sprites
- 3D: models
  - skeletal animations
  - motion capture
  - inverse kinematics
  - fluid simulations

# sprites

- 2D images representing frames of an animation
- combined together to create animation



# unity: sprites

- sprites defined as textures
- Sprite Renderer controls how sprites visually appear in scene



# sprite sheet

- collection of sprites representing frames of different animations
- limit number of images that need loading
- efficiency in rendering



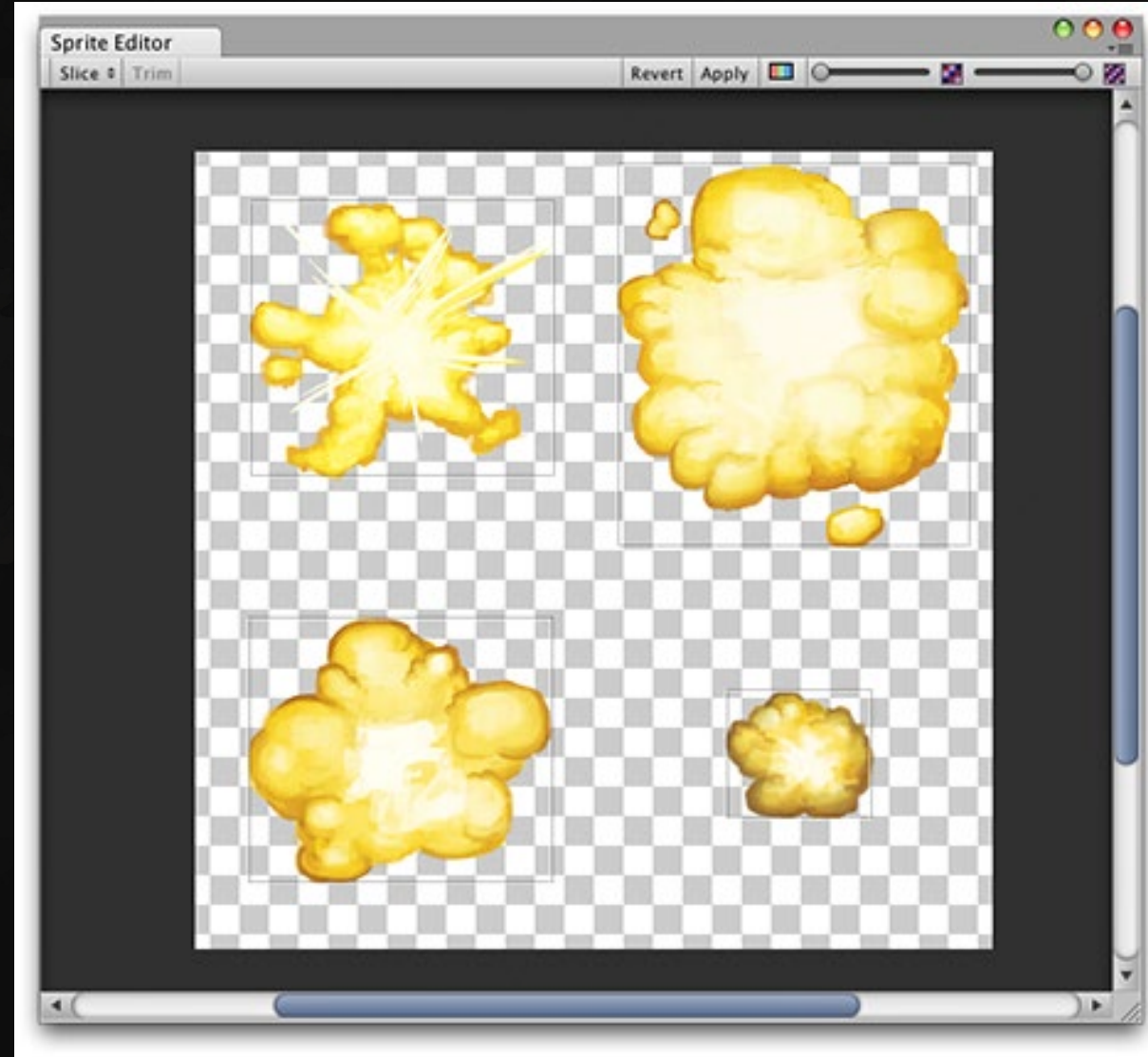
# continuous movement

- create loops or cycles
  - first and last sprite similar in appearance
- running, walking, idle cycles



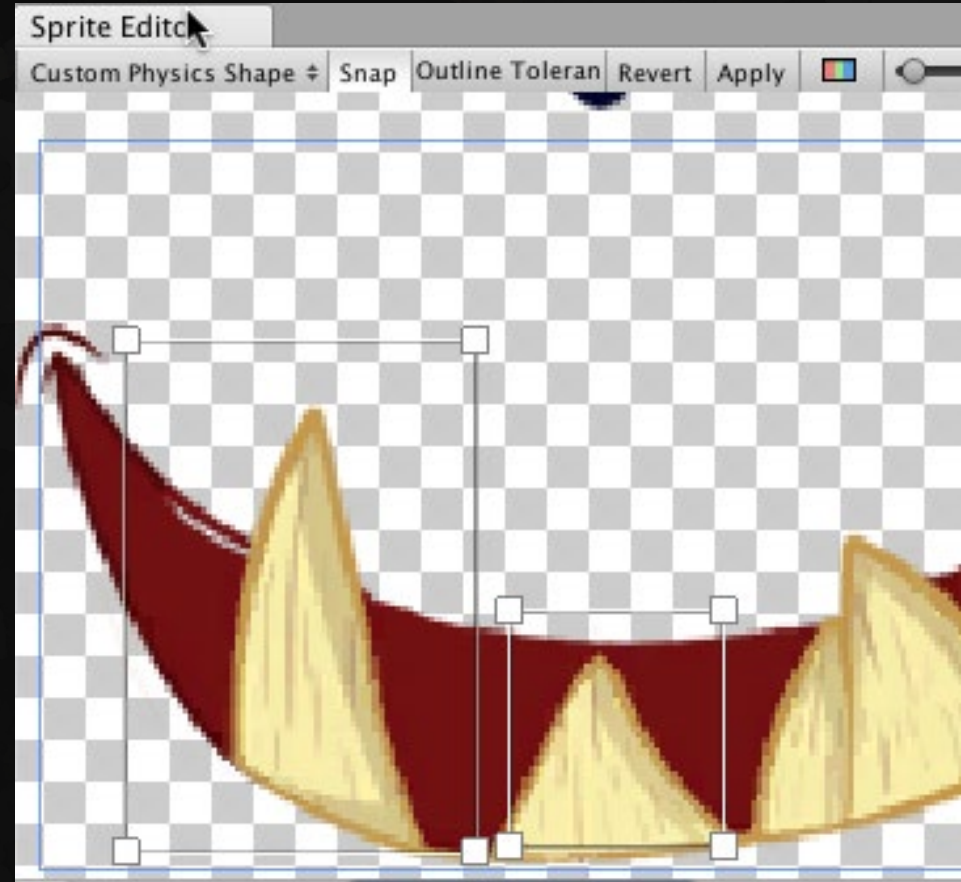
# slicing

- separate sprites on a sprite sheet
- manually or automatic



# unity: physics shape

- outline of sprites collider mesh
- multiple outlines

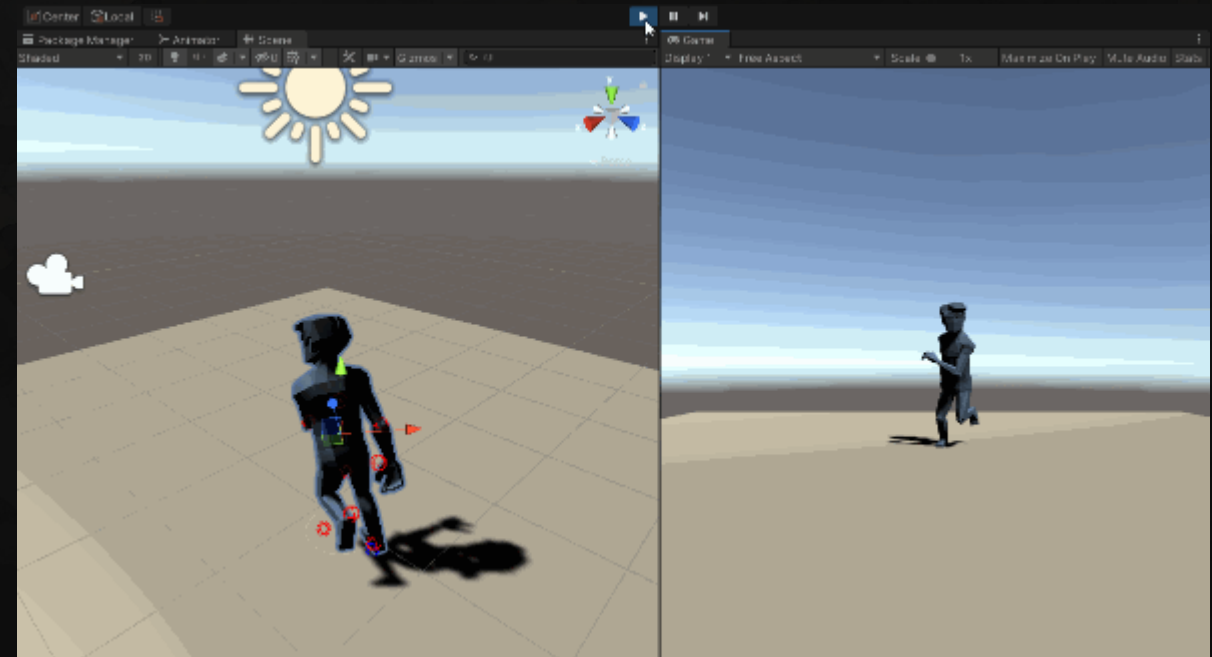


# unity: sprite atlas (sheet)

- grid of images compiled into a single texture
- imported or created from a collection of textures in a scene
- single draw call instead of multiple draw calls for each texture

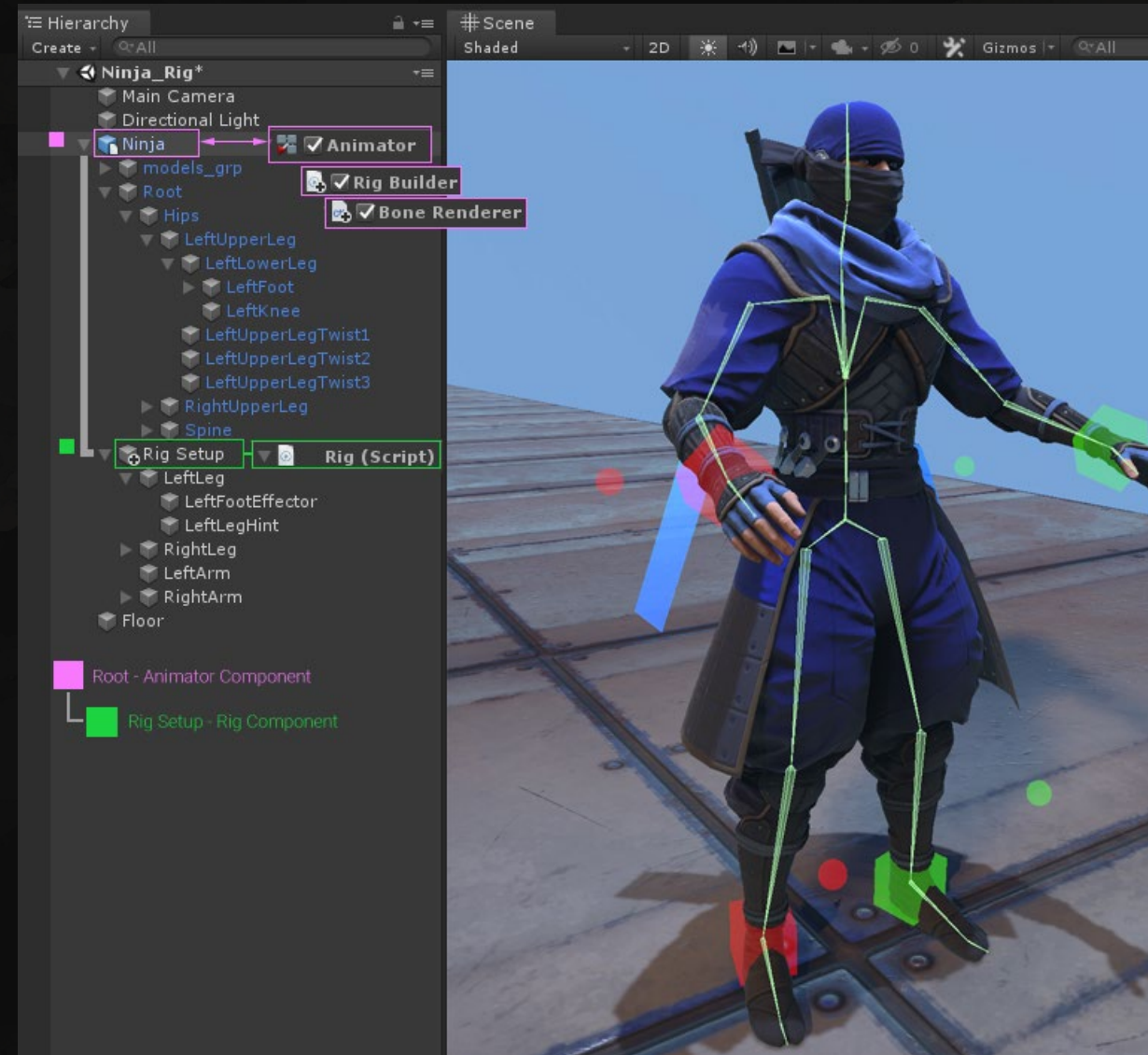
# 3D animation

- skeletal animation
- rigging
- forward/inverse kinematics



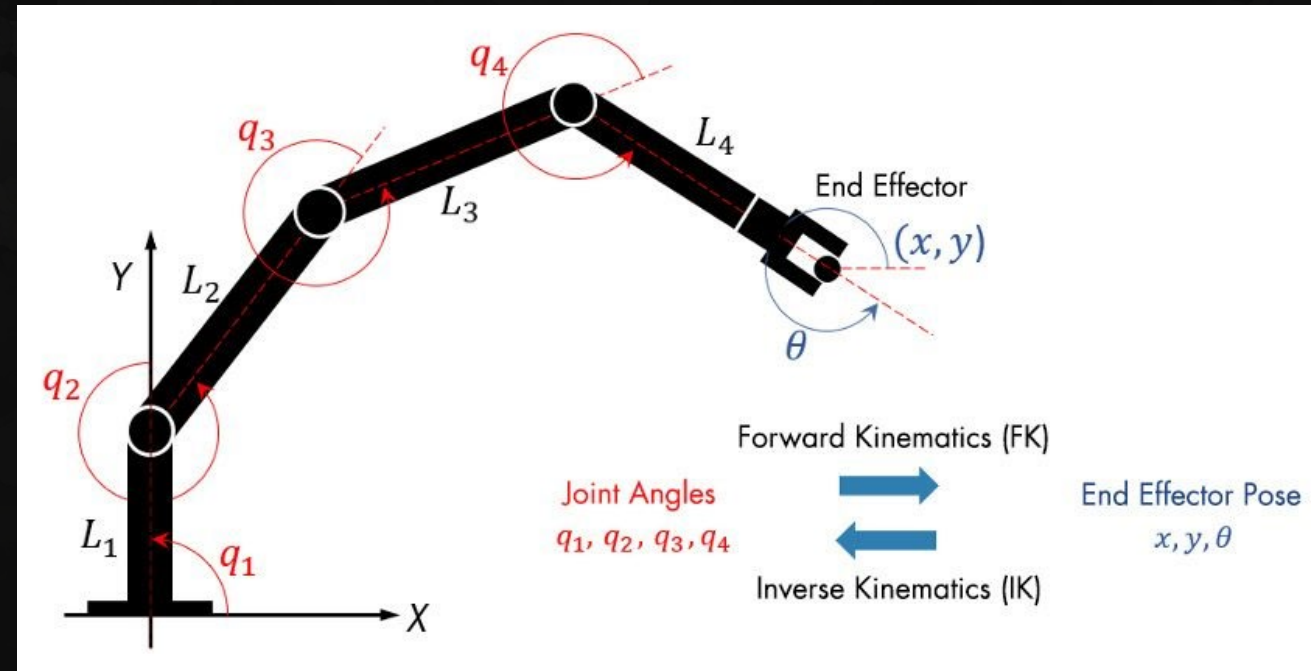
# rigging

- virtual bones, joints, and muscles that allow models to move
- hierarchical structure: transforms of parent affect children



# kinematics

- the branch of mechanics concerned with the motion of objects without reference to the forces which cause the motion.
- geometrically possible motion of a body
- robotics and animation



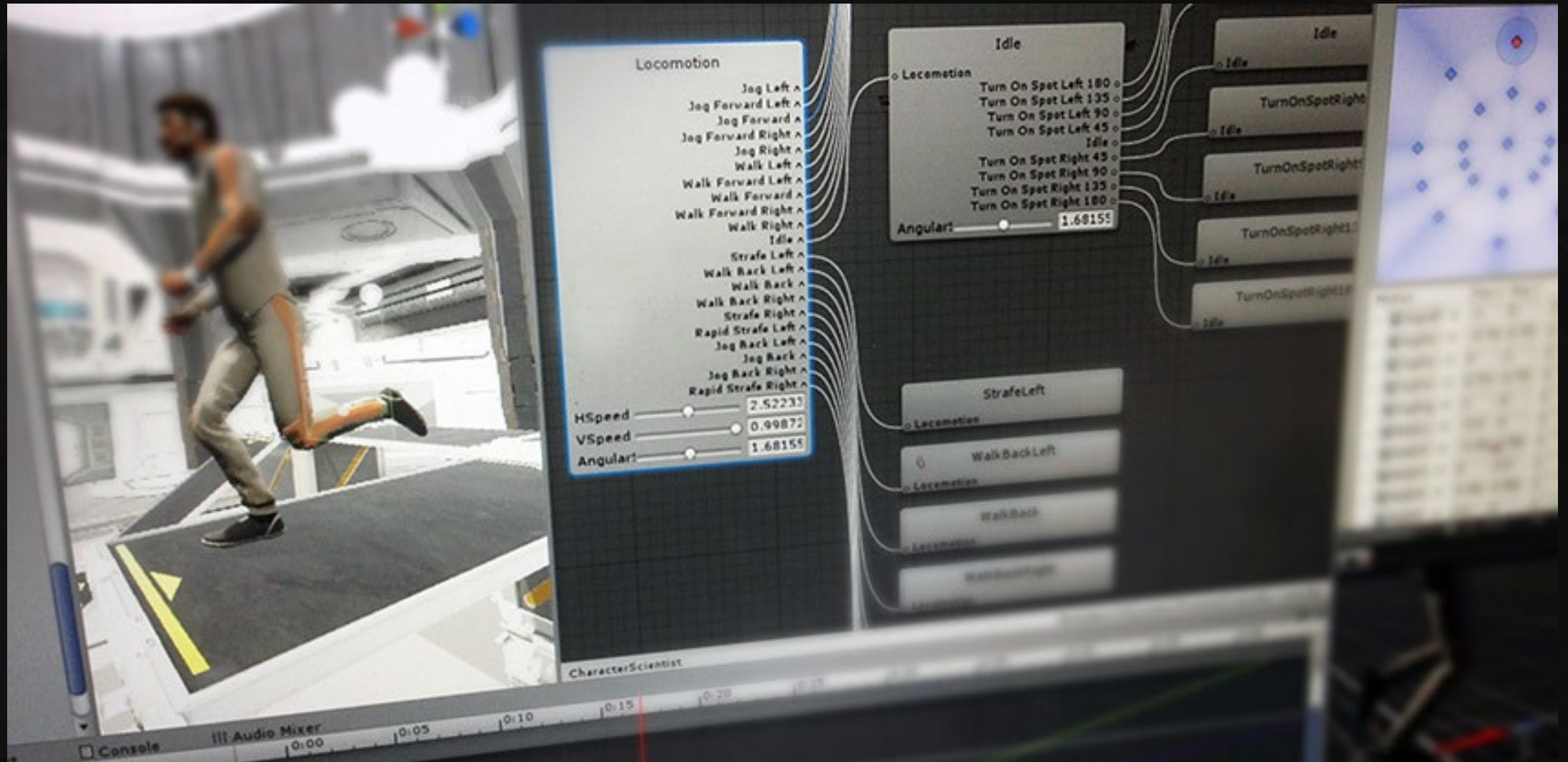
# forward kinematics

- given joint angles and velocities, compute position and velocity of end effector (e.g., hand)
- use of motion capture data or scripting specific movement

# inverse kinematics

- given position and velocity of end effector (e.g., hand)  
compute joint angles and velocities
- how should the joints bend and move given a specific end point
- dynamic animations for different environments
  - (e.g., climbing with different locations for hand and foot holds)

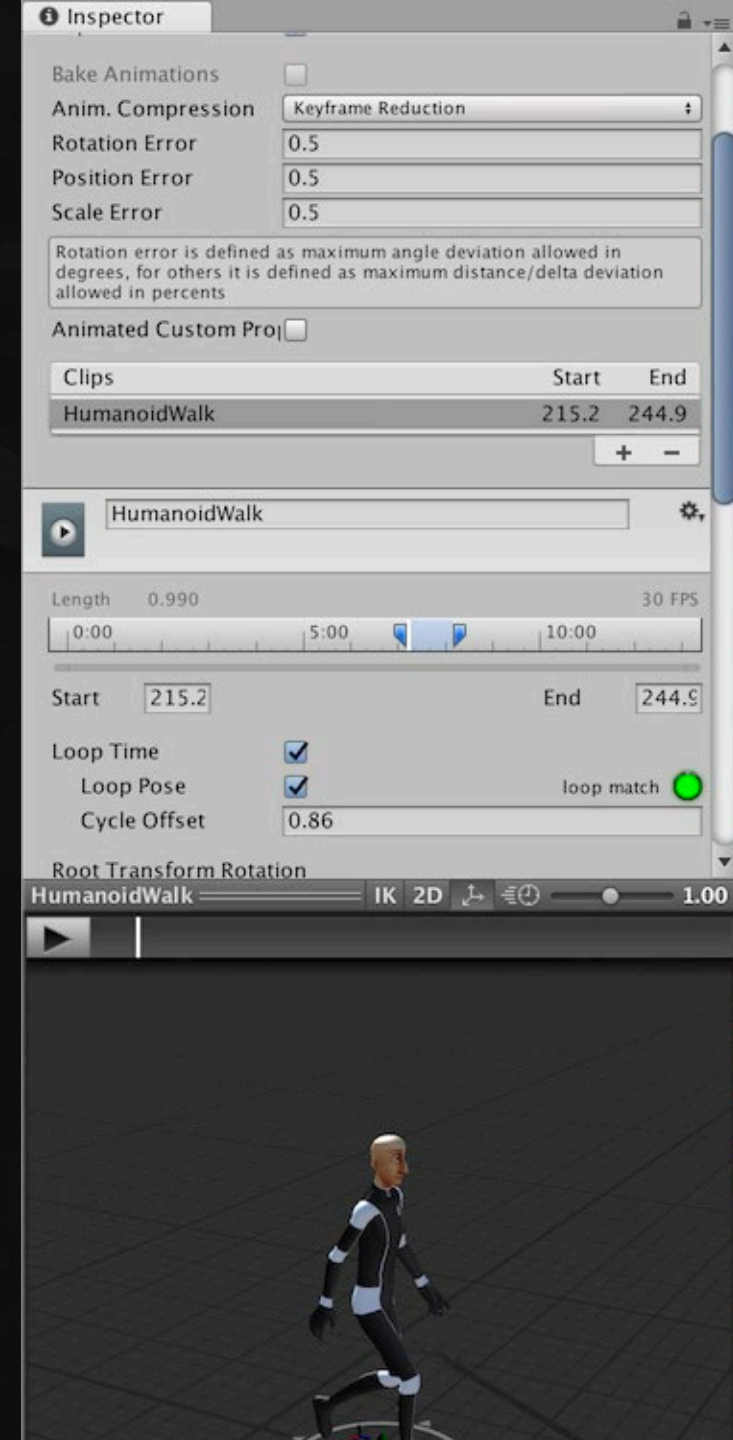
# unity: animation system



# animation clips

- animations from external sources
- created in Unity with Animation Window

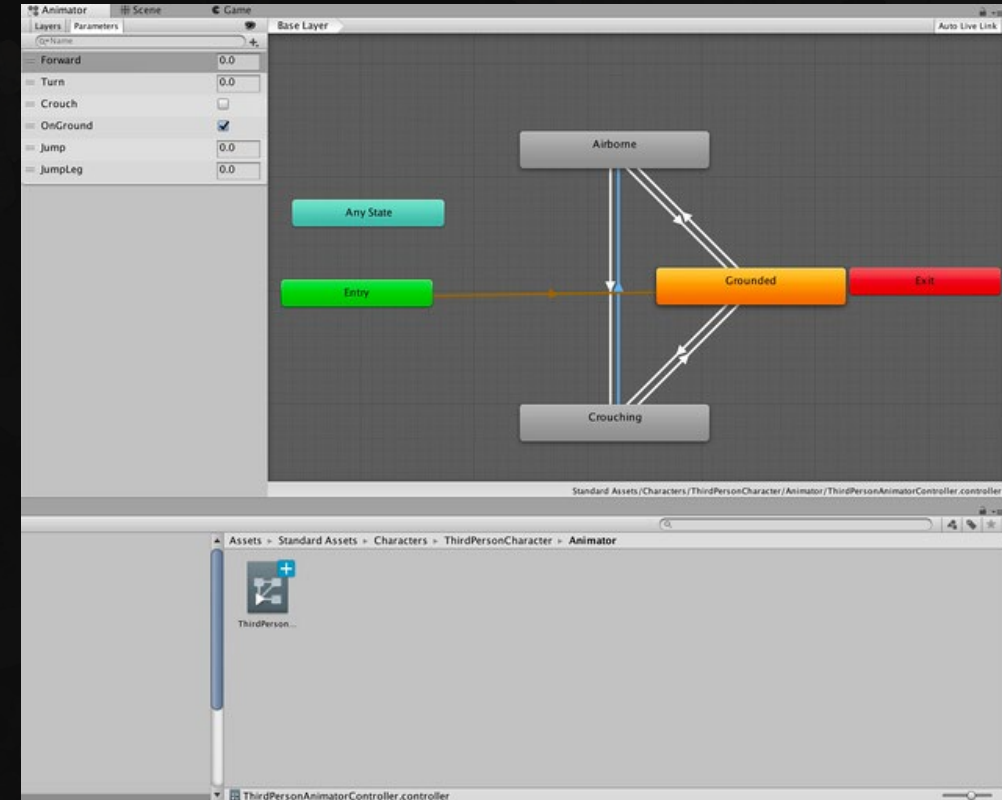
<https://docs.unity3d.com/Manual/AnimationClips.html>

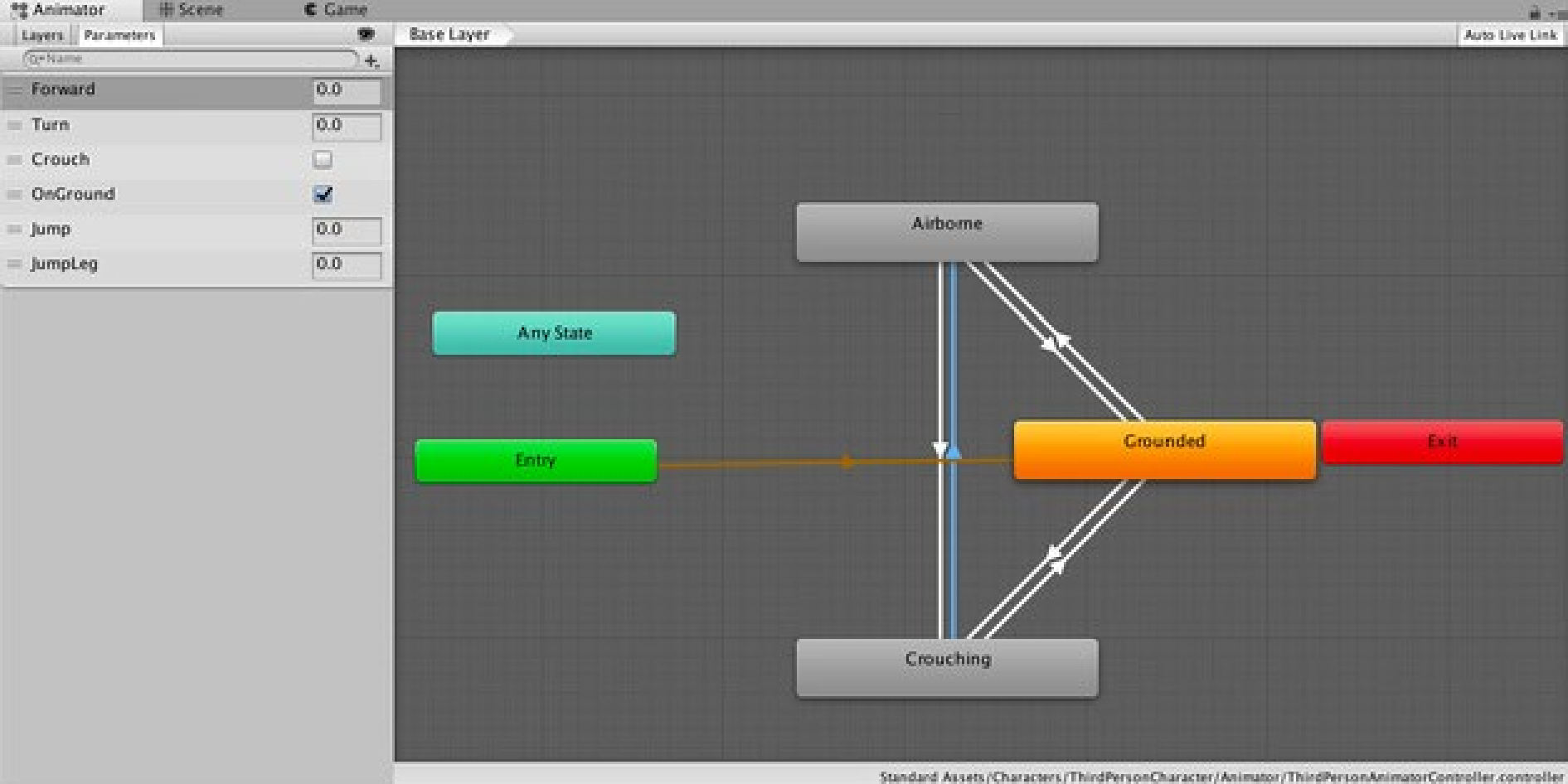


# Animator: finite state machines

- bringing animation clips together via finite state machines
- animation clips represented as states
- specify conditions for transitions (e.g., after animation completes, value of parameter)

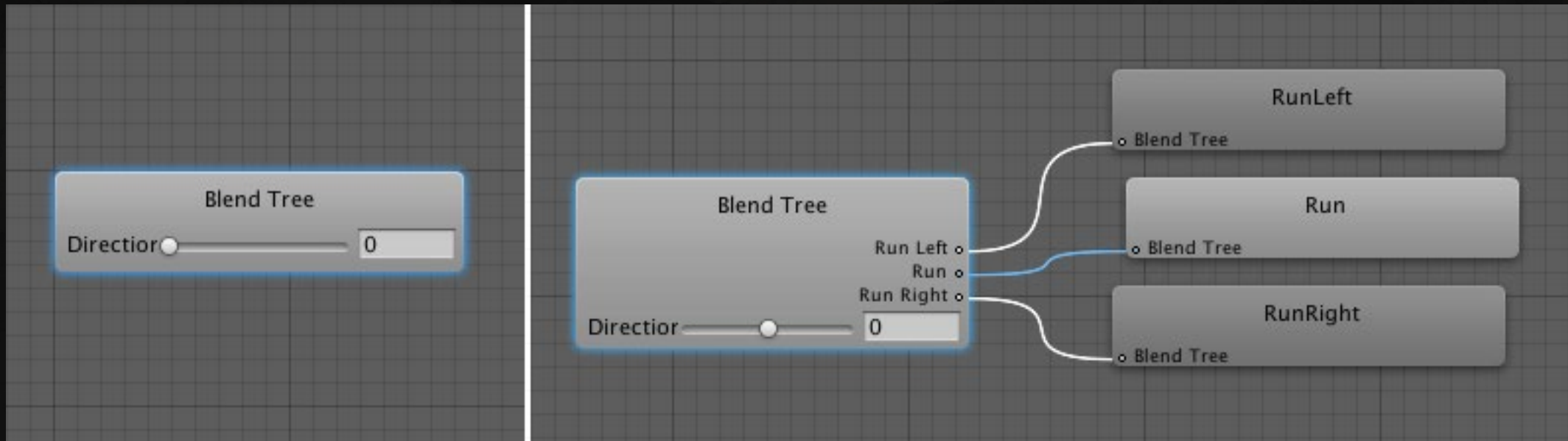
<https://docs.unity3d.com/Manual/Animator.html>





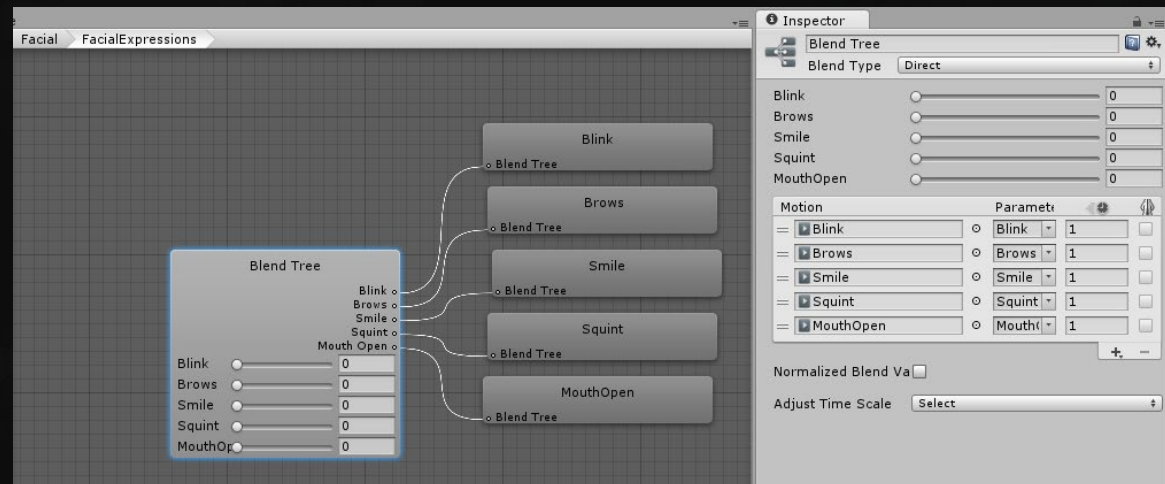
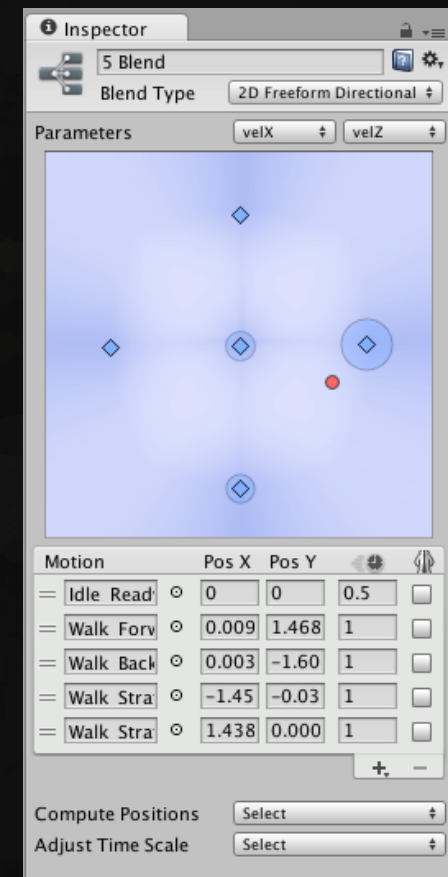
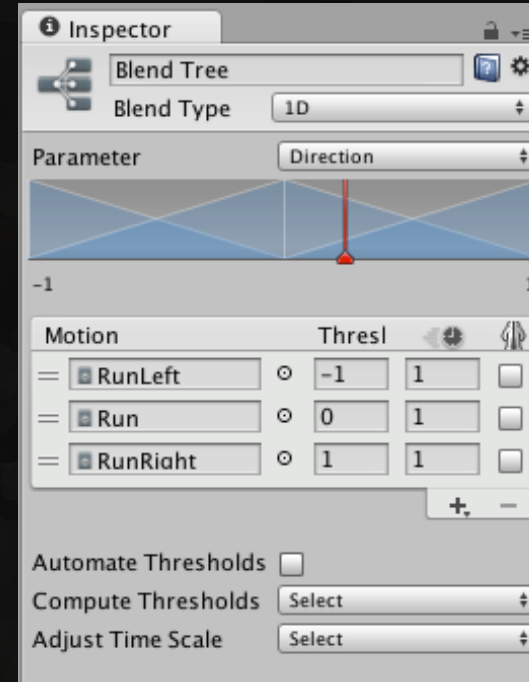
# blend trees

<https://docs.unity3d.com/Manual/class-BlendTree.html>



# blending

- 1D: single parameter
- 2D: two parameters
- direct: weights



# weights: facial expressions



**questions?**

