



# GDC

09 learn  
network  
inspire

[www.GDConf.com](http://www.GDConf.com)

Game Developers Conference®

March 23-27, 2009 | Moscone Center, San Francisco

# Dynamic Walking with Semi-Procedural Animation

Rune Skovbo Johansen

Creative Programmer, Unity Technologies





GDC<sup>09</sup>  
learn  
network  
inspire

# Walking

# Importance of Walking

- » Players often spend most of the time walking around
- » Games have increasingly detailed environments
- » We want to make it look good when walking through these environments



# Keeping Experts in Control

- » **Adaptation to Terrain**  
with believable walking over any steps and slopes
- » **Animators**  
in full control of style and personality
- » **Game Programmers**  
in full control of game logic and character movements

# What is Wanted

## Minimum Work

- » Only few animations are needed (as few as 2 per character)
- » Fully automatic blending of multiple walk cycles

## Maximum Flexibility

- » Should work for humans, quadrupeds, bugs, spiders, ...
- » Walking with any direction and curvature on any terrain

# Demo of Locomotion System





# Clarification

The Locomotion System is *not*:

- » A physics-based system or active animated ragdoll system
- » A behavior-based system  
(like NaturalMotion's euphoria)
- » A unified system that can be used for all animation of a character

# Best of Both Worlds

- » Full control over style
- » Dynamic movements

Traditional  
key-framed  
+ blending

Semi-  
procedural  
animation

Procedurally  
generated  
motion



# Minimal Model

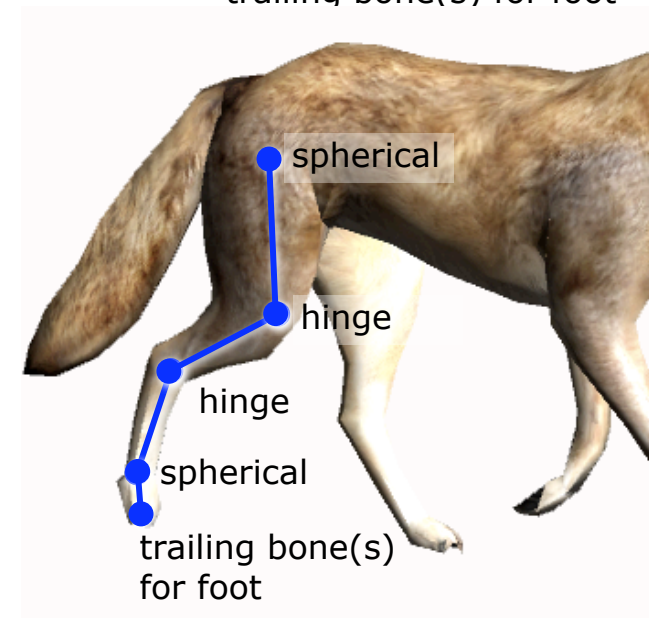
**Procedural animation**  
**=**  
**simulation** based on a **model**

- » Models are based on *assumptions*
- » Get a highly *flexible* system by *reducing* the assumptions



# Character Requirements

- » One or more legs
- » Each leg has two or more bones from hip to foot
- » First and last joints are spherical (ball-and-socket) joints
- » Middle joints are (at least) hinge joints
- » No animated scaling / stretching of leg bones (in current implementation...)



# Animation Requirements

- » Animations must be cyclic!
- » Animations are in character space  
"walking on the spot"
- » Legs (knees) should not over-bend
- » The feet should be at their lowest when in contact with the ground  
(and not penetrate the ground too much)
- » Feet should be moving backwards\* when in contact with the ground, with linear speed

\* for a forward walking animation

# Automating the Hard Work

## Easy setup

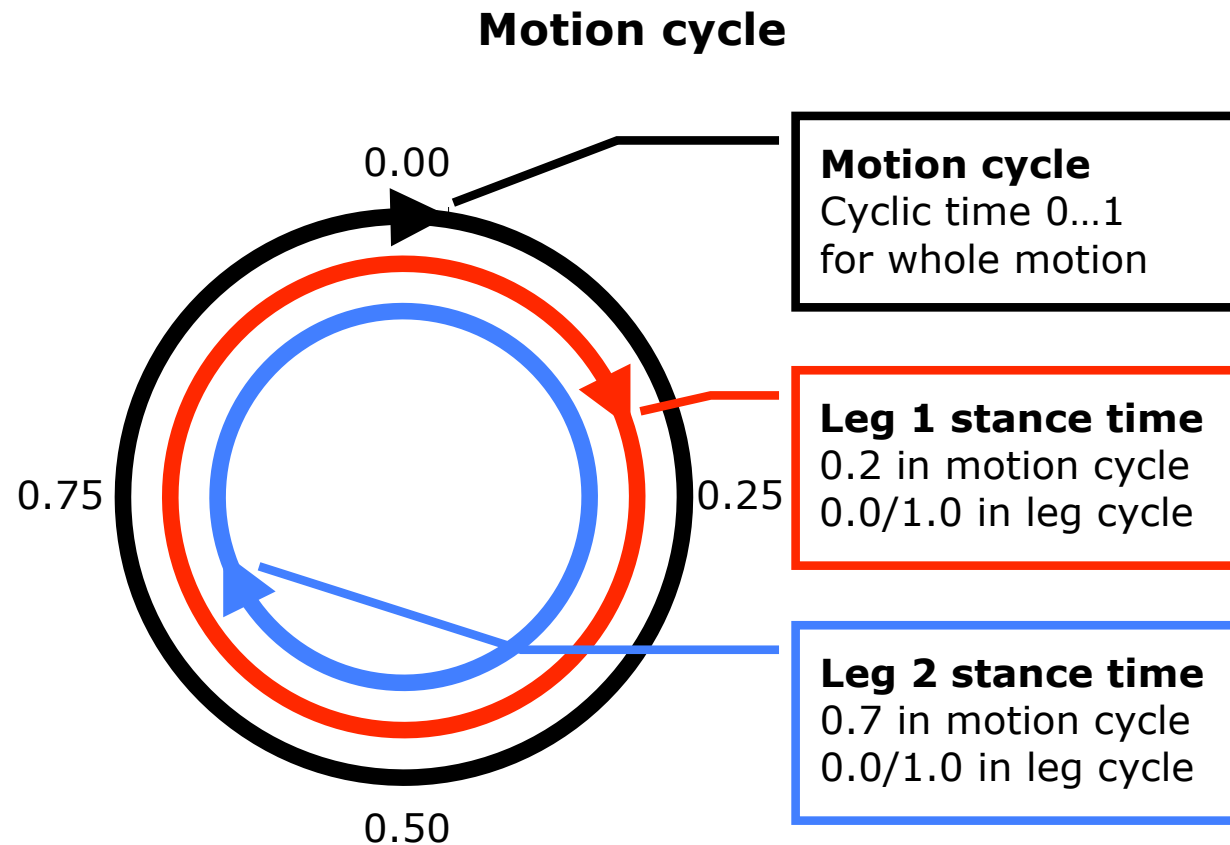
- » List of legs
- » List of animations

## Each animation is automatically analyzed:

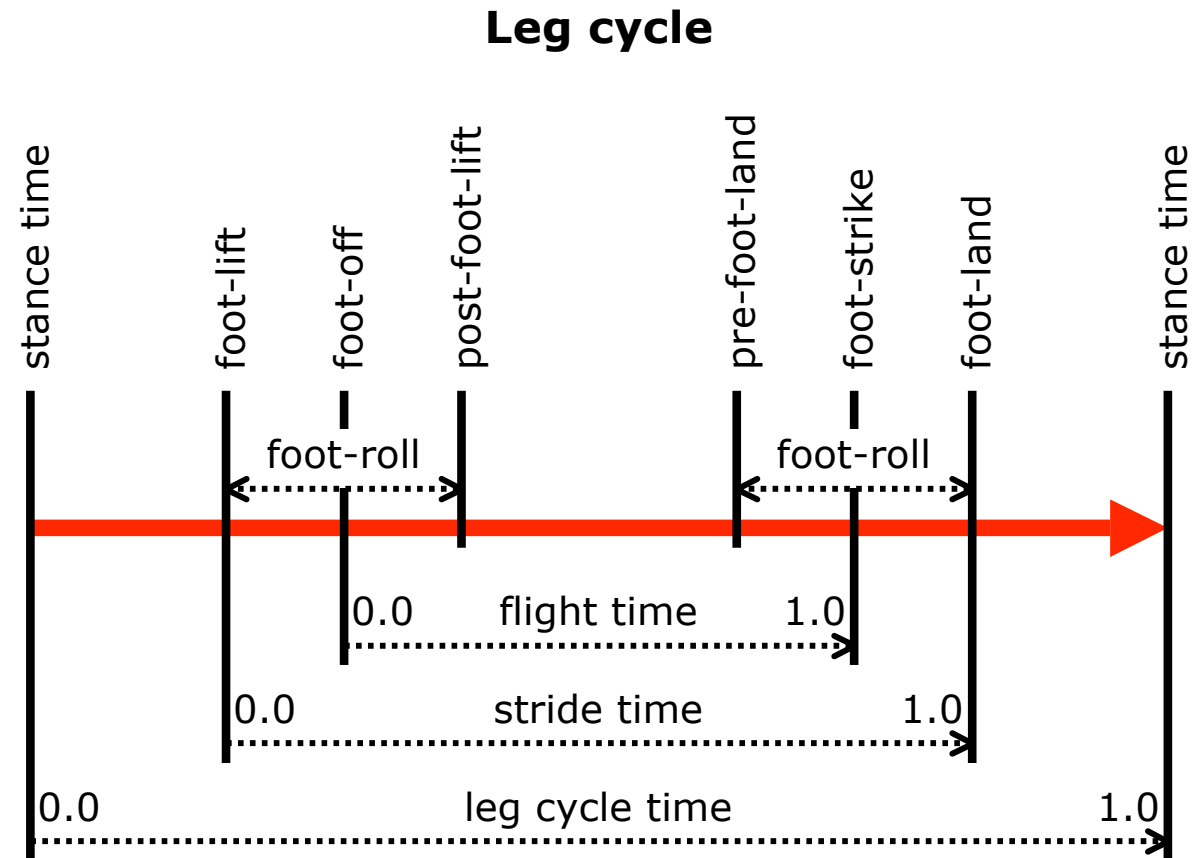
- » Speed and direction (velocity)
- » Foot cycle lifting and landing keytimes



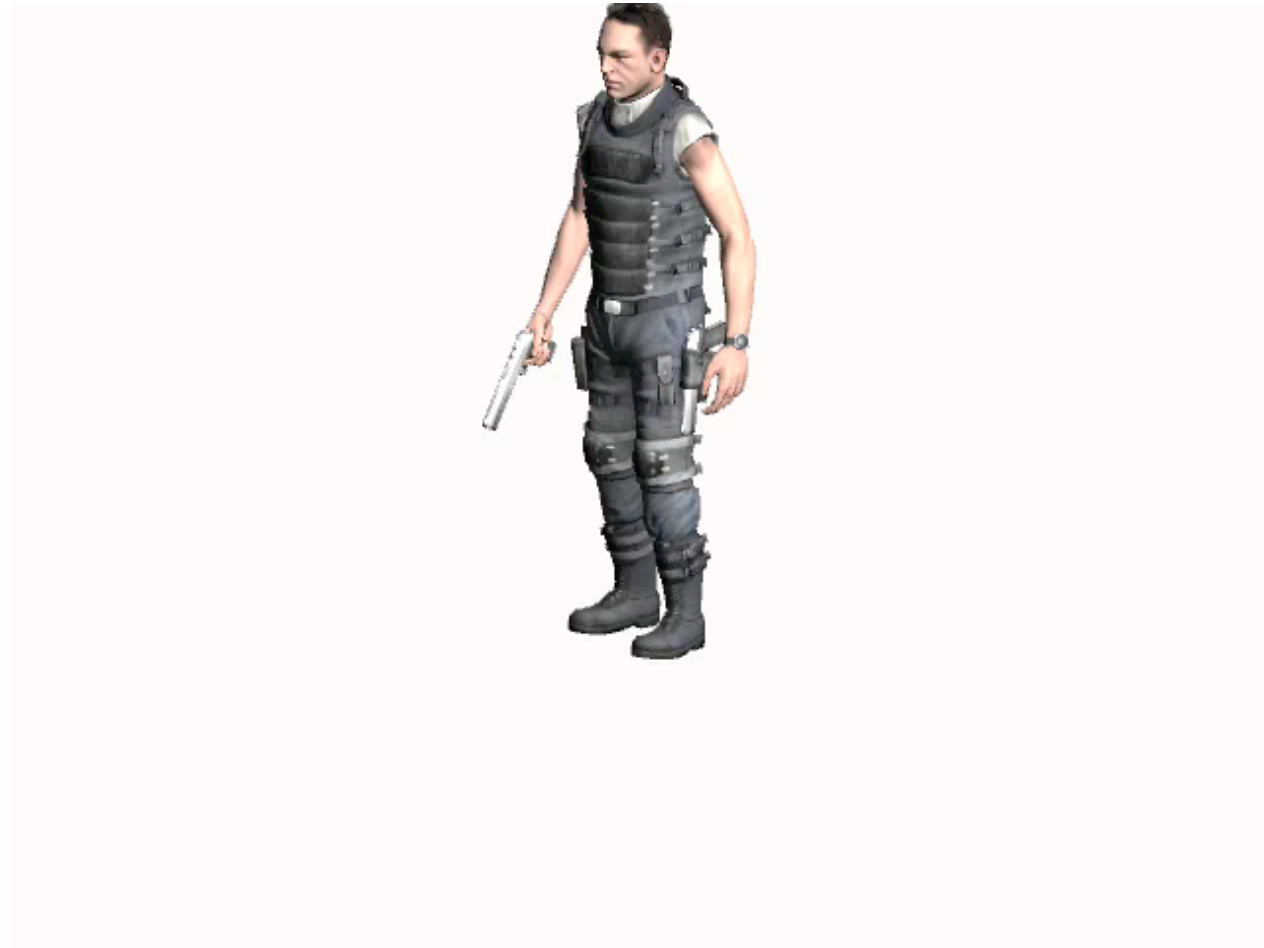
# The Motion Cycle



# Leg Cycle and Keytimes



# Sample Heel and Toe Trajectories

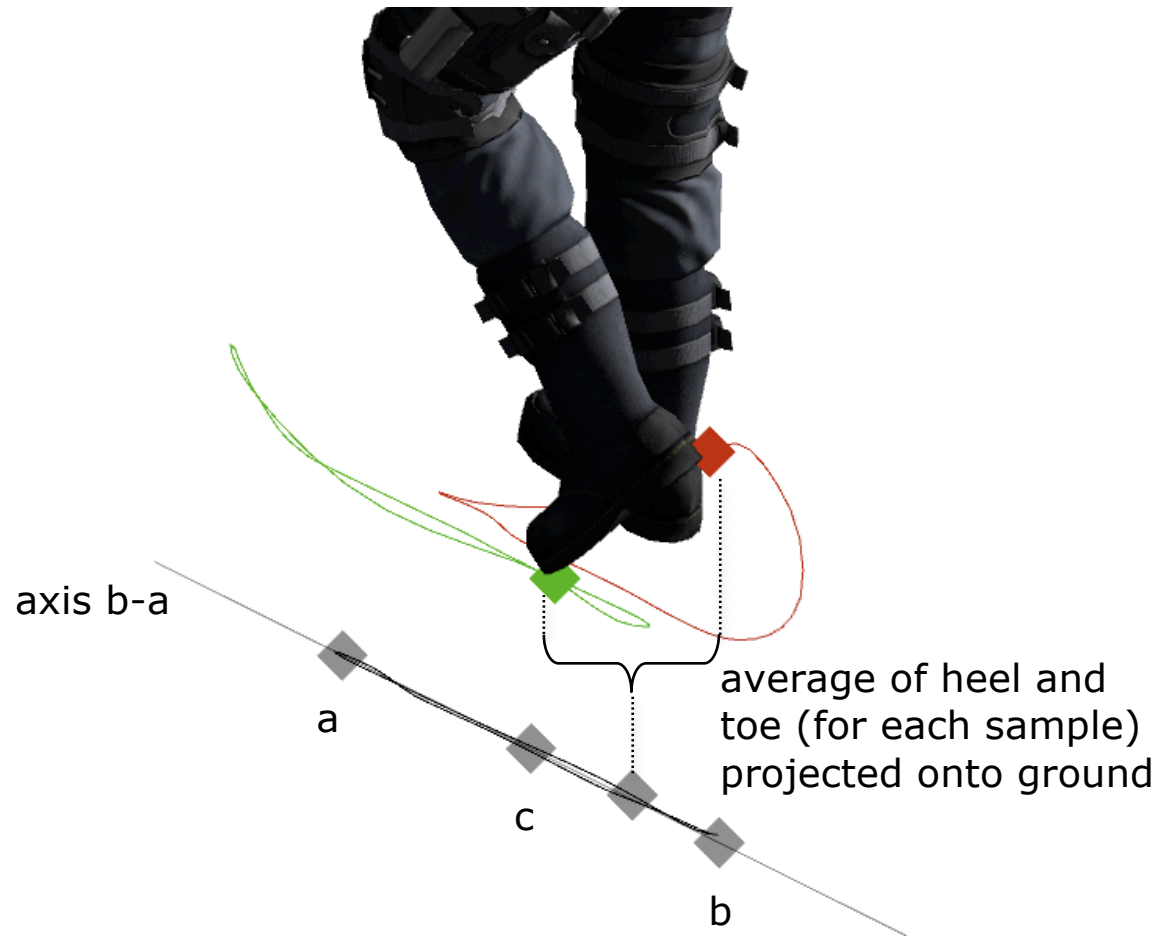




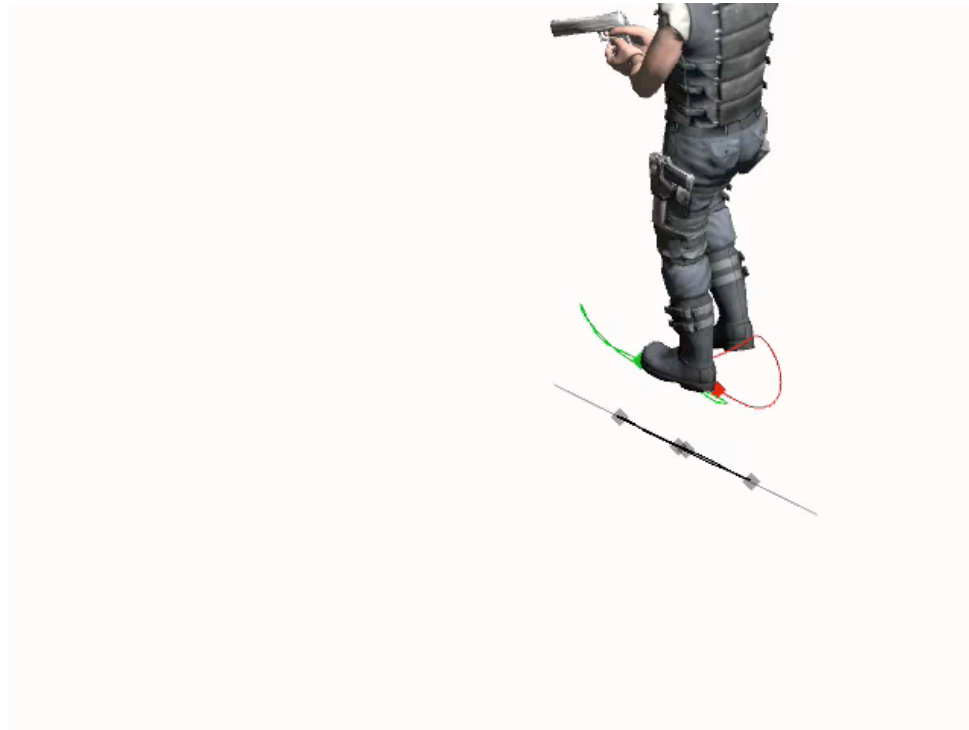
# Movement Axis



# Movement Axis



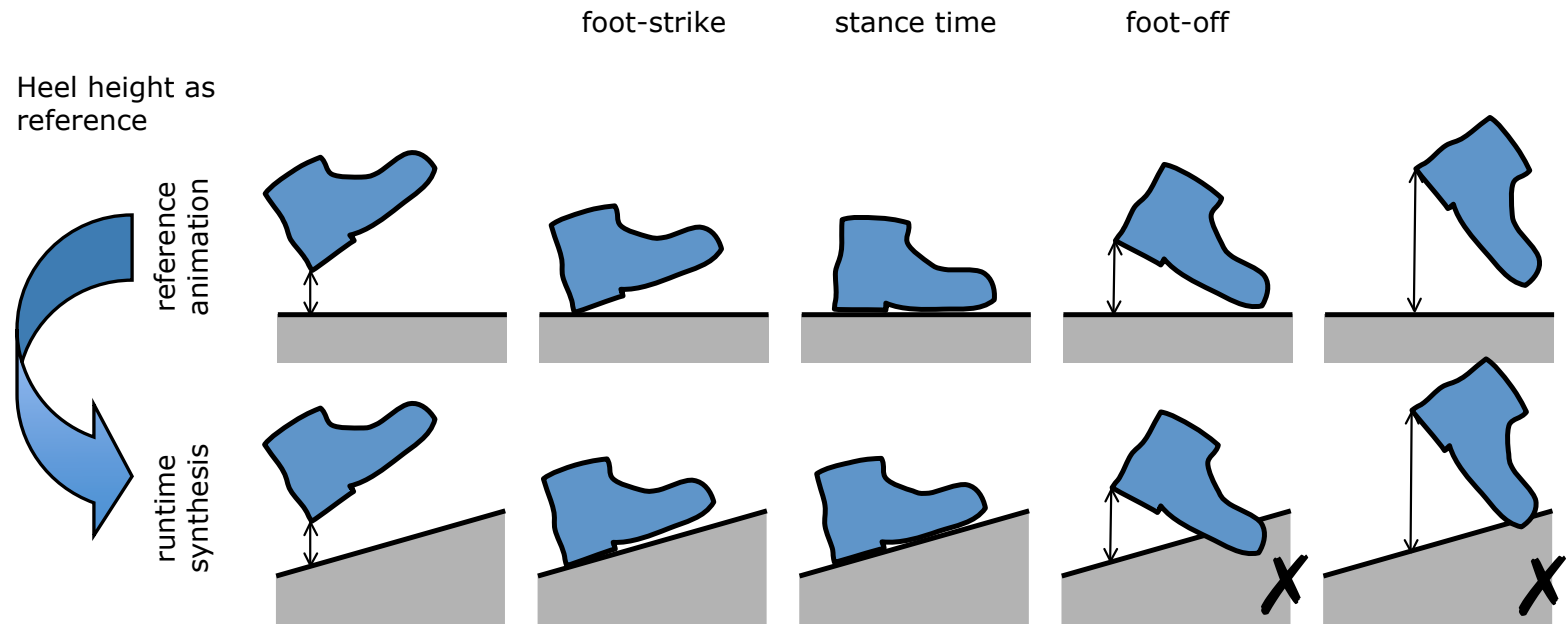
# Stance Time



To determine the stance time, a cost value is calculated for each sample based on:

- » Height of the heel and toe at that sample
- » Position along movement axis (middle=low cost)

# Foot Relative To Ground



How to measure height of foot over the ground?

- » Heel as reference point doesn't work.
- » Neither does the toe, or the middle.



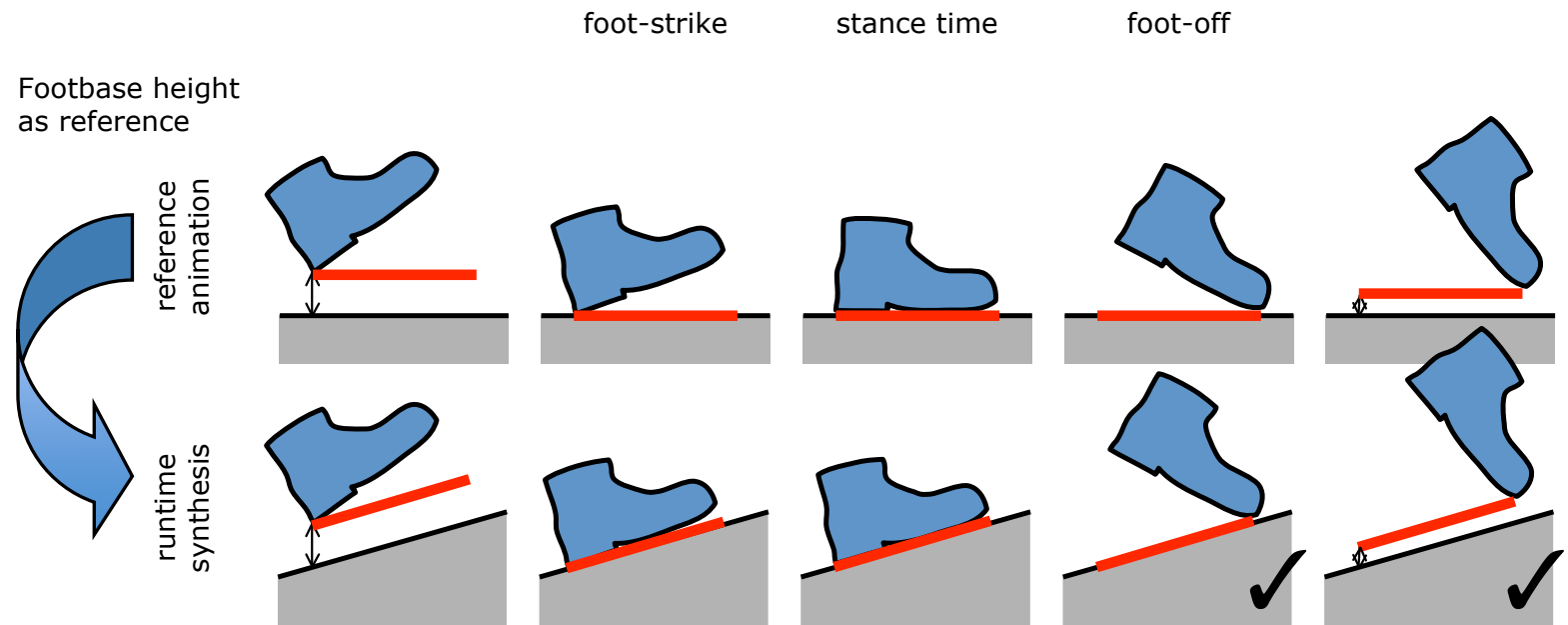
# The Footbase



Introducing: **The footbase**

- » Like a plate under the foot
- » Always perpendicular with the ground underneath
- » Foot touches footbase with heel, toe, or both

# The Footbase

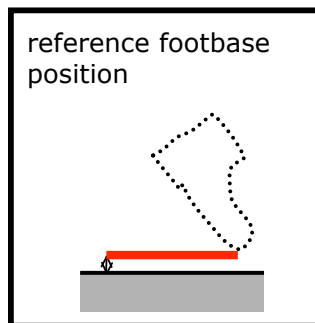
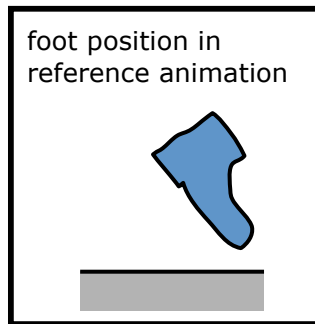


How to measure height of foot over the ground?

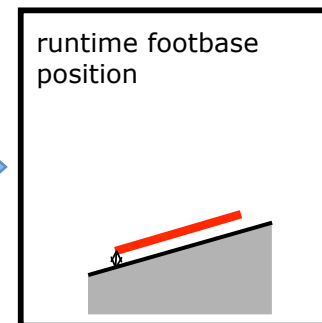
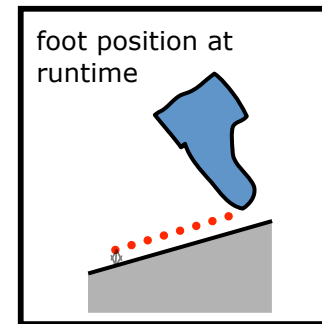
- » Find the footbase, given position and alignment of foot, and the slope of the ground
- » Measure the height of the footbase above the ground

# The Footbase

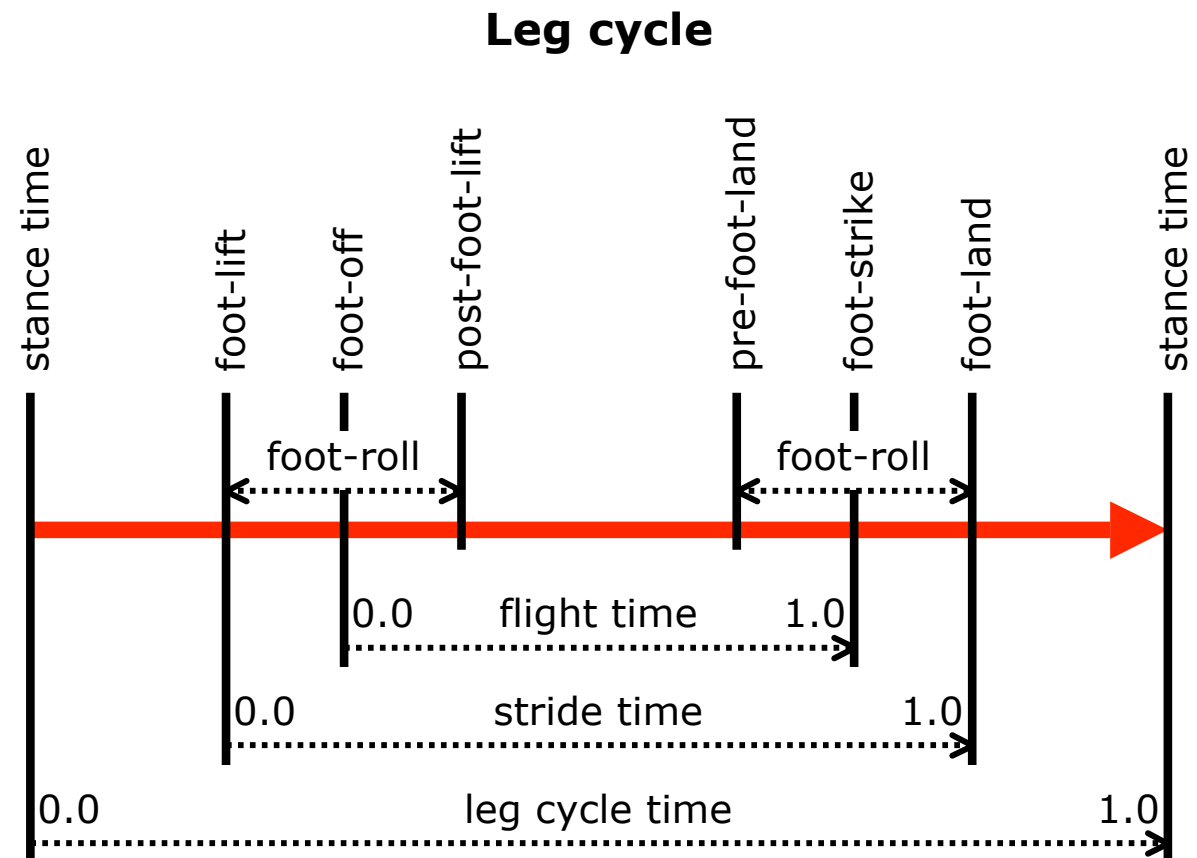
Motion analysis of  
reference animation



Motion synthesis at  
runtime



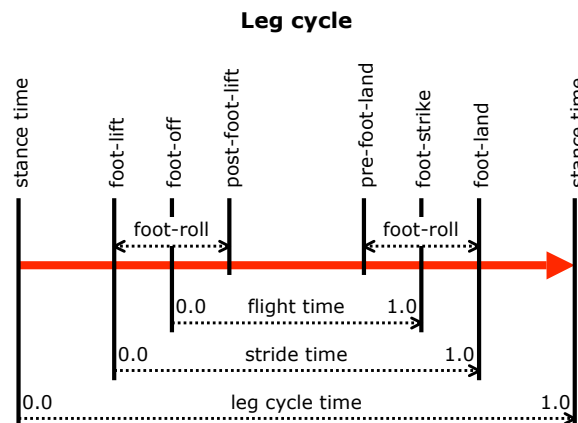
# Finding the Keytimes





# Finding the Keytimes

- » Keytimes need to be set for each leg, for each animation.
- » Animators can annotate these.
- » In the implemented system, an automatic heuristic was used as part of the motion analysis.



# Finding the Velocity

- » Measure the distance that footbase has moved from *foot-land* time to *foot-lift* time.
- » Divide by time-span between those two times.
- » Do the above for each foot, then use the average as the velocity for the motion.

# Normalized Footbase Trajectories



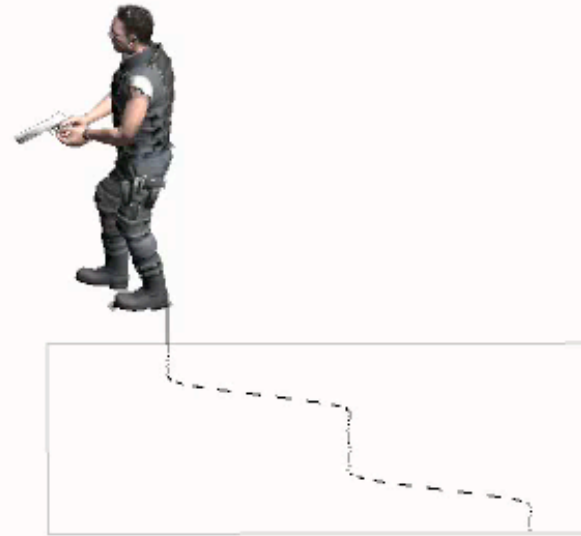
- » With the velocity known, we know the footbase trajectory relative to the ground (i.e. in world space).

# Normalized Footbase Trajectories

Normalize the trajectory and store it:

- » Trajectory starts at  $(0,0,0)$
- » Ends at  $(0,0,1)$ :
  - ⊙ Aligned with z-axis
  - ⊙ Scaled to have step length of 1
- » Keep vertical and sideways components un-scaled

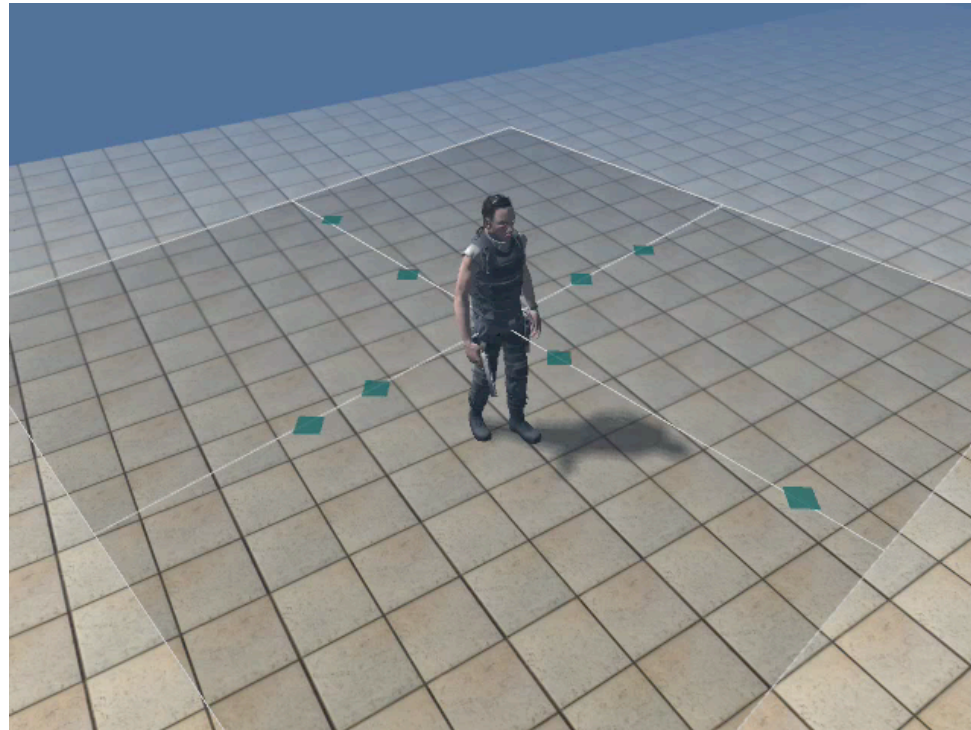




# Analysis done!

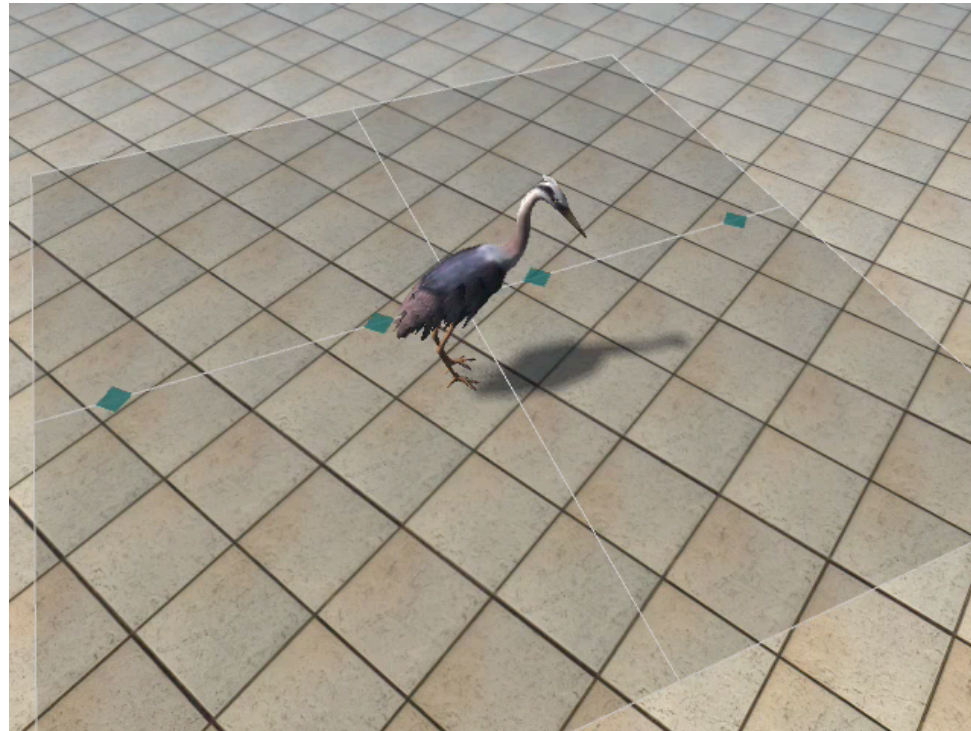
(repeat for each motion)

# Automatic Blending



- » The velocity of each sample animation is known.
- » Based on current velocity, assign weights to neighboring samples.
- » A problem of scattered data interpolation. (Plenty of research on the subject.)

# Automatic Blending



- » Only few animations are needed. (As few as 2)
- » Forward animations can double as backwards.
- » Walking sideways may not always be needed but always make turning look better.



# Blending the Data Too

Blending weights are used to blend animations *and*:

- » Normalized footbase trajectories
- » Keytimes
- » Stance position  
(Position of foot relative to body at stance time)
- » Basically all data analyzed for each animation!



# Leg Movements

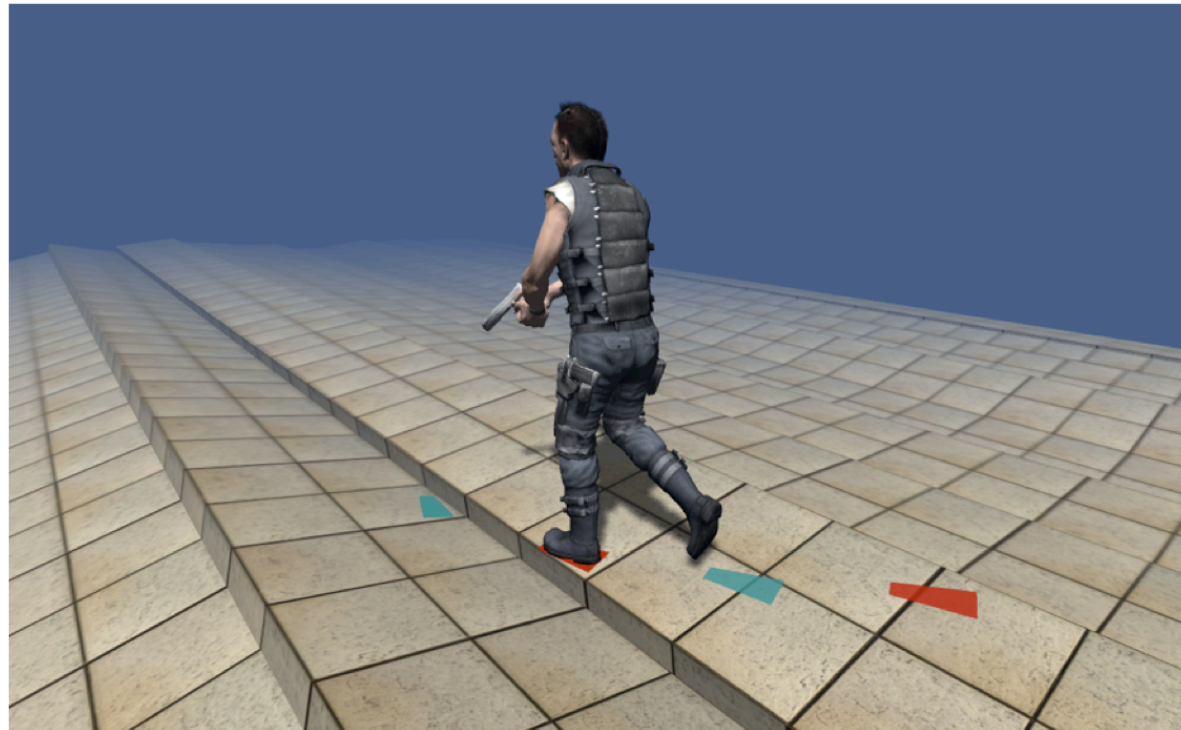
Inverse kinematics (IK) can be used at runtime to move a leg around by specifying the hip and ankle positions



# Leg Movements

...but *how* should the feet be moved around?

# Leg Movements



- » We determine spots on the ground where the feet land – the “footprints”
- » The foot takes a step from one footprint to the next

# Footprint Prediction

At the ***stance time*** (the start/end of the leg cycle) the foot is at its ***stance position*** (the “resting” position of the foot in character space).

- » Predict where the character is (the transform) at the next *stance time*.
- » Multiply the *stance position* with the predicted character transform.
- » Result: The next ***footprint***  
(The next place where the foot lands on the ground)
- » Keep updating the footprint prediction until the foot is planted

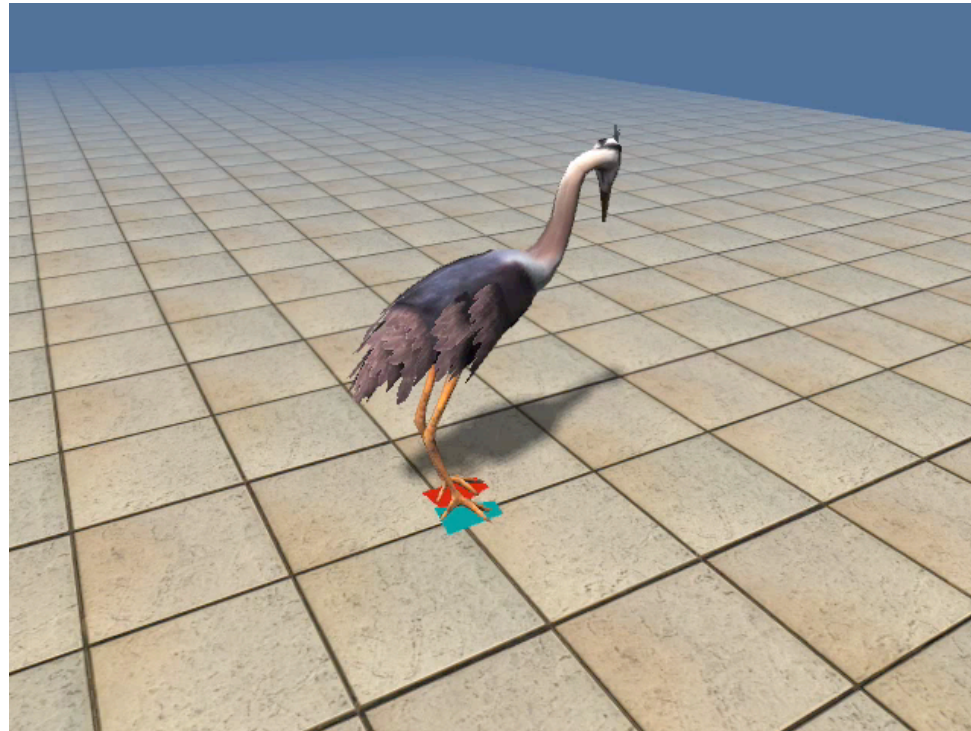


# Footprint Prediction



- » For each foot, the next footprint is predicted according to current velocity and rotational velocity.
- » The prediction is updated until the foot is planted.
- » Raycasts are used to place the footprints on the ground.

# Footprint Prediction



- » Prediction is individual for each leg.
- » Works for any number of legs.
- » Everything derived from sample animations.

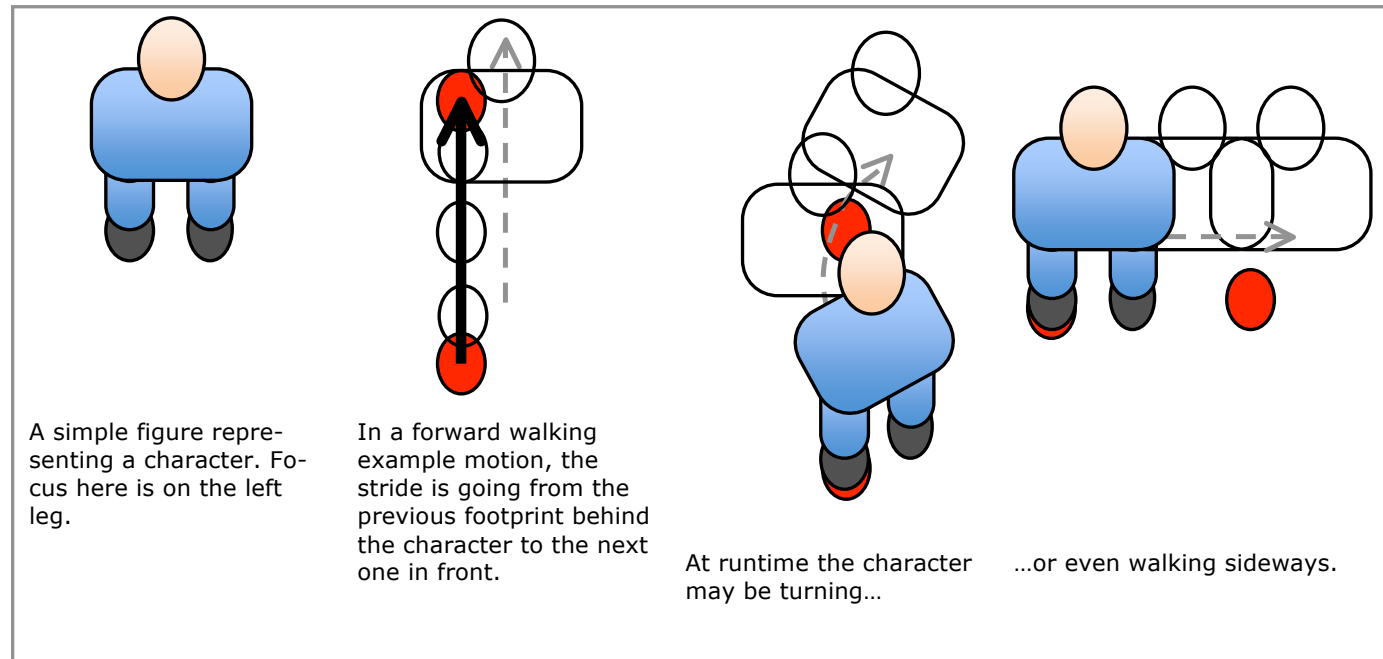
# Foot Flight Between Footprints

The foot must move from the *prev* to *next* footprint with a proper trajectory.

- » Proper horizontal curve that follows character...
- » Vertical lift that fits the (uneven) terrain...
- » If supporting leg is higher than lifted, lift to that height...
- » Preserve trajectory movements from original motion...



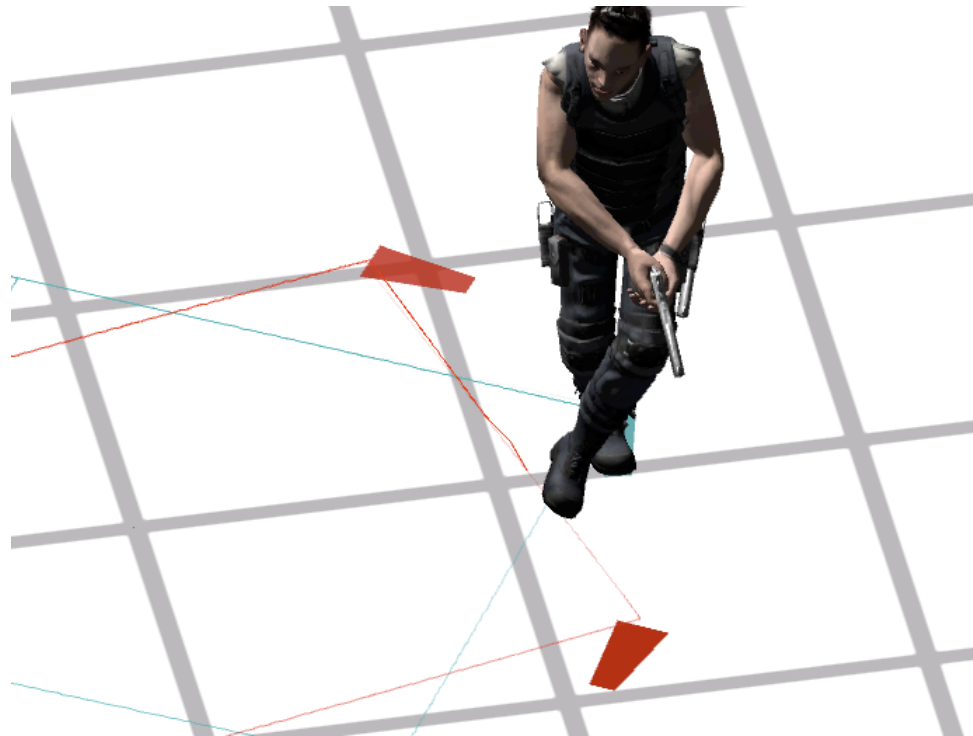
# Proper Horizontal Curve?



Need trajectories from *prev* to *next* footprint that work great no matter if adjusted velocity and curvature is completely different than in blended sample animations.



# Proper Horizontal Curve?

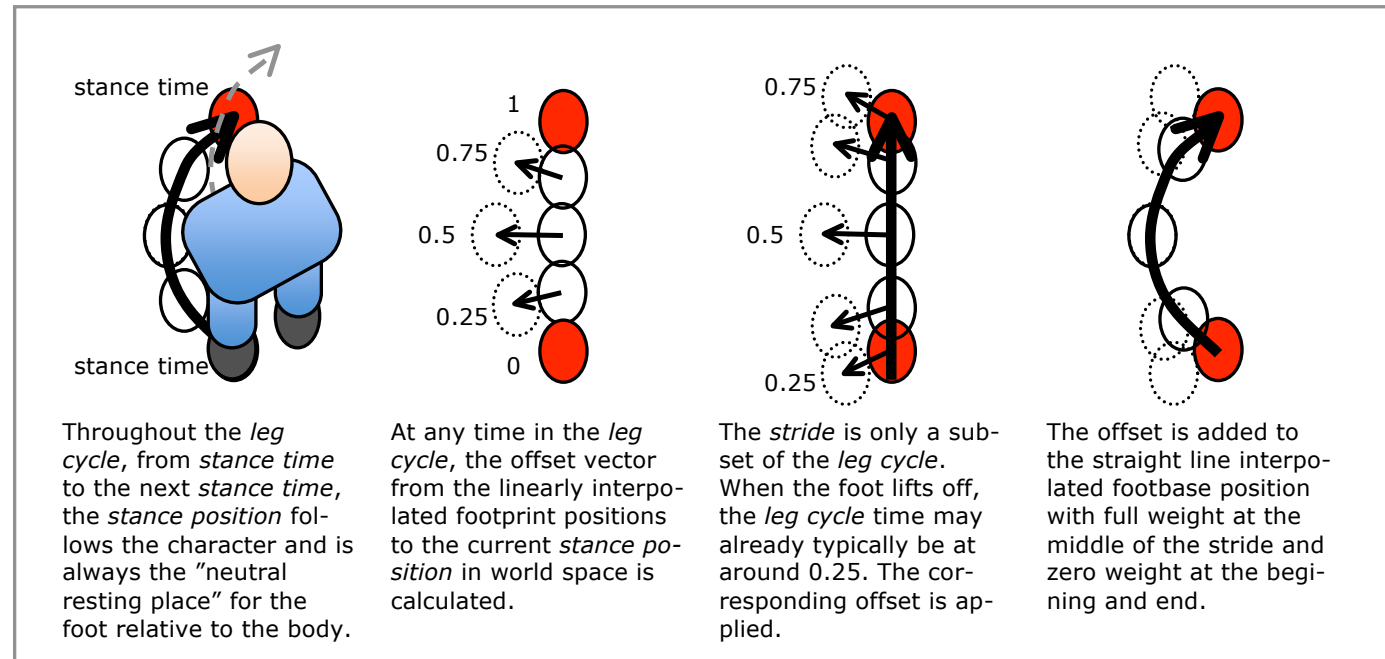


- » Use normalized footbase trajectory to move foot from *prev* to *next* footprint.
- » Doesn't work in a straight line though!

# Proper Horizontal Curve?

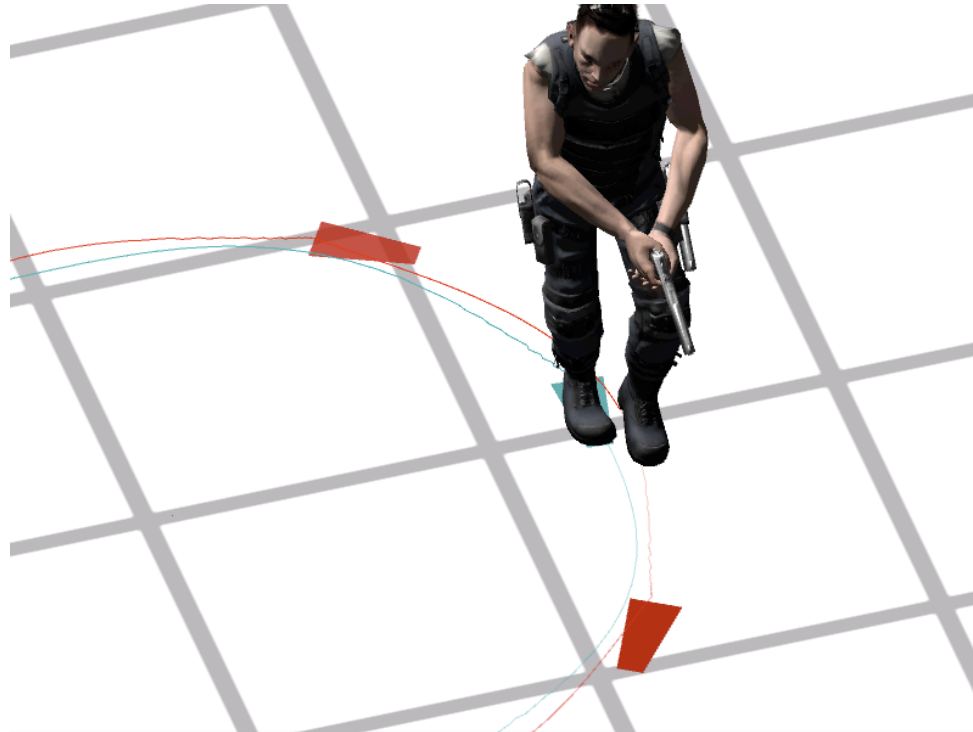
- » We also can't simply lerp into the original foot motion midway through the flight.
- » Gives wrong deltas if velocity at runtime is very different than velocity of blended sample animations.
- » (Footbase path from a forward walking animation applied to a sideways walking character will result in strange foot curves, even when only applied in mid-flight.)

# Proper Horizontal Curve!



- » Use straight line trajectory as a basis.
- » Add offset based on current stance position relative to lerped position between prev and next footprint.

# Proper Horizontal Curve!



- » Nice smooth curve for any adjusted movement.
- » Acceleration ("ease in / ease out") is still based on normalized footbase trajectory.



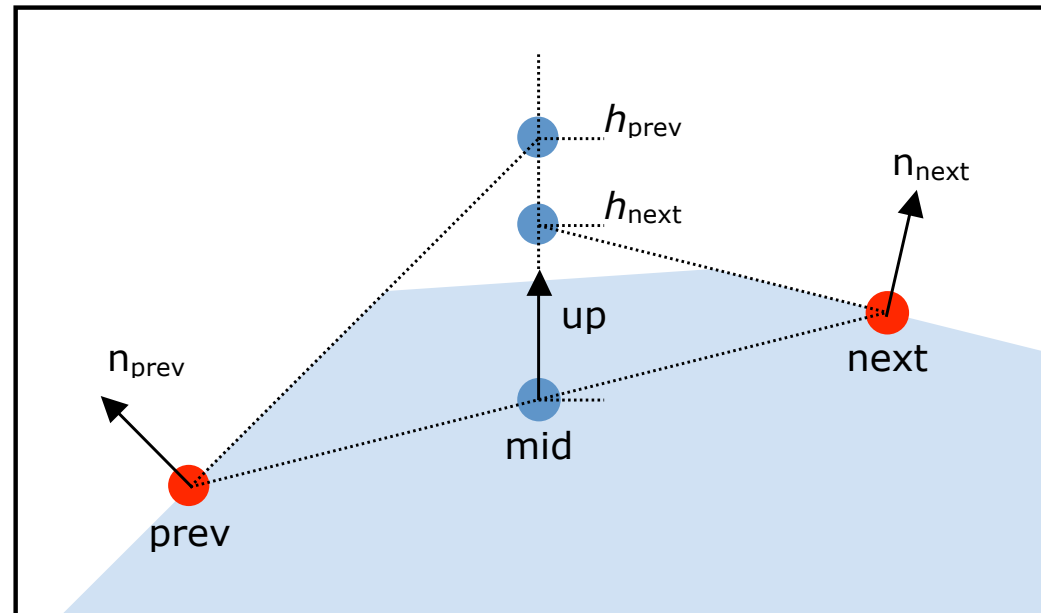
# Adjusted to New Velocities



- » Flexibility of foot trajectories make it possible to adjust animations to work for completely new directions.
- » (May not always look natural)
- » But highly useful for turning.

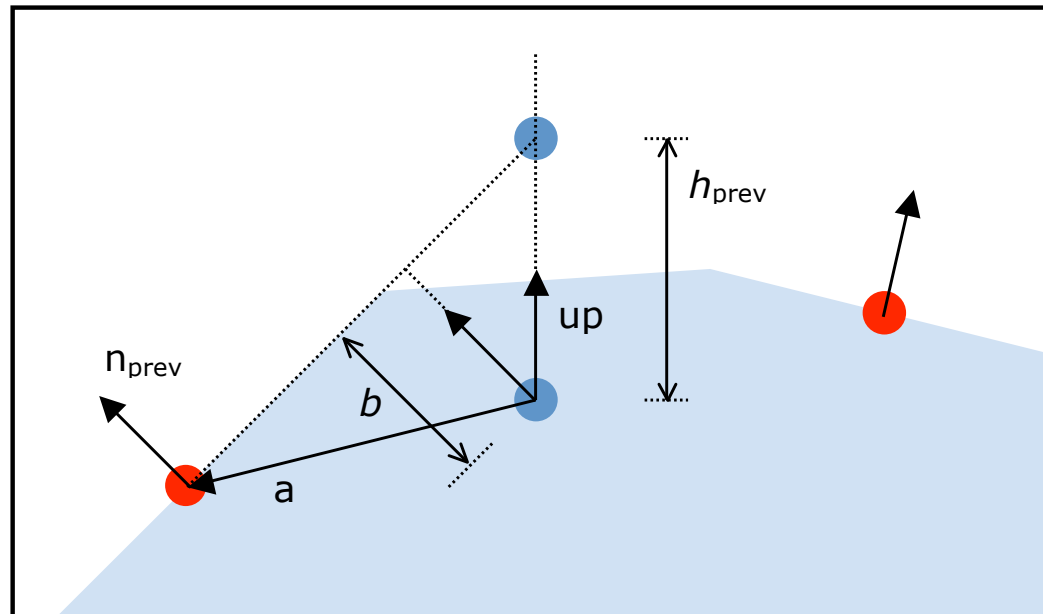
# Vertical Lift Fitting the Terrain

- » The foot should follow the terrain roughly, but in a smooth way.
- » Look at tangents of ground at prev and next footprint.
  - ⊗ Can be derived from ground normals at the footprint positions.



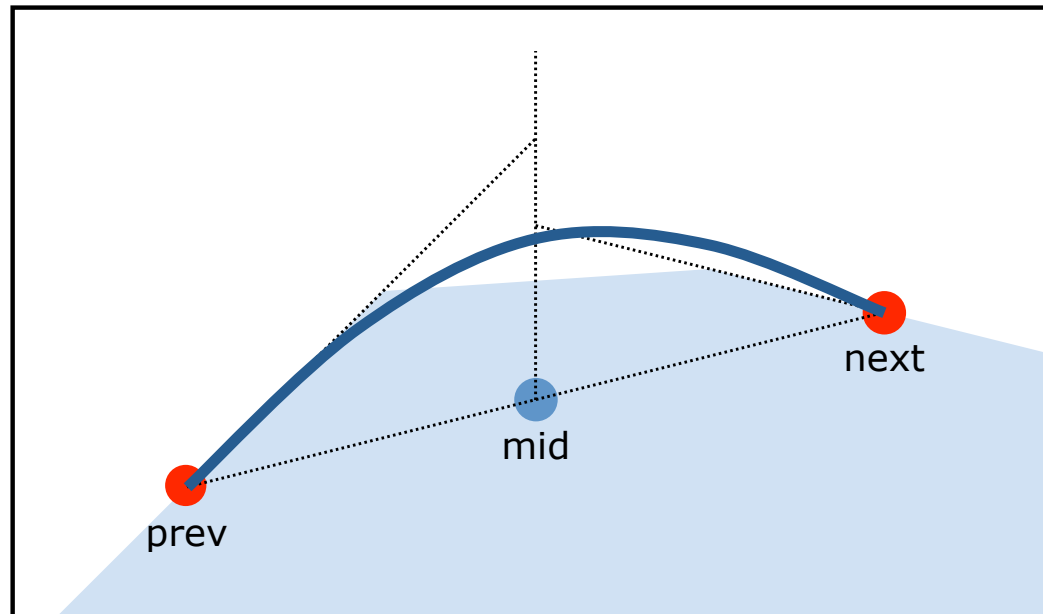
# Vertical Lift Fitting the Terrain

- » Find the intersection of tangent and vertical axis above mid-point between prev and next.
- » Use the intersection point with the highest altitude of the two.



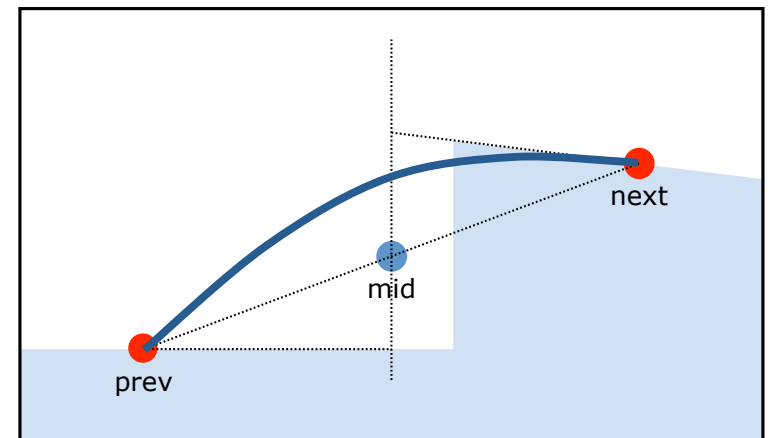
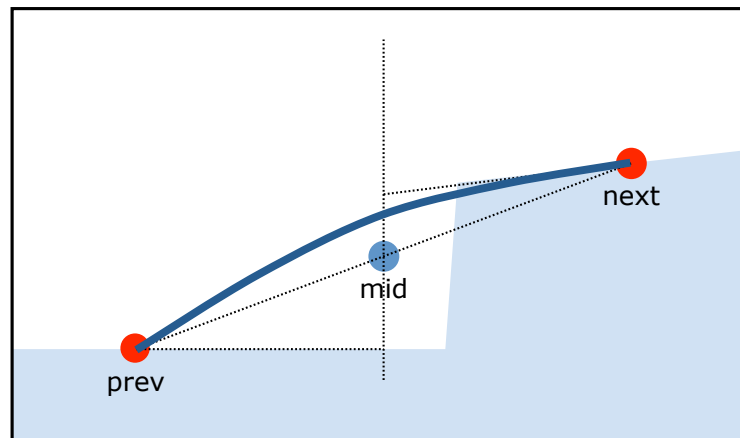
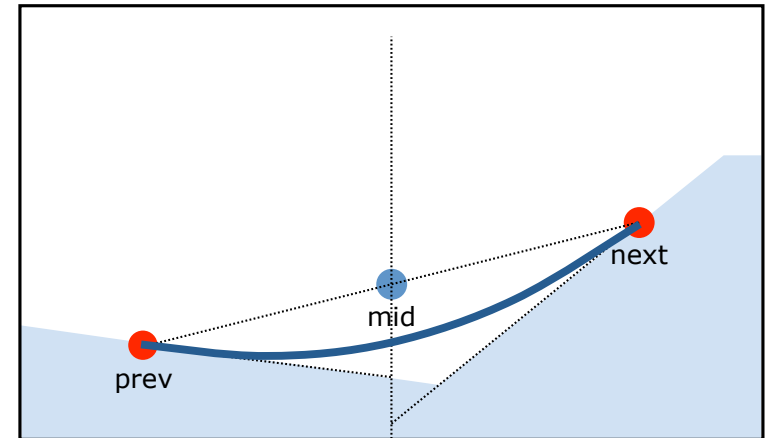
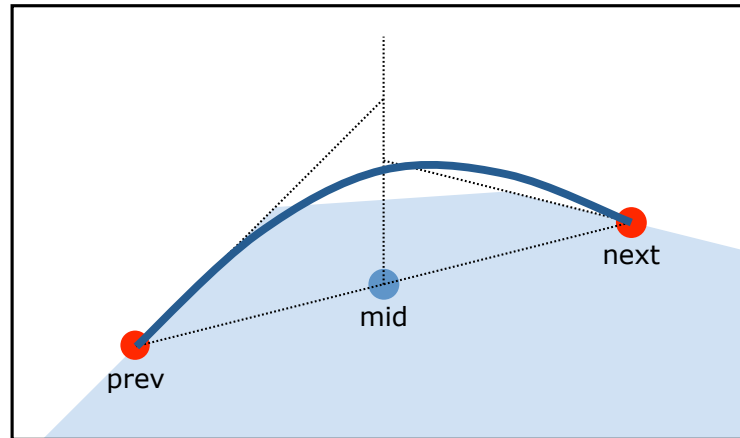
# Vertical Lift Fitting the Terrain

- » Use the height of the intersection point relative to the mid-point to determine the magnitude of an arc that is added to the footbase trajectory.
- » The arc follow the "highest" tangent.



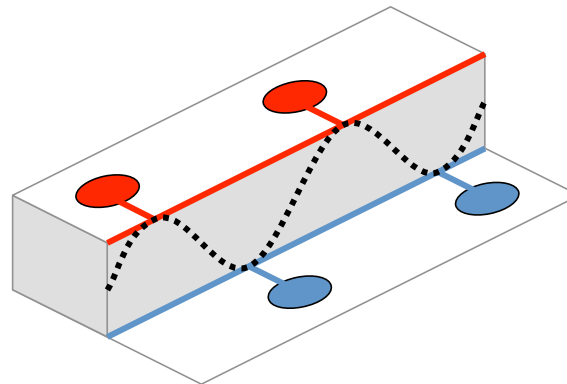
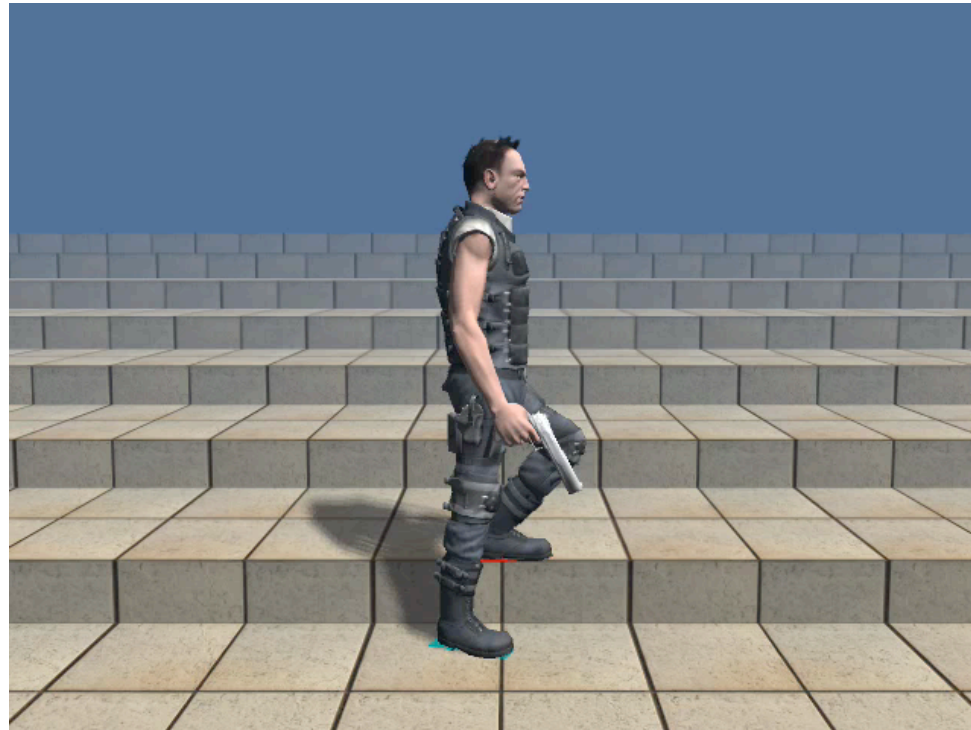


# Vertical Lift Fitting the Terrain



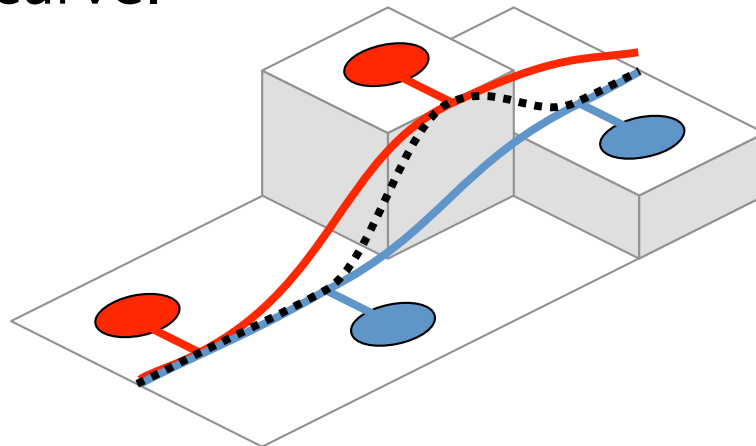
(This is the base trajectory. The original lifting is later added on top.)

# Lift to Height of Supporting Leg(s)



# Lift to Height of Supported Leg(s)

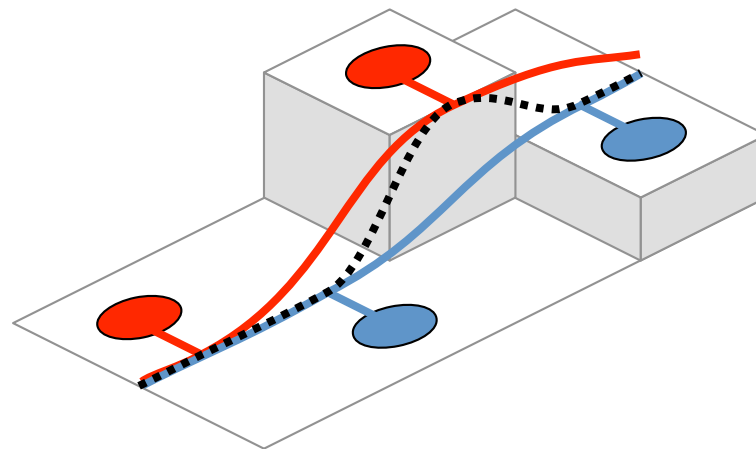
- » Calculate "basis height" of character:
- » For each foot, create a smooth curve going through the footprints.
  - ⊗ Minus offset of stance position.
- » Make weighted average of those curves:
  - ⊗ Grounded feet have full weight.
  - ⊗ Feet in mid-flight have almost zero weight.
- » Measure the height at the current position on the curve.



# Lift to Height of Supported Leg(s)

For each foot:

- » If the basis height is higher than this foot while in flight, lift the foot up:
- » From foot-off, lerp the lifting in until it has full influence at mid-flight, then back to zero influence at foot-strike.





# Lift to Height of Supporting Leg(s)



- » Advantage:  
Does not assume specific number of legs

# Preserve Original Trajectories

- » All the trajectory calculations so far were just to take uneven terrain and non-constant velocity / turning into account.
- » (When walking on plain horizontal surface, they result in just a ***straight line*** trajectory for the footbase.)
- » Now, add the ***normalized footbase trajectories*** (from the motion analysis) as additional offset.
- » The style is preserved!



# Foot Alignment

- » Now the footbase trajectories are calculated, but we still need to find good foot *alignments*
- » Feet move differently on a flat surface than they do on e.g. stairs
- » So alignment of feet cannot be used directly from animation data

# Foot Alignment

When the foot is flat on the ground:

- » Keep original ***alignment relative to ground.***

When the foot is lifted in the air:

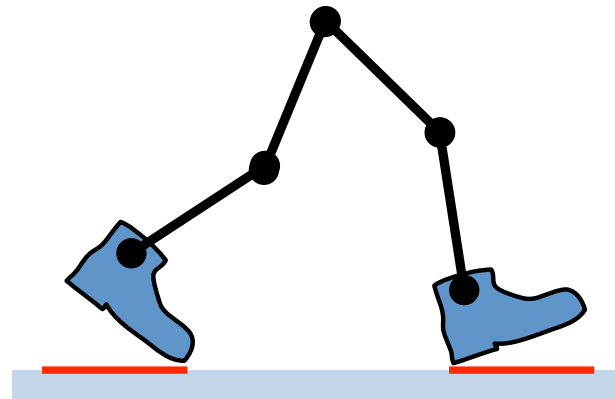
- » Keep original ***ankle joint rotation.***

Transition smoothly between these two states.

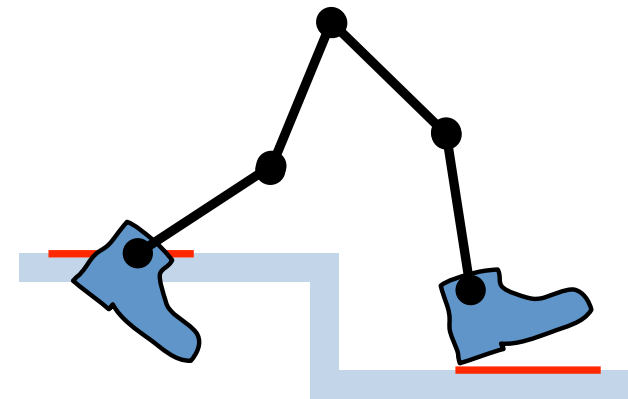


# Foot Alignment

**Pose in original motion**



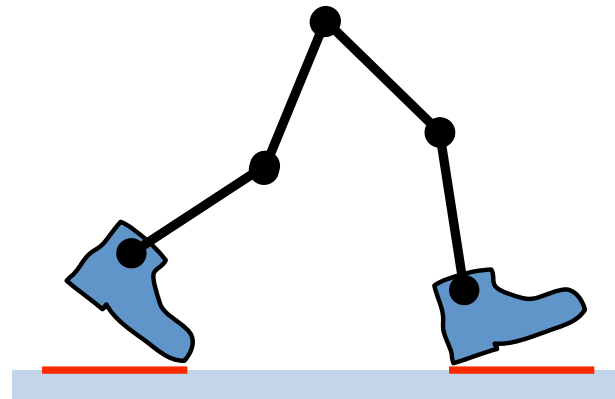
**Adjusted pose at runtime**



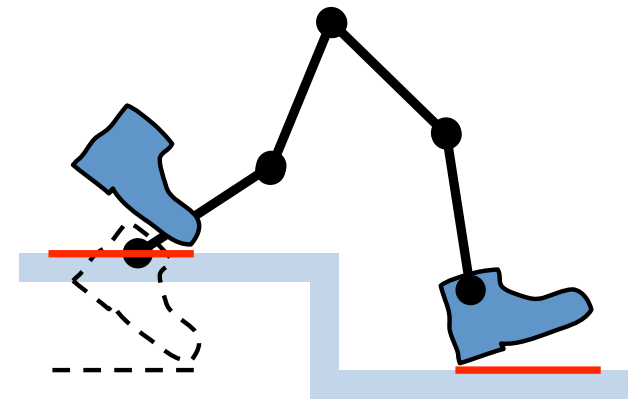
Need to have proper foot-roll on uneven terrain.

# Foot Alignment

**Pose in original motion**



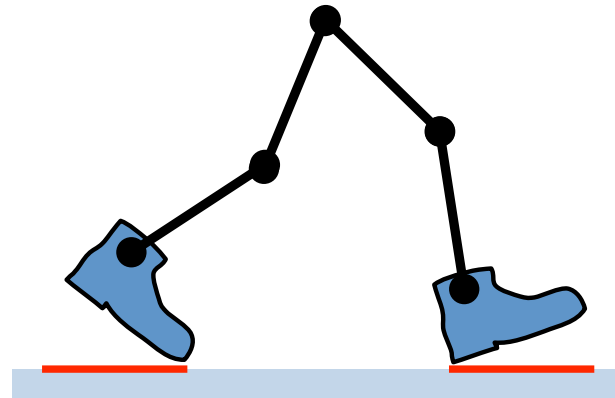
**Adjusted pose at runtime**



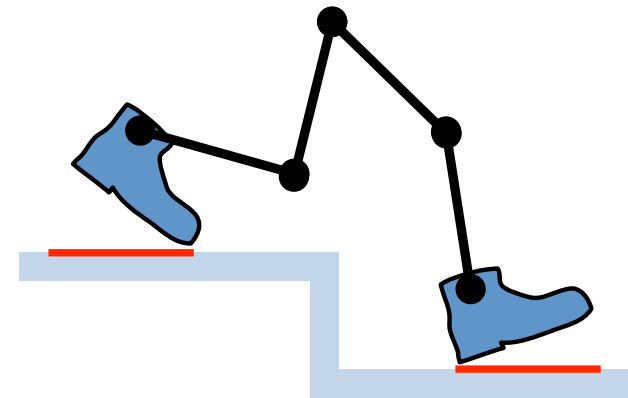
At runtime, move the foot to its footbase.

# Foot Alignment

**Pose in original motion**



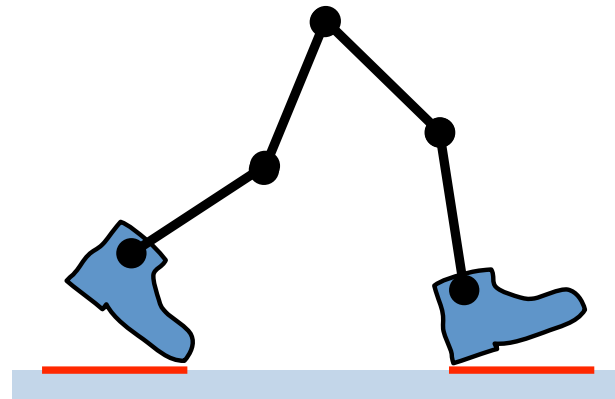
**Adjusted pose at runtime**



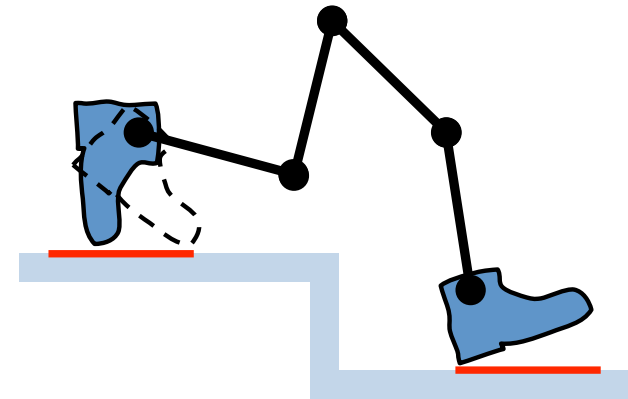
Use IK to get bone alignments between hip and ankle.

# Foot Alignment

**Pose in original motion**



**Adjusted pose at runtime**



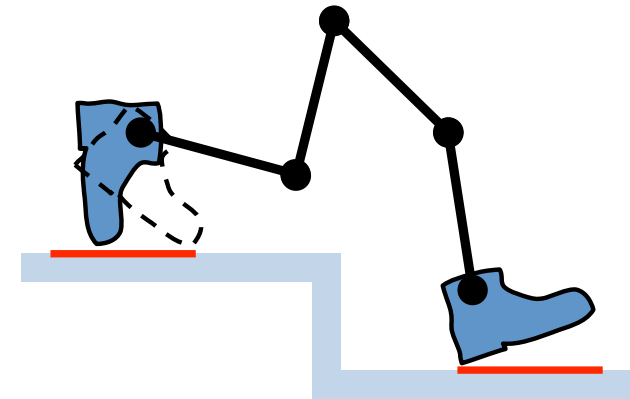
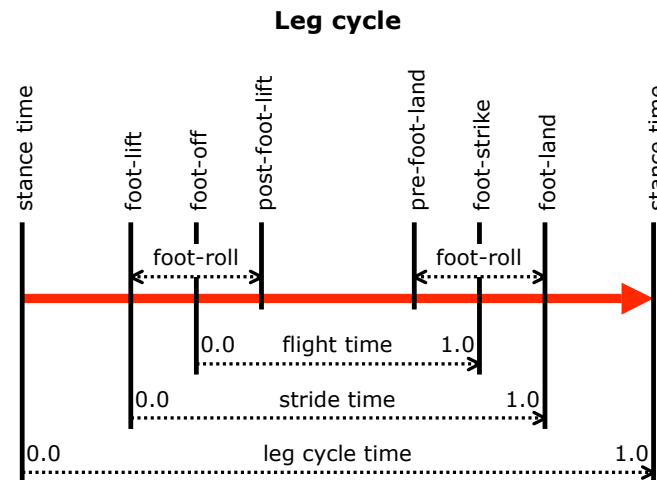
Apply the local rotation from the original motion at the ankle joint.



# Foot Alignment

**Pose in original motion**

**Adjusted pose at runtime**



Apply the local rotation from the original motion at the ankle joint.

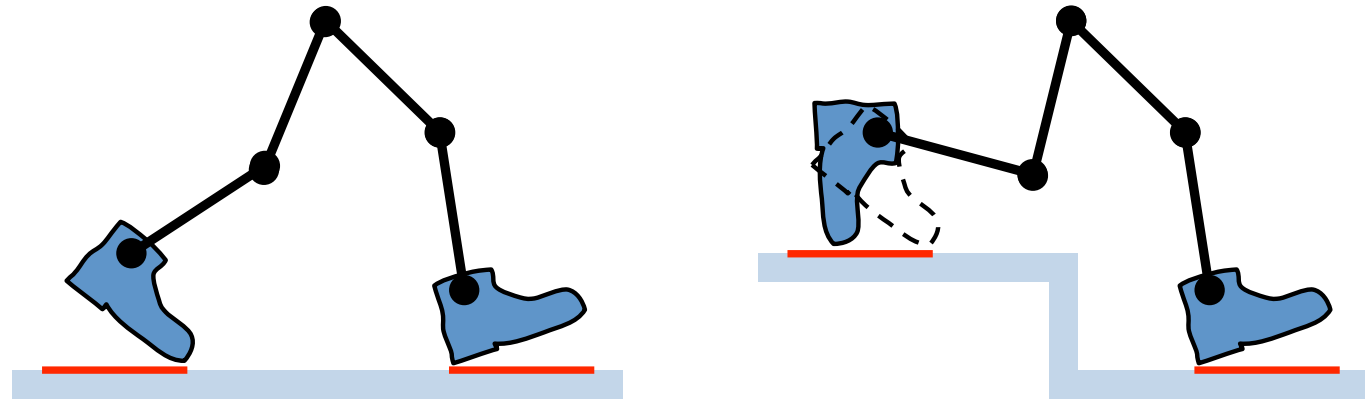
(0% when grounded, 100% when in flight.)



# Foot Alignment

**Pose in original motion**

**Adjusted pose at runtime**



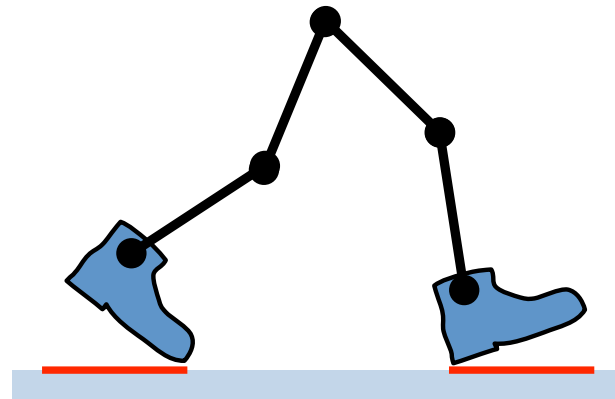
Apply the local rotation from the original motion at the ankle joint.

(0% when grounded, 100% when in flight.)

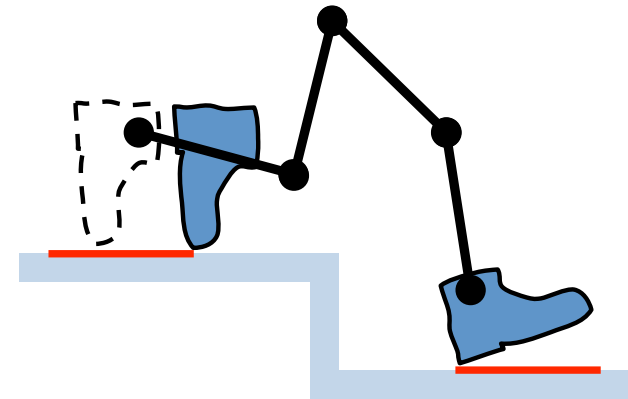


# Foot Alignment

**Pose in original motion**



**Adjusted pose at runtime**

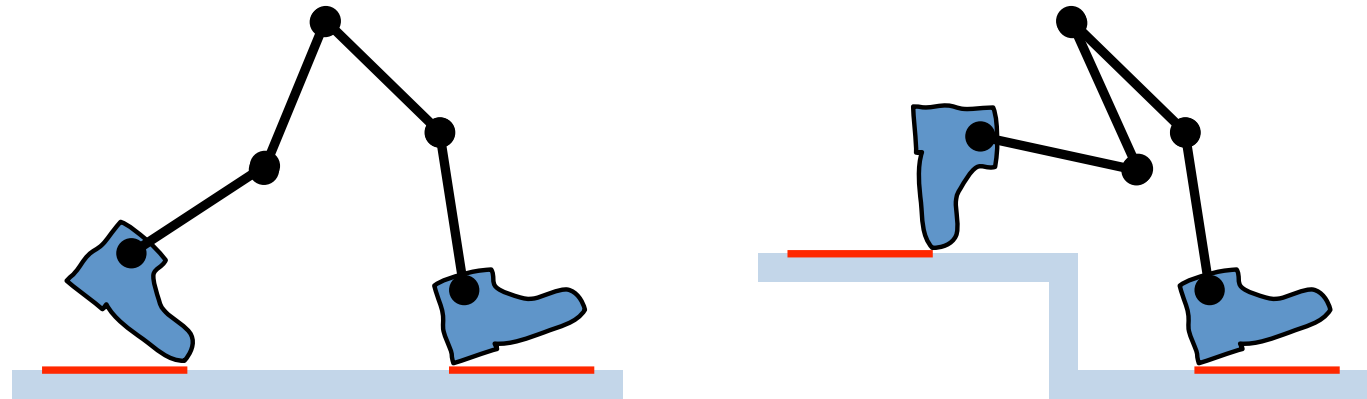


Move the foot to its footbase again.

# Foot Alignment

**Pose in original motion**

**Adjusted pose at runtime**



Use IK to get bone alignments between hip and ankle again.

**Done.** (Perform all steps in each frame.)





# Demo: Adapted Foot-Rolls



# Starting and Stopping Walking

- » Character may start or stop walking at any time
- » Fixed key-framed transitions are not flexible
- » Automatic transitions are desired

# Starting and Stopping Walking

- » Walking and running is a continuous cycle where the feet touch the ground one at a time
- » In idle animations both feet are grounded at the same time
- » How to transition?

# Starting and Stopping Walking

- » Each leg has its own *leg cycle*.
- » Normally the leg cycles all turn around together with the overall motion cycle.
- » But they **can** independently “stop turning” one at a time (or start).

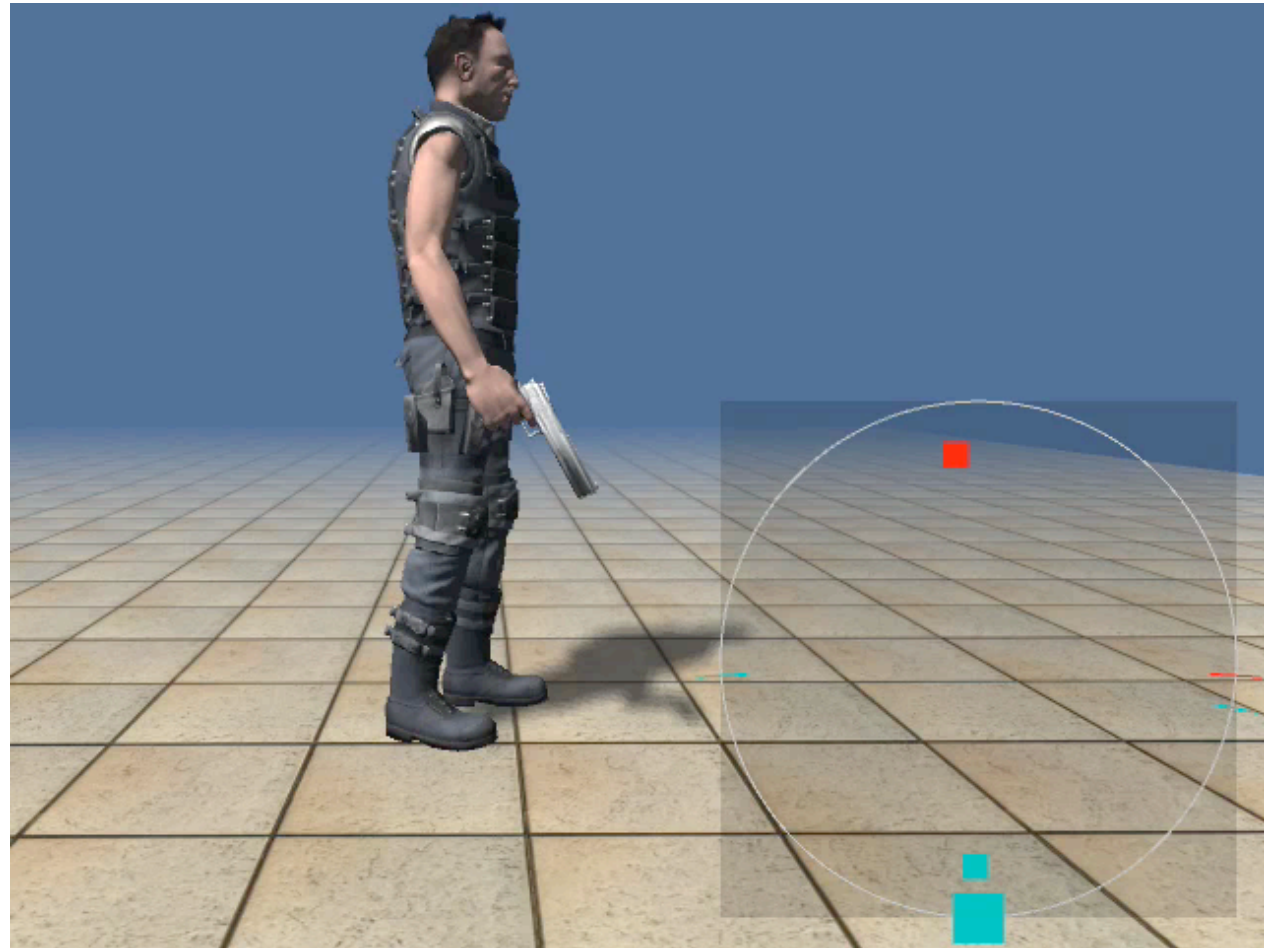


# Starting and Stopping Walking

For each leg:

- » If length of next step is below given threshold:
  - ⌚ "Park" the leg cycle next time it reaches its stance time.
- » If the leg is parked, and the next step is above given threshold:
  - ⌚ "Unpark" the leg cycle the next time the "would be" leg cycle value passes the stance time.
- » Maybe allow "catching up" to make mechanism less rigid.

# Demo: Automated Transitions



# Interface and Integration

How to *integrate* the system  
with other animation systems used  
in the game?

# Controlling Styles

- » Blending of idle, walking, running animations is automatic
- » But how to control *style*?



# Styles Through Animation Groups

- » Animation Groups are collections of animations for locomotion.
- » Each group contains up to several animations with the same style but with different velocities.
- » Example:
  - ⊕ Group "normal" - normal idle + walk + run
  - ⊕ Group "sneaky" - sneaky idle + walk

# Animation Groups

- » Each animation group can be controlled as if it was a single animation

```
animation["normal"].weight = 0.8f;  
animation["sneaky"].weight = 0.2f;
```

```
animation.CrossFade("sneaky", 0.5f);
```

# Current Limitation in Motion Types

3 types of motions

## » **Walk / run cycles**

- ⌚ Walking / running with feet adjusted to uneven terrain

## » **Grounded animations** (feet are not moving)

- ⌚ Feet still adjusted to uneven terrain below

## » **Everything else**

- ⌚ Not directly supported at this point / feet not adjusted
- ⌚ But adjustments are gracefully (gradually) turned off
- ⌚ E.g. jumping: Adjustments turned off while in air



# Performance

Reference machine: 2.4 GHz Intel Core 2 Duo

## Biped:

**0.25** ms per frame

1 biped:

100 -> 98 fps

50 -> 49 fps

10 bipeds:

100 -> 80 fps

50 -> 44 fps

## Quadruped:

**0.55** ms per frame

1 quadruped:

100 -> 95 fps

50 -> 49 fps

10 quadrupeds:

100 -> 65 fps

50 -> 39 fps



# Many Characters



# Many Characters





Quadrupled animal models  
(coyote, grizzly bear, wolf)  
courtesy of WolfQuest.org

# Thank you!

## Questions...?

» ...

» Email: [rune@unity3d.com](mailto:rune@unity3d.com)

## Resources

- » Locomotion System for Unity  
[unity3d.com/support/resources/](http://unity3d.com/support/resources/)
- » Unity Game Engine  
[unity3d.com](http://unity3d.com)
- » The Master Thesis behind the system is coming soon...  
[runevision.com/multimedia/](http://runevision.com/multimedia/)

