
Welcome to the Advanced Look Component documentation!

by Kinemation

This document covers how the component works, how you can use the asset, and how you can expand it.

How it works

The ALC (Advanced Look Component) includes three scripts:

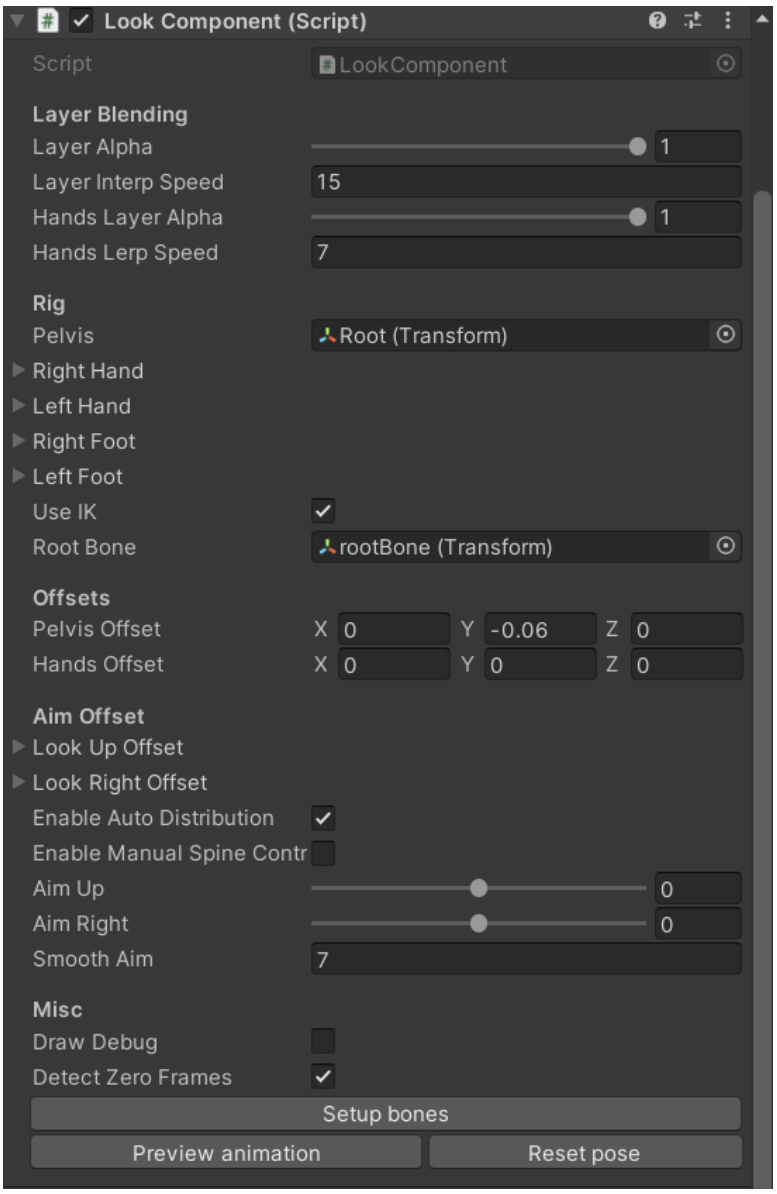
- LookComponent.cs
- AnimToolkitLib.cs
- LookComponentEditor.cs

LookComponent is what you need to attach to your character to get things working!

AnimToolkitLib is a helpful library when it comes to procedural animation. It includes functions for translation/rotation of bone in other bone space, frame-rate independent interpolation, and Two-Bone IK function.

LookComponentEditor is a customized inspector for LookComponent. Nothing special.

LookComponent is a script that makes the character look up/down, right/left procedurally. It doesn't affect the base animation layer, so you can reload/shoot/punch, and **look around without affecting your animations!**
Now, let's take a closer look at the component itself:



Overview

Layer Blending

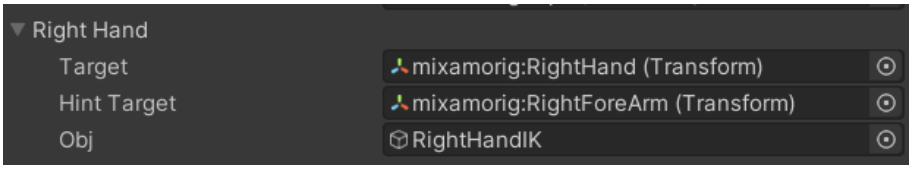
Layer Alpha defines the weight of the applied procedural layer: 1 - 100% effect, 0 - no effect. *Hands Layer Alpha* - 0 means base animation layer, 1 means with a procedural offset applied. By default, hands IK targets are parented to the head, this can be a problem when you have an unarmed motion, like this one:

Goblin lookin' up

So, in this case, you can decrease the Hands Layer Alpha to something like 0.15-0.2 so you still have some effect on the arms.

Rig

Right Hand, Left Hand, Right Foot, Left Foot - are dynamic bones:



Target is the actual bone the bone whose rotation&location is going to be copied by the dynamic bone.
Hint target is a pole target used for the Two-Bone IK function. By default, it's the parent of the Target
Obj is an empty GameObject, so you can easily drag or reparent it easily in the inspector.

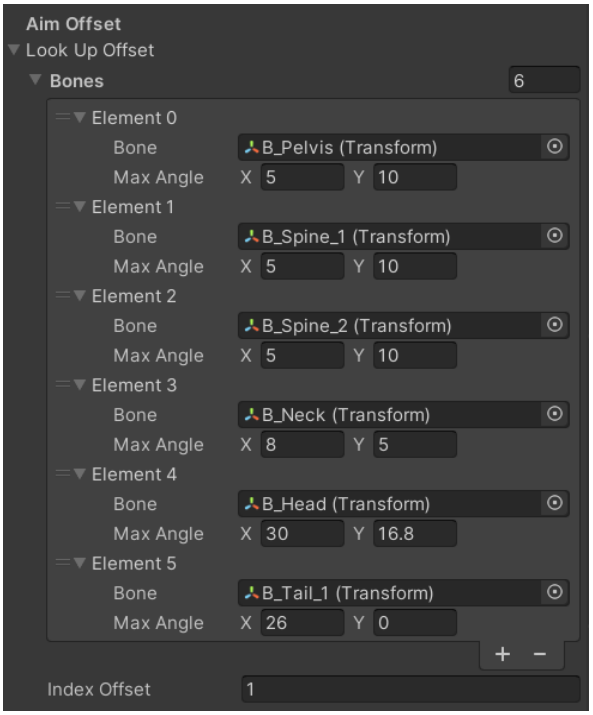
You can disable IK, if you want to modify spine bones only. However, **using IK is strongly recommended to achieve more natural results!**

RootBone is an empty GameObject used for bones rotation/translation. All operations are done in the space of the RootBone, which makes it possible to apply an additive offset to arms, spine, and hips without affecting the base animation layer.

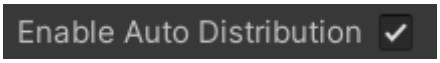
By default, RootBone is created as a child of the object LookComponent is attached to. The Offsets category allows you to add a translational offset to hips and arms.

Aim Offsets

AimOffset contains 2 lists of bones that are used for aiming.



Every element contains a reference to the bone transform, and maximum look angles: X is the max look Up/Right, Y is the max look Down/Left.
Editing each bone is a very boring task, so you can automize it by enabling this flag:



So whenever you edit the higher element, the rotation of the other lower elements will be adjusted automatically.
If there're bones, which you don't want to adjust you can use the Index Offset property. This offset defines the number of elements (from the end) that won't be automatically adjusted.

Example from above: Goblin character has a tail, so we want it to be affected by aim offset, so just add the tailbone to the list and set Index Offset to 1 - now it's not going to be changed by auto distribution.

Enable Manual Spine control is useful only in Play mode.

Workflow

After importing the ALC package just add LookComponent to your character:
After that click on "Setup bones", this will automatically create the necessary objects and find bones.

If something went wrong, you will see a message in the logs.

Keep in mind, that even if all bones were found this doesn't mean that the ALC found the right bones :)
So it's better to check if bones are assigned properly.

P.S. works perfectly with Mixamo rigs.

There're no restrictions regarding the bone orientation or your rig: the system works with both Generic and Humanoid rigs, but keep in mind, that arms and legs should be a chain of 2 limbs because a Two-Bone IK is used for legs/arms.

API

In order to modify the ALC via C# scripting, there're 5 essential methods you need to use:

```
public void SetAimRotation(Vector2 newAimRot) - this is user input, X - horizontal Y - vertical movement

public void SetHandsOffset(Vector3 offset)

public void SetHipsOffset(Vector3 offset)

// 0 - no effect, 1 - fully applied
public void SetBlendAlpha(float alpha)

// 0 - Hands aren't affected by spine/neck/head rotations
// 1 - Hands are fully affected by spine/neck/head rotations
public void SetHandsBlendAlpha(float alpha)
```

How to extend asset functionality?

Before adding new functionality to the component or even creating your own, it's essentially important to understand the core concepts.

The ALC is powered by dynamic IK retargeting and runtime bone modification.

Let's start with retargeting, it's pretty simple actually: we create empty objects (IK targets) for hands and feet, then get the rotation/location of hands/feet and set them to the empty objects.
Then we can add offsets and apply more complex motions using empty objects.

This allows us not to break the base animation layer, so it's possible to look around and reload/shoot/punch at the same time!

Runtime bone modification is performed by rotation/moving bones in the RootBone space. Simple as that.

Here's the order of operations:

1. Perform dynamic retargeting
2. Apply procedural offsets

3. Apply IK

All these things must be executed after the Animator update, in LateUpdate specifically. Note, that you will need to update CheckZeroFrames() and CacheBones() methods in order to use Zero-Keyframe detection properly - this is applied for everything that is additive, like rotations or translations.