
Welcome to the SightsAligner documentation page!

by Kinemation

This document covers how the system works and how to integrate it into your project.

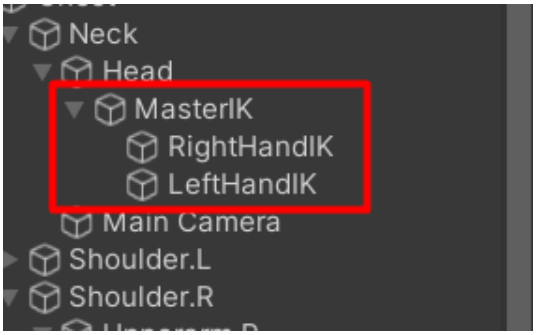
[How it works](#)
- [Basic principles](#)

[The workflow](#)
- [Step 1](#)
- [Step 2](#)
- [Step 3](#)
- [Step 4](#)
- [Step 4](#)

How it works

Basic principles

SightsAligner is based on dynamic retargeting - empty objects copy animation data of hands in runtime, then, we apply procedural layers and don't affect base animation at all.



Dynamic IK targets

The system uses 2 types of aiming:
- Additive: animations are fully applied
- Absolute: entirely overrides animation so sights are aligned

The absolute approach is fully automatic and doesn't require additional setup.
The additive approach requires aim pose data calculation as it simply adds translation&rotation when aiming.

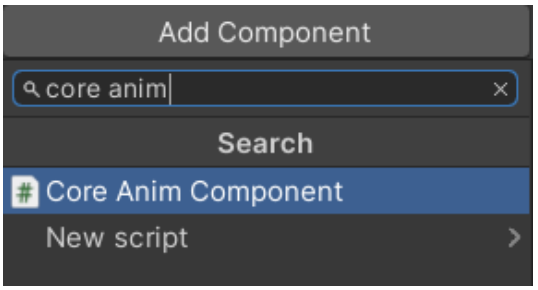
CoreAnimComponent - this is a procedural animation system that does all the work.

```
public void CalculateAimData() // This is used for calculating the aim offset between the aim point and the aim target. Used for the additive approach

public void SetupBones() // Used in the editor only

public void Init(GunAimData data, Transform aimPoint) // Must be called when a gun is equipped

public float aimLayerAlphaLoc; // How much animation (translation) should affect aiming? 0 - no effect, 1 - fully applied

public float aimLayerAlphaRot; // How much animation (rotation) should affect aiming? 0 - no effect, 1 - fully applied

public GunAimData aimData;
```

The workflow

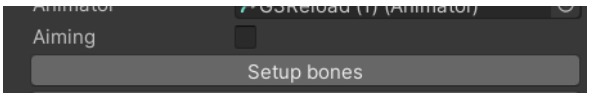
Step 1

Add CoreAnimComponent to the character



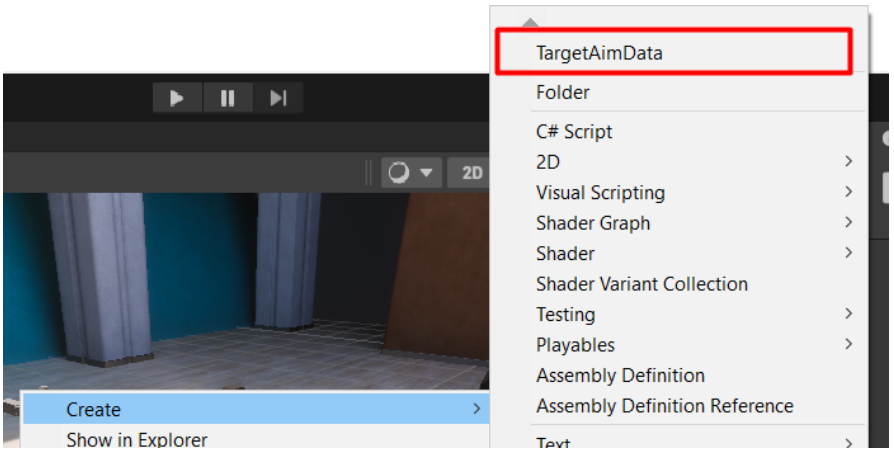
Step 2

Set up bones and make sure bones are assigned properly. **NOTE: RightHandIK and LeftHandIK should be parented to the MasterIK**



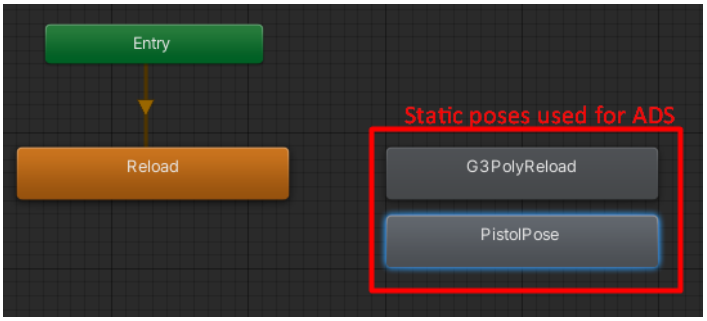
Step 3

Create a new **TargetAimData** and assign the static pose clip - only the first frame of the animation will be used, so you can even assign a reloading animation here.



Note: you need to create this SO[1] for each gun

Then add an empty state to your animator: when calculating bone data it's important to play the static pose.



Also, make sure that the name of the state and animation clip is **THE SAME!**
You can specify a custom name by editing a string field: if the **stateName** string is empty, the animation clip name will be used instead.

Step 4

Add **CoreAnimComponent** reference to your weapon system class.

Make sure that Init() is called when a new gun is equipped:

```
private void EquipWeapon()
{
    ...
    animComponent.Init(gun.gunAimData, scopeTransform);
}
```

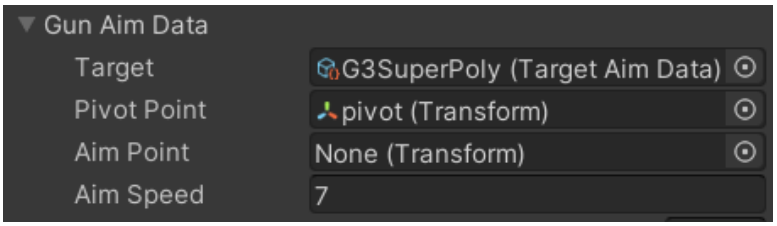
[1] Scriptable object

Then, make sure that you update the current aim point transform when cycling sights:

```
    private void Update()
     {
          ...
          //
          if (Input.GetKeyDown(KeyCode.V)) <- changing sights/scopes
          {
               animComponent.aimData.aimPoint = GetGun().GetScope(); // update
scope
          } …
```

The demo project contains examples of weapon and weapon system classes, so you will have a practical example of how the sights aligner can be applied.

Add **GunAimData** to your weapon and set it up:

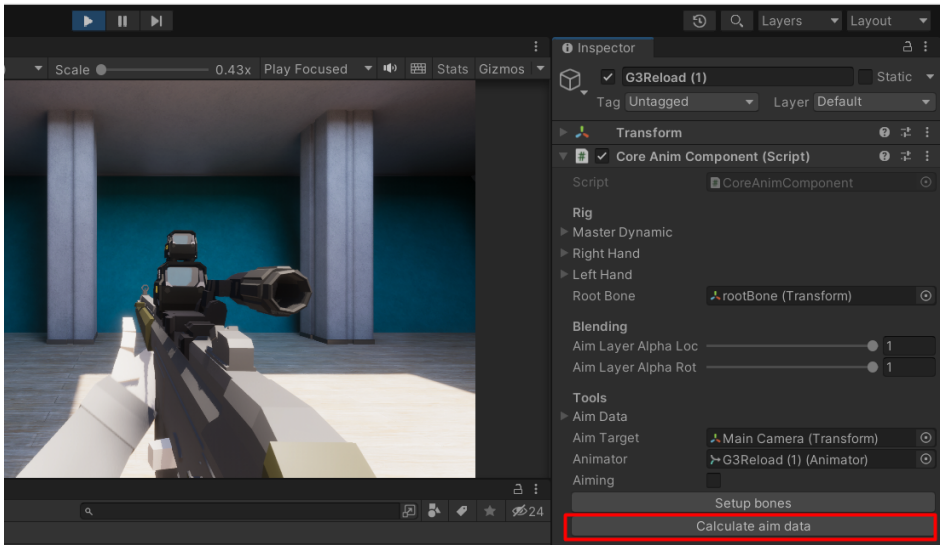


Target - TargetAimData (ScriptbaleObject, contains Pos&Rot and static pose anim clip)
PivotPoint - an empty object, that can serve as a default aim point. Make sure it has the right rotation: X is the forward, Y is the up etc.
AimPoint - should be none as default. This is modified in a runtime.

Step 4

Note: This step is only useful if you want to preserve base anim layer

Calculate aim target data in play mode when a weapon is equipped:



If all the above steps were followed correctly, this will write aim translation and rotation based on the static pose clip.

In fact, you can automate the process, here's an example:

```
private void Update()
 {
      ...
      if (Input.GetKeyDown(KeyCode.SomeKeySode))
      {
          foreach (var gun in weaponPrefabs)
          {
              ChangeWeapon();
              animComponent.aimData = gun.gunAimData;
              animComponent.CalculateAimData();
          }
      }
 }
```

All you have to do is to iterate through all the weapons, update the aimData and calculate new aim data.