
Welcome to the Recoilly documentation page!

by Kinemation

This document covers how the system works and how it can be applied.

How it works

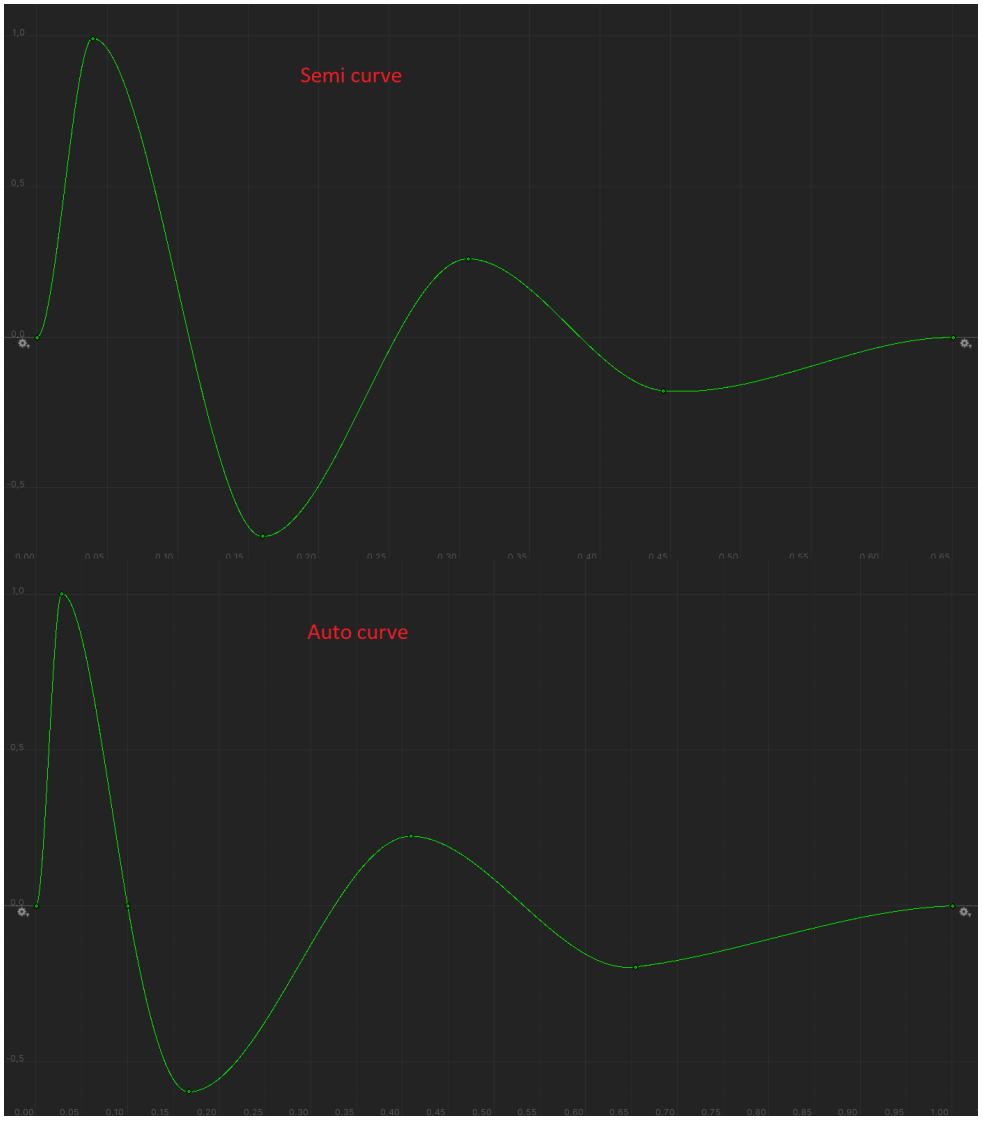
RecoilAnimation.cs

Recoilly is based on Unity Animation Curves. The formula is pretty simple:

Animation Value = LerpUnclamped(0, Randomized Value, AnimationCurveValue)
LerpUnclamped is used to achieve a bouncy effect (curve value less than 0).
All recoil curves must start and end with zero.

Curves for auto/burst weapons

Auto curves are actually just modified semi curves, here's an example:

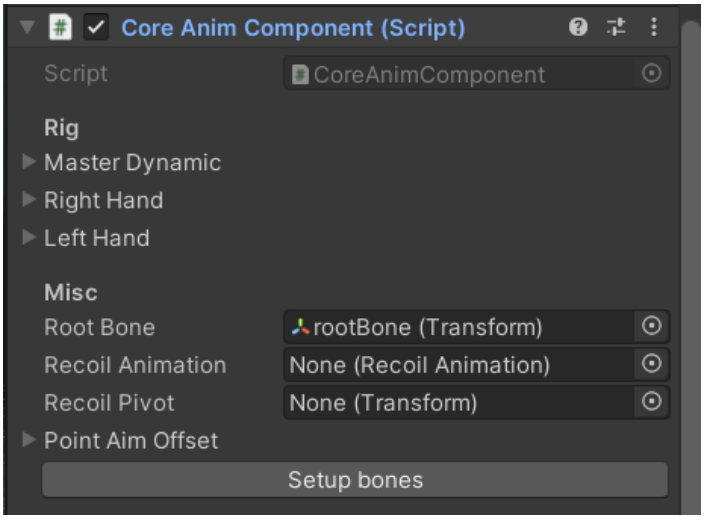


Curves are pretty much the same, **BUT the** Auto curve value is zero at some point - this point is the delay between shots. Let's say the fire rate is 600 RPM, this means a 0.1s delay between each shot. Consequently, our auto curve value should be zero at 0.1s, otherwise, glitches might be expected.

Why is that? Because Auto/burst animation gets looped and the animation max time is set to the delay between shots in seconds.

CoreAnimComponent.cs

This script is responsible for applying actual recoil to the character. It uses the Two-Bone IK function from the *AnimToolkitLib.cs*. Also, this component is used for applying pose correction and point aiming.



How to implement

Step 1

Add CoreAnimComponent to your character and click setup bones. Sometimes bones aren't set up correctly, so you better check that the bones are assigned properly.

Step 2

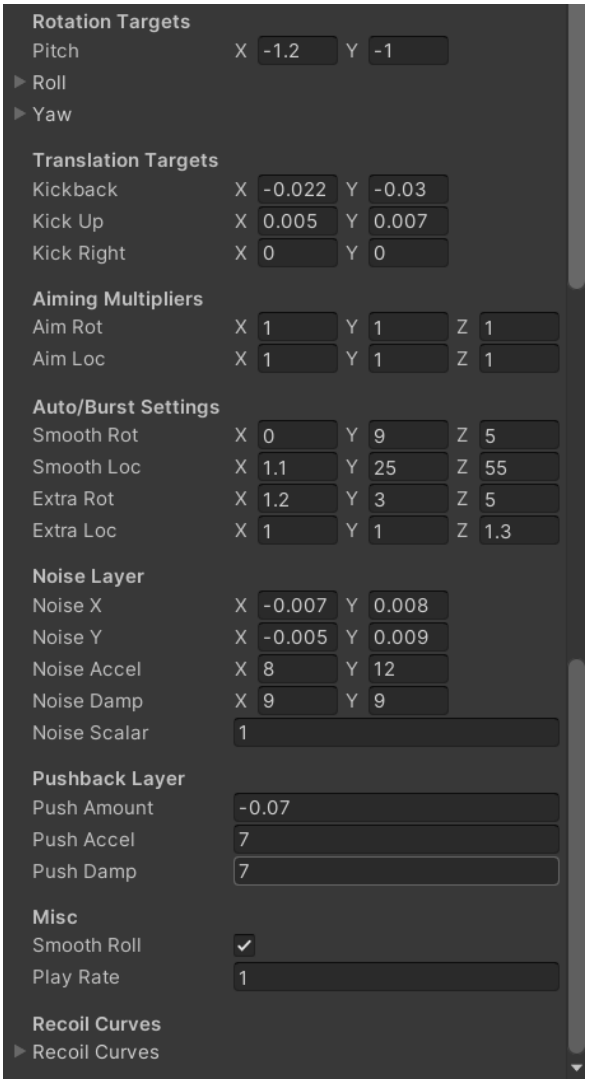
Add RecoilAnimation to your character. RecoilAnimation handles all the procedural animation logic based on the input parameters. Here're the most important methods:

```
        public void Init(RecoilAnimData data, float fireRate) // This should be called whenever a new weapon is equipped

        public void Play()
        public void Stop()
```

Step 3

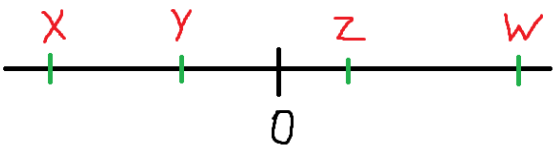
Add RecoilAnimData to your Weapon class. To create a new data asset: Left click/Create/RecoilAnimData.



Recoil data example

Pitch defines the min and max values of look up/down rotation. Roll defines the rotation around the Z(forward) axis. Yaw defines the rotation around the Y (left/right) axis.

Roll&Yaw are Vector4, here's why:



This is done in order to prevent getting a random value very close to 0, because 0 means no animation effect => or strange results.

Translation targets define maximum and minimum values.
Aiming multipliers are applied when aiming flag is set to 1.

Smooth Rot/Loc defines interpolation speed when firing in auto/burst mode.
Extra Rot/Loc are multipliers applied when firing in auto/burst mode.

Noise layer performs a smooth movement in the YX plane (left/right-up/down).
Noise scalar is used when aiming.
Pushback layer is a strong kickback after the first shot in full-auto burst mode.

Step 4

Call Init(), Play() and Stop()
This is when you need to integrate the package logic into your weapon system. Add a reference to the RecoilAnimation in your weapon system class (or any other class where you handle shooting/equipping).

Add Init(), Play() and Stop() to your code.

Step 5

Add CoreAnimComponent functionality to your weapon system class (or any other class where you handle shooting/equipping).

```
private void EquipWeapon()
 {
     var gun = weapons[_index];

     _recoilAnimation.fireMode = gun.fireMode;
     _recoilAnimation.Init(gun.recoilData, gun.fireRate);

    // Start
```

```
        _animComponent.recoilPivot = gun.recoilPivot;
        _animComponent.handsOffset = gun.handsOffset;
        _animComponent.pointAimOffset = GetGun().pontAimData;
      // End

        _bursts = gun.burstAmount;

        _animator.Play(gun.poseName);
        gun.gameObject.SetActive(true);
    }
```

Check the demo project for more info.