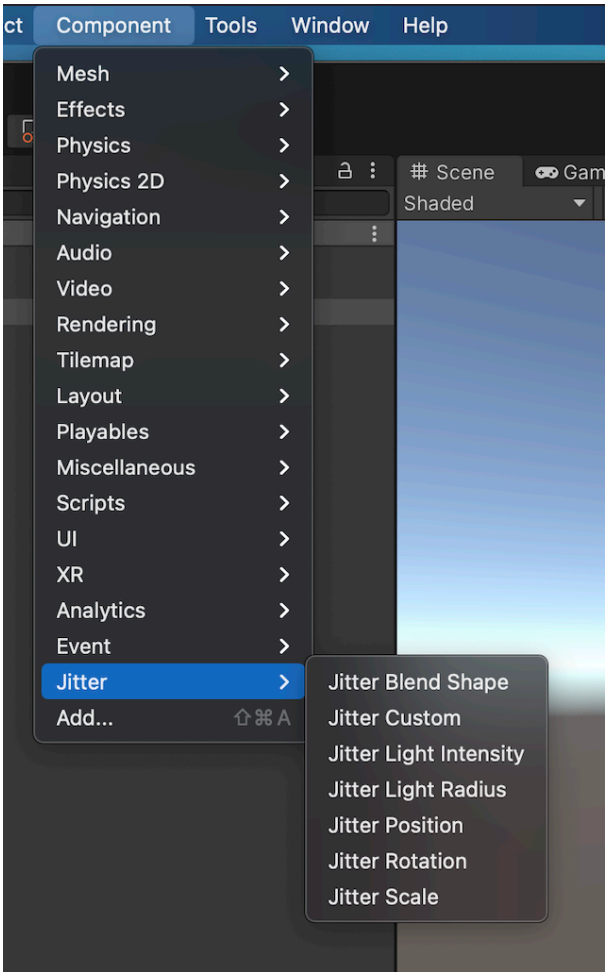
JITTER USER GUIDE

Introduction

Real life is noisy. To help make your Unity games more realistic, Jitter provides the ability to easily add animated noise to various properties of your GameObjects. Jitter is fully configurable and flexible enough to be used in a wide variety of applications.



Jitter provides a collection of components that let you choose what properties of your GameObject should be noisy, and it allows you to layer multiple components to create exactly the right kind of noisy motion you want. From simulating natural effects like the motion of fire and wind, to shakes and impact crashes, Jitter has you covered.

Jitter noise is non-destructive. This means you can, for example, add a Jitter Position component and are still able to move your Gameobject around. Rather than replacing your objects position values, the noise will be added to your objects position.

The various Jitter components can be found in the Component menu under Jitter. They are:

- Jitter Blend Shape
- Jitter Custom
- Jitter Light Intensity
- Jitter Light Radius
- Jitter Position
- Jitter Rotation
- Jitter Scale

Examples

If you want a quick demonstration of what Jitter can do, a good place to start is to look in Jitter/Examples/Scenes. Open any of these example scenes and hit the *play* button to see Jitter in action. The following basic examples demonstrate various aspects of the Jitter asset:

Handy Cam
Multiple instances of the Jitter Rotation component control the rotation of the main camera. This shows how Jitter components of the same type can be layered to create noise of different frequencies, leading to more dynamic movement.

Camera Shake
An obvious application. This example shows how Jitter can be controlled using Unity's animation system. If you open the Animation window you'll see that a property called Master Magnitude has been key-framed. This controls the magnitude of all Jitter Rotation instances on the game object.

Star Twinkle
Various stars with Jitter Scale applied to the prefab to create a simple 'twinkle' effect. If you select a star, notice that the channels are locked so that the stars scale uniformly in each direction.

Camp Fire
This scene demonstrates how Jitter can be used to simulate natural noisy effects like fire. Both Jitter Light Intensity and Jitter Light Radius components are controlling the lights properties. Both components use the same frequency and, importantly, the same *seed*, which means the noise is the same. Finally, Jitter Position is also applied to move the lights position around.

Jitter Blend Shapes
A wiggling worm that uses Jitter Blend Shape to randomly blend from one BlendShape to another, combined with a bit of Jitter Rotation.

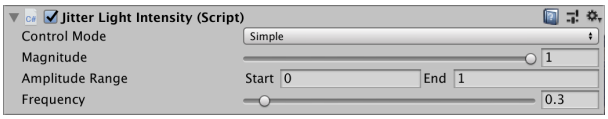
Scripted Jitter
An example of how to instantiate Jitter on an object at runtime through a script. Open the associated JitterInstantiator.cs file, attached to the Cube, to see how this is accomplished. More details about the Jitter code interface can be found in the Jitter API section of this document.

Custom Jitter
An example of how to user the Custom Jitter component. Custom Jitter does not control any predefined property on an object. Instead it is possible to access the value from the component using GetComponent<JitterCustom>().jitterValue; to drive your own behaviour in a script. This example shows how to do this.

Controlling Jitter

Once you've looked at the examples, you're ready to author your own Jitter. The following section describes what the various Jitter controls do.

Common controls
All Jitter components share a set of common controls. These will be below any component-specific controls. Light Intensity and Light Radius have no additional component-specific controls of their own.



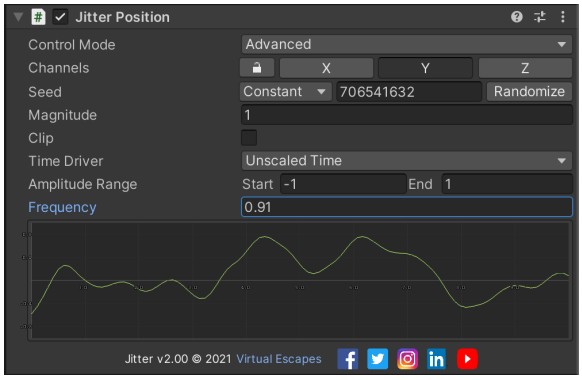
Control Mode
This allows you to switch between Simple and Advanced layout modes. In Advanced Mode, you have access to the seed (see Seed below), clipping, and the magnitude and frequency values can be entered via numerical input with no maximum limits. In simple mode, the seed control is not shown and the magnitude and frequency can be controlled with sliders, limited to a range of useful values. If a value has been entered in advanced mode that is outside the range of the slider in simple mode, then that control will continue to be displayed as a numerical input field, so don't worry about losing your values by switching modes.

Magnitude
Larger numbers increase the noise output, smaller numbers decrease it. Magnitude provides a quick and easy way to adjust the amount of noise. Negative values are not valid.

Amplitude Range
Similar but not quite the same as magnitude. The amplitude range lets you control exactly what values the jitter will vary between. If the magnitude is set to 1.0, the noise values will exactly fall within these ranges. If you set the magnitude to 2.0, the noise values will be double those specified by the amplitude range.

Frequency
This controls the speed of the Jitter, in seconds. Larger numbers make the noise slow, like a boat swaying on the ocean, while smaller numbers (values less than around 0.5) make the Jitter become more like a shake, as if your spaceship has just been hit by a photon blast. If you want different Jitter components to share the same noise, make sure they share both the seed and the frequency values. Negative values are not valid.

Channels
Applicable to Jitter Position, Jitter Rotation and Jitter Scale:
The channels (or axes) that Jitter can be applied to can be toggled on or off. eg if you want your Gameobject's position to vary in all three channels, x, y and z, then toggle all the channel buttons to *on* (dark grey).
To lock the channels together so they share the same noise, toggle the *lock* to on. This is particularly useful for the Jitter Scale component, as it allows you to add the same amount of noise to each scale channel of an object, making it appear as a solid object. If you apply different noise values to each channel independently (lock - off) then it will wobble like jelly!



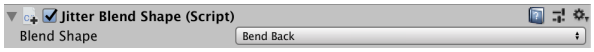
Advanced Control Mode
Advanced Control Mode gives you greater control of the properties of Jitter. If you don't need this then you can keep it in Simple mode.

Seed
Similar to any standard random number generated by a computer, the perlin noise used by Jitter isn't really random, but rather pseudo-random. This is is actually very useful because it means that the noise pattern can be the same every time.
The first control lets you switch between Constant and Variable seeds. Constant will use the same seed as shown in the inspector each time. Variable will choose a random different seed each time the Jitter component becomes active in your scene.
The numerical input field lets you change the seed number if you want a different noise pattern. Finally, hit the *Randomize* button to let Jitter pick a new (pseudo)random number for the seed. If you want various Jitter components to share the same pattern of noise, then copy and paste the same seed values between them.

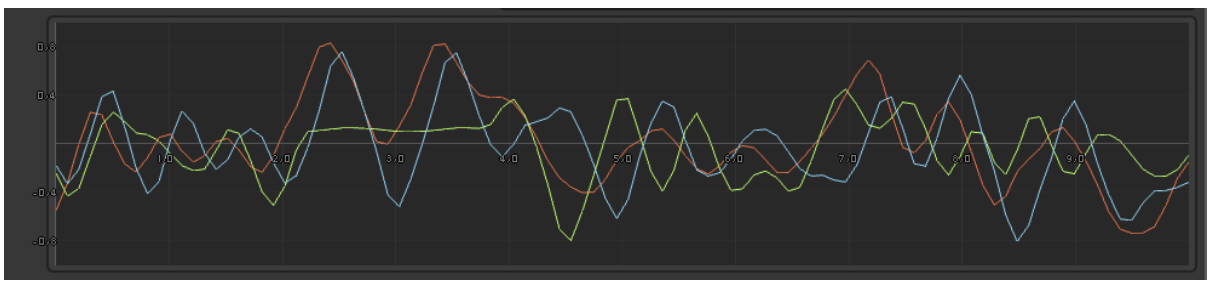
Clip
Setting this to ON will clip the noise when the values go outside of the Amplitude Range. Useful if, for example, you need to constrain movement to the insides of a container.

Time Driver
Normally Jitter is tied to the timing of your game using Time.time. This means that if you employ time scaling to create slow-motion effects, for example, Jitter will also slow down and will match the timing of the rest of the action. However, if you do not want this you can switch to Unscaled Time so that Jitter will continue playing normally regardless of the timescale.

Jitter Blend
This component has a drop down that allows you to specify which BlendShape to apply the Jitter to.



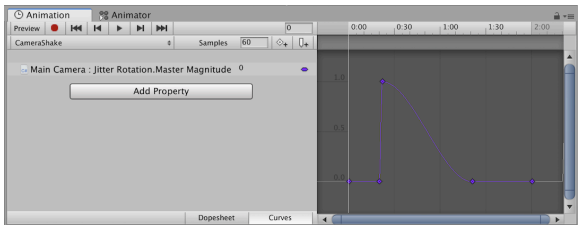
Realtime Preview
If your seed is set to C*onstant*, when setting values such as Frequency, Magnitude etc, this realtime preview shows you exactly what noise you are going to get. If the seed is Random, then this won't be an exact representation but is still useful in determining the overall 'shape' of the noise.



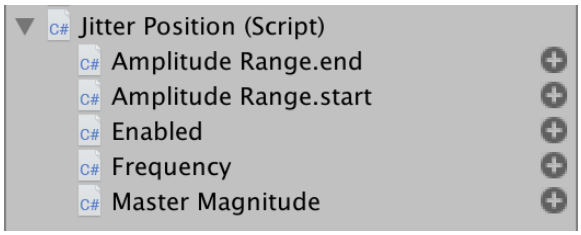
Animating Jitter

Jitter is, by its nature, an animated effect but it's also possible to animate (add keyframes to) the various properties of Jitter using Unity's animation system.

Example of controlling multiple Jitter Rotation components by keyframing the Master Magnitude property



The various animatable properties of a Jitter Component.



Master Magnitude
When a Gameobject contains just one instance of a Jitter Component, it is possible to add its properties and animate without limitations. However, if you have multiple instances of the same component, eg three Jitter Rotation components, then Unity's animation system limits you to only being able to add properties and animate one of them. To make matters worse, it's not necessarily clear which one of the components will be animated.
If you need to animate the frequency or amplitude ranges of various (eg) Jitter Rotations on the same GameObject, then you'll need to nest it under empty GameObjects and add one instance of the desired Jitter component to each.

However, if you want to simply animate the magnitude (and this is the most common/useful animatable property) then Jitter provides a **Master Magnitude** property to make this simple. Master Magnitude controls the magnitude of all the Jitter components of that type on one game object, removing the need to split your Jitter over multiple nested objects. An example of the usage can be found in the Camera Shake example.

General Usage Notes

Jitter applies its values in the LateUpdate cycle of a Gameobject. In order for Jitter to work reliably, it is recommended that you apply any changes to your objects properties that will also be affected by a Jitter component to the Update or FixedUpdate methods and avoid altering them in LateUpdate.
If you do need to change a value which is affected by Jitter in LateUpdate, then it's possible but only if you adjust the script execution order, ensuring that your routines occur before those of Jitter.

Jitter API

Jitter provides public methods and properties in order to be able to provide a high level of control through scripting.
The public classes which can be instantiated using AddComponent<> are:
JitterPosition
JitterRotation
JitterScale
JitterLightRadius
JitterLightIntensity
JitterBlendShape
JitterCustom

Public Methods
void Initialise() - Each Initialise() method is tailored to the specific kind of Jitter you are initialising.

Common Properties
Range amplitudeRange
float frequency
bool isValid (readonly)
float magnitude
float masterMagnitude
int seed

JitterPosition, JitterRotation, JitterScale have the following class-specific properties:
bool axisLocked
AxisMasks axisMask

JitterCustom Property
float jitterValue

If you have any bugs, or suggestions for improvement please don't hesitate to get in touch:
info@virtualescapes.no

We hope you find Jitter useful and look forward to seeing all the creative ways you use it.