EZ Effects v1.0.1

Created By Kevin Somers

Contents

EZ Effects Overview

The use of particle systems and lighting to create effects is an essential part of making a game fun to play. This is especially true for shooters, where effects such as muzzle flashes, impacts, and tracers add a substantial amount of polish to the game, and usually make it feel a lot more satisfying.

This package contains scripts that handle these muzzle flash, impact, and tracer effects, as well as a couple other scripts that are useful for particles and lights. It is very simple to set up and very easy to use.

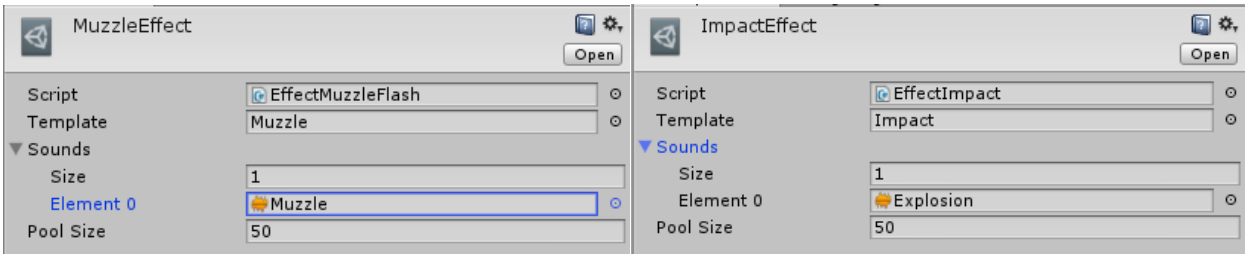
IMPORTANT: This package relies on the EZ Object Pools system to function. If you do not have this package it can be downloaded here: https://www.assetstore.unity3d.com/#!/content/28002
THIS PACKAGE WILL NOT WORK WITHOUT IT.

Setting Up The Effects

To create a new effect, you must make a new asset that holds all the data for that effect. To do this, go to Assets > Create and choose from **New Tracer Effect**, **New Muzzle Effect**, or **New Impact Effect**. All of these effects have similar properties, with a few minor differences.

It is important to say that objects being used as **Templates** for these effects should have a **PooledObject** script (or derivative script such as **TimedDisable**) attached to them, so that they will automatically be put back into the object pool for later use.

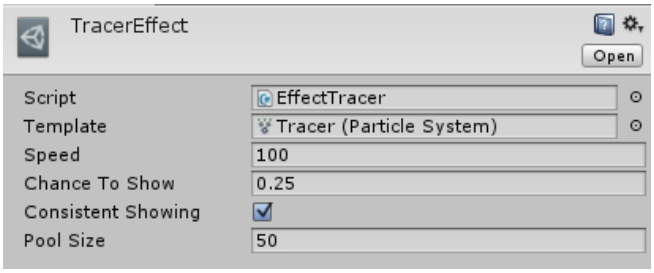
Muzzle Flashes and Impacts



Muzzle flashes and Impacts have 3 properties:

- **Template:** The object this effect will use and make copies of. For Impacts, the template object should have an AudioSource attached if you want it to play sounds.
- **Sounds:** Sounds in this list will play whenever the effect is displayed. Sounds are randomly chosen from the list.
- **Pool Size:** The size of the object pool that contains the objects for this effect. The more often this effect should be displayed, the larger the pool size should be.

Tracers



Tracers have slightly different properties associated with them:

- **Template:** This is the Particle System the effect will use as a template. Unlike the Muzzle Flash and Impact effects, this object ***must*** be a particle system.
- **Speed:** How fast the tracer should travel.
- **Chance To Show:** Real life weapons usually do not fire tracer rounds every shot. This value allows you to define the chance that a tracer effect will be shown. So for example in the picture above, a tracer has a 25% chance of being shown.
- **Consistent Showing:** If this is checked, tracer effects will be shown consistently based on the **Chance To Show** value. So in the picture above, a tracer effect will be displayed every 4th round (0.25 = 1 / 4).
- **Pool Size:** The size of the object pool that contains the objects for this effect. The more often this effect should be displayed, the larger the pool size should be.

Using The Effects In Code

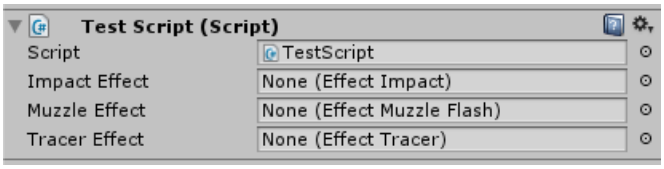
Using any of the effects is very simple, and generally only involves a single line of code. Each effect has a different set of parameters, which I will explain here.

***Note: Much of the info required by these effects can be found in the RaycastHit data structure provided in Unity.**

Setup

In order to gain access to an effect, you must have a field in your scripts that corresponds to that effect. So if your script needs all 3 effects, you should have these fields:

```
public EffectImpact ImpactEffect;
public EffectMuzzleFlash MuzzleEffect;
public EffectTracer TracerEffect;
```



Simply drag and drop the assets explained in the previous section into these fields and you are ready to use them!

Muzzle Flashes

In order to display a muzzle flash, you need to call the **ShowMuzzleEffect** method:

```
MuzzleEffect.ShowMuzzleEffect(Cannon.transform, true, Audio);
```

The parameters for this method are:

- **Origin:** The transform this muzzle flash will be spawned at. It uses both the position of the transform and the direction it is facing.
- **ParentToTransform:** Whether or not the effect should be parented to the given transform for the duration that it is active. Useful for longer effects.
- **AudioSource:** An optional AudioSource that will play the sounds defined in your **EffectMuzzleFlash** asset.

An alternative method is one that supplies separate position and direction vectors rather than a Transform:

```
MuzzleEffect.ShowMuzzleEffect(Cannon.transform.position, Cannon.transform.forward,
Audio, Cannon.transform);
```

The parameters for this method are:

- **Origin:** The position this muzzle flash will be spawned at.
- **Direction:** The direction this muzzle flash should face when spawned.
- **AudioSource:** An optional AudioSource that will play the sounds defined in your **EffectMuzzleFlash** asset.
- **ParentTo:** An optional transform that the effect will be parented to when spawned.

Impacts

Similarly to the muzzle effects, impact effects are displayed by calling the **ShowImpactEffect** method:

```
ImpactEffect.ShowImpactEffect(hitPos, hitNormal);
```

The parameters for this method are:

- **Position:** The position this effect should be displayed at.
- **Normal:** An optional parameter that defines the normal of the surface hit. Useful for effects like mortar impacts that have a non-spherical shape.

***IMPORTANT: When creating your impact effects that use the Normal of a surface, remember that the Z-axis will be facing away from the surface, so you should treat the Z-axis as the 'up' direction for the particle systems.**

Tracers

Tracer effects are displayed by calling the **ShowTracerEffect** method:

```
TracerEffect.ShowTracerEffect(Cannon.transform.position, Cannon.transform.forward, distance);
```

The parameters for this method are:

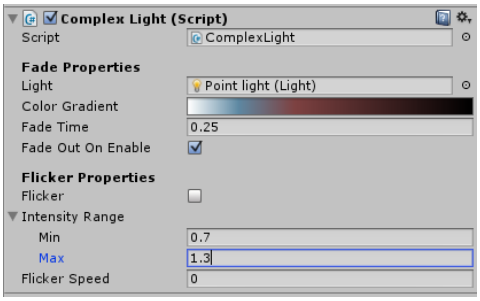
- **Origin:** The position the tracer should be spawned at. This is usually the end of the barrel of a gun.
- **Direction:** The direction the tracer should travel.
- **Distance:** How far the tracer should travel. This is usually the distance to the point of impact. This is used to accurately set the lifetime of the tracer particle. You can find this info in the **RaycastHit** data structure provided in Unity.
-

Complex Light and Complex Particles

In addition to the 3 effects explained above, this package contains 2 scripts that have useful functions for lights and particle systems.

Complex Light

The **ComplexLight** script has options to fade out a light over a colored gradient as well as do a flickering effect.

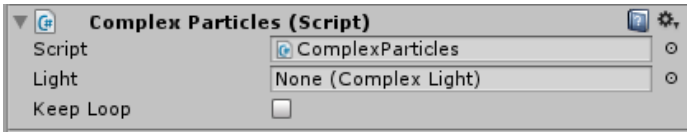


- **Light:** The light this script will effect. Will be automatically assigned if left null.
- **Color Gradient:** The gradient the light will use to change its color. When fading out, the light's color will move from left to right, and right to left for fading in.
- **Fade Time:** The time in seconds the light will take to fade out / in.
- **Fade Out On Enable:** Should the light fade out as soon as it is enabled? Useful for explosion effects that are spawned using an **EffectImpact**.
- **Flicker:** Should this light flicker?
- **Intensity Range:** The range of intensity the light should randomly flicker between.
- **Flicker Speed:** The speed of the flicker. Lower values make for a slow pulse effect while higher values make for a strobe effect.

In order to fade the light, call the **FadeIn** or **FadeOut** methods.

Complex Particles

The **ComplexParticles** script groups many particle systems into one place, so they can be easily enabled and disabled. To use this script, simply set all of the particle systems you want to be grouped together as children of a single parent object, and attach this script to that parent object.



- **Light:** An optional **ComplexLight** that will fade in and out with the particles as they are enabled and disabled. Useful for fire particles and the like.
- **Keep Loop:** Whether or not the particle systems should keep looping even when they are disabled. Useful for looping effects like fires and fountains that should be able to start emitting again after they are disabled. This **SHOULD NOT** be checked for effects like explosions or other one-shot effects.

To enable or disable emissions, call the **EnableEmissions(bool enable)** method. Passing True will enable the particles and False will disable them.

I have also included a **ComplexParticlesDisable** script to be used with the EZ Object Pools. This script will automatically disable a **ComplexParticles** object once all of the children particle systems are done emitting. Make sure that **Keep Loop** is ***not*** checked for effects that use this script.

Scripting

EffectMuzzleFlash

ShowMuzzleEffect (Overload 1)
Displays the muzzle effect at the given transform and plays a sound if applicable.

Parameters:

- **Origin (Transform):** The transform this effect will be spawned at. Uses both the position and rotation data of the transform.
- **ParentToTransform (bool):** Should the effect be parented to the given transform when spawned?
- **Audio Source (AudioSource):** An optional audio source that sounds will be played from.

ShowMuzzleEffect (Overload 2)
Displays the muzzle effect at the given position and direction and plays a sound if applicable.

Parameters:

- **Origin (Vector3):** The position this effect will be spawned at.
- **Direction (Vector3):** The direction this effect will face when spawned.
- **ParentTo (Transform):** An optional transform the effect will be parented to when spawned.
- **Audio Source (AudioSource):** An optional audio source that sounds will be played from.

SetupPool
Sets up the object pool for this effect. This is automatically done on the first call to **ShowMuzzleEffect**, so calling this method manually is usually not necessary.

EffectImpact

ShowImpactEffect (Overload 1)
Displays the impact effect at the given position and rotates it to match the given normal. Useful for impacts that have a non-spherical shape.

Parameters:

- **Position (Vector3):** The position this effect will be spawned at.
- **Normal (Vector3):** The normal vector the effect will be rotated to match. This info is usually located in the **RaycastHit** data structure provided in Unity (for Raycasts only).

ShowMuzzleEffect (Overload 2)
Displays the impact effect at the given position.

Parameters:

- **Position (Vector3):** The position this effect will be spawned at.

SetupPool
Sets up the object pool for this effect. This is automatically done on the first call to **ShowImpactEffect**.

EffectTracer

ShowTracerEffect
Displays the tracer effect starting at the given origin position and moving in the given direction.

Parameters:
- **Origin (Vector3):** The position this effect will be spawned at.
- **Direction (Vector3):** The direction the tracer will travel.
- **Distance (Float):** How far the tracer should travel. This is usually the distance to the point of impact. Used to accurately calculate the lifetime of the effect. This info is usually located in the **RaycastHit** data structure provided in Unity (for Raycasts only).

SetupPool
Sets up the object pool for this effect. This is automatically done on the first call to **ShowTracerEffect**, so calling this method manually is usually not necessary.

ComplexLight

FadeOut
Begins fading out the light. If not given a time, it uses **FadeTime** as a default.

Parameters:
- **Time (Float):** An optional parameter defining how long to fade the light (in seconds).

FadeIn
Begins fading in the light. If not given a time, it uses **FadeTime** as a default.

Parameters:
- **Time (Float):** An optional parameter defining how long to fade the light (in seconds).

ComplexParticles

EnableEmissions
Enables or disables the emission of all children particles.

Parameters:
- **Enable (Boolean):** True enables the particles, False disables them.

IsAlive
Checks if any of the child particle systems are still emitting or have particles that are still active.

Returns:
- **True** if 1 or more children particles are still active, **False** if all children particles are inactive.