Sci-Fi Effects

Version 1.0

http://www.forge3d.com
support@forge3d.com

Contents

Introduction

Thank you for purchasing Sci-Fi Effects!

This guide describes the features of the Sci-Fi Effects integration in Unity3D. A basic understanding of the Unity3D engine, as well as C# programming language is assumed. Having basic knowledge of the shader and visual effects design in Unity3D may be advantageous.

For more information please visit www.forge3d.com

If you have any questions, suggestions, comments or feature request please do not hesitate to contact us at support@forge3d.com

Brief overview

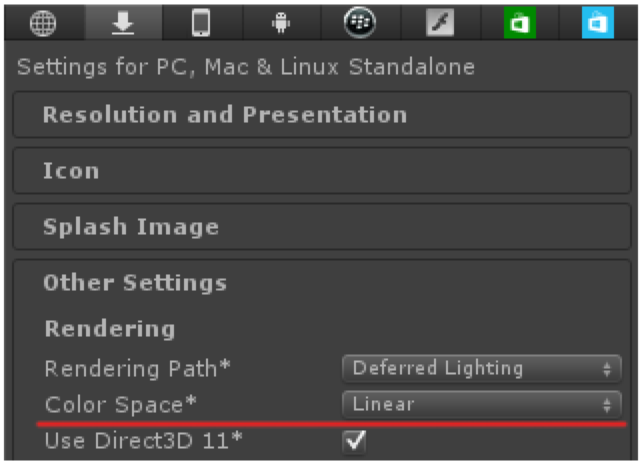
Sci-Fi Effects is a collection of more than 80 ready to use prefabs including complex particle systems, billboards, sounds and helper scripts. Example scenes provided within the package utilize some of the prefabs and demonstrate usage of these effects while all scripts are self explanatory and well documented.

For ease of use this package includes particle scaling script to help you match prefabs scale to your scene scale. See corresponding section of this manual describing particle scaling procedure in better details.

Please note this package is best used with linear color space.

Getting started

First of all make sure you set your project settings to linear color space. If you haven't done so go to **Edit -> Project Settings -> Player** and select Linear using drop down list:


Settings for PC, Mac & Linux Standalone

Resolution and Presentation

Icon

Splash Image

Other Settings

Rendering

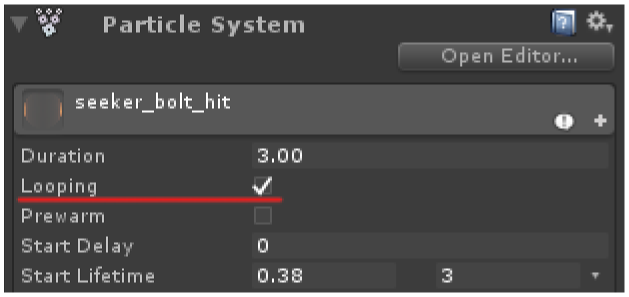
Rendering Path* Deferred Lighting

Color Space* Linear

Use Direct3D 11* ✓

** If you decide to stay in gamma space it may require tweaking color values depending on your personal preferences.*

Most of the particle effects provided are set to be looped over time for demo purposes. Before using the actual prefabs make sure to uncheck **Looping** flag for the current parent and all of the child particle systems if appropriate:


Particle System

seeker_bolt_hit

Duration 3.00

Looping ✓

Prewarm

Start Delay 0

Start Lifetime 0.38 3

Example scenes

There are two example scenes included at **Assets/FORGE3D/Sci-Fi Effects/Examples** path:

turret_scene - Demonstrates various assembled weapon effects with help of pool manager and several weapon controllers. Use your mouse cursor to aim and left mouse button to fire. Left and right cursor keys used to switch weapon types.

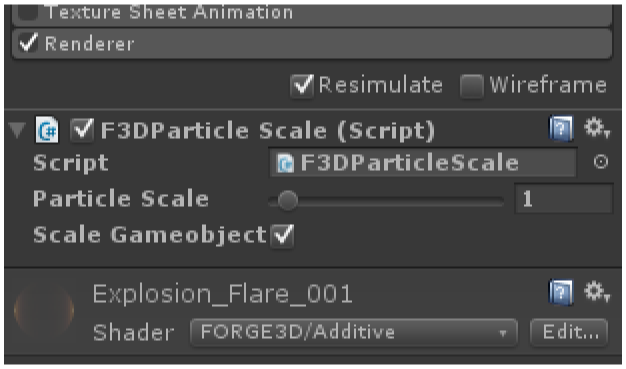
preview_scene - Shows all available effects included in this package.

** Please note that example scripts provided may require additional modifications before they can be used in your project or may not be suited for usage in different environments at all.*

Scaling particle effects

Particle scaling is made possible with use of **F3DParticleScale** script located at **Assets/FORGE3D/Sci-Fi Effects/Code** path.

To scale a particle system attach this script to a parent object and use the slider to scale the parent and all included child objects at the same time:



F3DParticle Scale (Script)
Script: F3DParticleScale
Particle Scale: 1
Scale Gameobject: ✓

Script reference

This section will give you brief details on scripts used by the turret example. It also important to know that most scripts rely on F3DTimer class and require an instance of such to be present in the scene. Please take your time to examine each script more carefully to fully understand what is happening behind the scene.

F3DAudioController

This script is an example of audio management and playback. What it does is playing an audio clip at specific position and modifies various audio settings such as random volume or pitch depending on the method called.

F3DBeam

This script is mainly used for updating beam weapons such as beam laser with uv animation for tiled textures, real time raycasting and interacting with rigid bodies by applying AddForceAtPosition. It is also scales the texture along its length depending on the beam length so it’s never gets stretched.

F3DDespawn

This script is used to despawn most of the effects after predefined delay by calling corresponding method of the pool manager that is included in this package.

F3DFlameThrower

This script is used by flame thrower prefab to manage some utility tasks such as fading in/out the lights and despawning the effect.

F3DFXController

This script defines all the weapon types and the way the are spawned such as managing prefab references, rate of fire, invoking specific audio routines and finally the GUI drawing seen in turret example.

F3DLightning

This script is mainly used for updating lightning gun weapon such as updating amount of lightning points and animating the uvs. It is also scales the texture along its length depending on the beam length so it’s never gets stretched.

F3DPool

This script is a pool manager which is used to pre instantiate all the provided prefabs before the scene starts playing. All weapon scripts use **OnSpawned** and **OnDespawned** methods which also makes them compatible with other pool managers found on the asset store.

F3DProjectile

This script is a projectile controller. It is using ray casting to detect colliders in advance and in case of an impact calls corresponding method to play sound effects and spawn impact prefabs.

F3DPulsewave

This script is used to control pulse wave scaling and fading over time.

F3DRandomize

This script is used to randomize transform’s scale and rotation for currently spawned object. Mainly used with muzzle flashes and projectiles.

F3DShotgun

This script is used to manage shotgun particle system and react to particle collision events sent by spawning impact prefabs and playing audio clips at impact points.

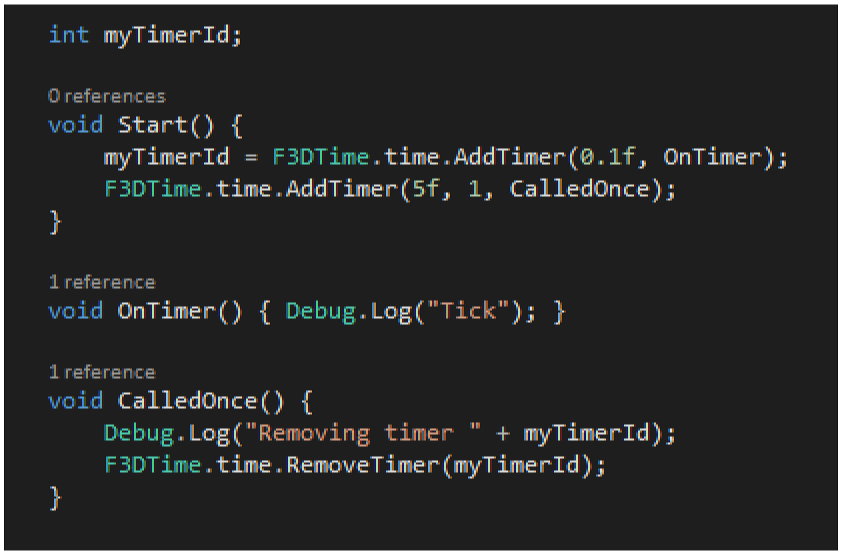
F3DTime

F3DTime class is a singleton instance used to create and manage the timers. To start using this component simply attach it to any gameobject in scene use one of the following overloads to create a timer:

int F3DTime.time.AddTimer(float rate, System.Action callBack);
int F3DTime.time.AddTimer(float rate, int ticks, System.Action callBack);

AddTimer method has two overloads. The first one can be used to invoke a specified method at specified rate until stopped while the second one requires you to specify number of ticks before it stops. The return value of AddTimer is a unique int handle which should be used with RemoveTimer method to stop it's execution.

Let's look at the code example below where two timers are created. Note that we store the id for the first timer in myTimerId variable so we could dispose it later.



```
int myTimerId;

0 references
void Start() {
    myTimerId = F3DTime.time.AddTimer(0.1f, OnTimer);
    F3DTime.time.AddTimer(5f, 1, CalledOnce);
}

1 reference
void OnTimer() { Debug.Log("Tick"); }

1 reference
void CalledOnce() {
    Debug.Log("Removing timer " + myTimerId);
    F3DTime.time.RemoveTimer(myTimerId);
}
```

Once initialized the first timer will begin invoking OnTimer method each 0.1 seconds until stopped by second timer after 5 seconds elapses. Then both timers are disposed since we explicitly tell first timer to be removed and the second self removes since it's life scope is only a single tick.

F3DTurret

This script is used to control turret's base and barrel rotation in a specified range as well as checking user input and invoking weapon firing methods on F3DFXController.