Lerp Factory Manual

Thank you for purchasing the Lerp Factory asset. This document will guide you through how to use the lerp factory script.

Welcome V1.10

This version has improved syntax compared with V1.00. However the improvements necessitated a restructuring of the code so this section is a guide to upgrading from V1.00 to V1.10.

Change any LerpManager variables to LerpFactoryScript variables.
Unfortunately it is not really possible to run the new LerpValue and LerpReference scripts outside of the LerpFactoryScript, so any lerping will have to go through the LerpFactoryScript that you will add into the game. Then use the syntax's shown in Appendix 4 Lerp functions syntaxes to change over the code.

In the future there will be more restructuring of the code so as to make it even easier to use and have more functionality.

Setup Guide

Create an empty Gameobject in your scene. Go to the folder where the lerp factory has installed. Locate the LerpFactoryScript script and place it on the empty Gameobject. Then in any script where you wish to start the lerp in, add at the very top of the script where the namespace declarations are using LerpFactory. Then in any script that you will wish to start a lerp from, create a public LerpFactoryScript variable and assign it to the LerpFactoryScript that you have added into the scene. Normally I call this variable LFS.

Then you start a lerp using the syntax's shown in Appendix 4 Lerp functions syntaxes.

What the scripts do

LerpFactoryScript: this is the interface script that you send requests for lerps to.

LerpValue: this is one of the two lerp scripts. It handles lerping of bools, ints, floats, doubles, Vector2s, Vector3s, Vector4s, Quaternions, and Colours. Instances of this class are automatically spawned as needed by the LerpFactoryScript.

LerpReference: this is the other lerp script. It handles lerping transforms.

Roadmap

The next update will feature another major code restructuring, however when that happens I plan to move to having just one script and have it so that you do not need to have the new LerpFactoryScript in the scene at all. You will just call a static function as and when you need a lerp. After this big step, the updates will focus on bug fixes and adding functionality.

Appendix 1 Modes of Lerping.

LerpMode is an enumeration that has three values: Normal, Rewind and Flicker. Below explains what they do.

LerpMode.Normal Lerps from A to B by T;

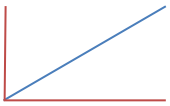
LerpMode.Rewind lerps from A to B by T waits for a specified length of time and then lerps from B to A by T;

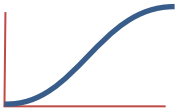
LerpMode.Flicker lerps from A to B by T and adds a random noise that is between a max value and a minimum value.

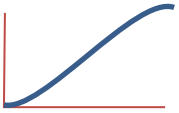
Note that LerpMode.Flicker cannot be used when lerping Quaternions and Transforms.

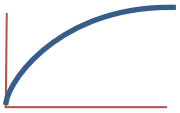
Appendix 2 Types of Lerping.

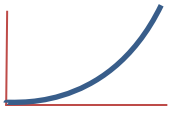
TypeOfLerp is an enumeration that can be used in the both the LerpValue script and the LerpReference script. It has seven different values Lerp, SmoothStep, SmootherStep, sinLerp, cosLerp, Expo, InverseExpo. Below explains what they do.

TypeOfLerp.Lerp your normal lerp, its graphic representation looks like this 

TypeOfLerp.SmoothStep uses the SmoothStep method, it looks like this 

TypeOfLerp.SmootherStep is slightly smoother that the SmoothStep method, it looks like this 

TypeOfLerp.sinLerp eases out of the start variable, it looks like this 

TypeOfLerp.cosLerp eases into the start variable, it looks like this 

TypeOfLerp.Expo is **similar** to CosLerp but not **quite** the same.

TypeOfLerp.InverseExpo is **similar** to SinLerp but not **quite** the same.

Appendix 3 useSpeed, what it does.

useSpeed is a Boolean variable that you can input in to both the LerpValue script and the LerpReference script. When it is set to false, the time parameter inputted is treated as the time to take lerping from point A to point B. If it is set to true, then the time value is treated as the speed to travel at. For example if the sv = 0, ev = 10, time = 2, useSpeed = false then the lerp script will take 2 seconds to lerp from sv to ev. If useSpeed = true then the lerp will behave as if it is traveling at 2 units per second and so will take 5 seconds to lerp from sv to ev.

Appendix 4 Lerp function syntaxes

Below is a complete list of all the functions that you can call from the LerpFactoryScript.

sv, ev, f1, f2 and V must all be of the same variable type. They are respectively the start value(sv), the end value(ev), the max flicker value(f1), the min flicker value(f2), and the actual variable that you wish to lerp(V).

t and reT must also be of type float. They are respectively the time to take lerping/the lerp speed(t) and the time till the lerp rewinds its self(reT).

use is the useSpeed variable(see Appendix 3) and is use to determine if the t input is the time to take or the speed to lerp at.

Mode and Type are the LerpMode and TypeOfLerp respectively.

LFS is the instance of the LerpFactoryScript that you have set in the script.

This is the most basic form of the lerp call:

```
LFS.Int((x)=> V =x, sv, ev, t);
LFS.Float((x)=> V =x, sv, ev, t);
LFS.Double((x)=> V =x, sv, ev, t);
LFS.Vector2((x)=> V =x, sv, ev, t);
LFS.Vector3((x)=> V =x, sv, ev, t);
LFS.Vector4((x)=> V =x, sv, ev, t);
LFS.Colour((x)=> V =x, sv, ev, t);
LFS. Quaternion ((x)=> V =x, sv, ev, t);
LFS.Bool((x)=> V =x, sv, ev, t);
```

This allows flickering. Note, Quaternions cannot have a flicker noise added to them so there is no code for Quaternions here:

```
LFS.Int((x)=> V =x, sv, ev, t, f1, f2);
LFS.Float((x)=> V =x, sv, ev, t, f1, f2);
LFS.Double((x)=> V =x, sv, ev, t, f1, f2);
LFS.Vector2((x)=> V =x, sv, ev, t, f1, f2);
LFS.Vector3((x)=> V =x, sv, ev, t, f1, f2);
LFS.Vector4((x)=> V =x, sv, ev, t, f1, f2);
LFS.Colour((x)=> V =x, sv, ev, t, f1, f2);
```

This is the basic form of the lerp call with useSpeed:

```
LFS.Int((x)=> V =x, sv, ev, t, use);
LFS.Float((x)=> V =x, sv, ev, t, use);
LFS.Double((x)=> V =x, sv, ev, t, use);
LFS.Vector2((x)=> V =x, sv, ev, t, use);
LFS.Vector3((x)=> V =x, sv, ev, t, use);
LFS.Vector4((x)=> V =x, sv, ev, t, use);
LFS.Colour((x)=> V =x, sv, ev, t, use);
LFS. Quaternion ((x)=> V =x, sv, ev, t, use);
```

This is basic form of the lerp call with a rewind time. Note, when you start the lerp using this syntax, the LerpMode is automatically set the LerpMode.Rewind:

```
LFS.Int((x)=> V =x, sv, ev, t, reT);
LFS.Float((x)=> V =x, sv, ev, t, reT);
LFS.Double((x)=> V =x, sv, ev, t, reT);
LFS.Vector2((x)=> V =x, sv, ev, t, reT);
LFS.Vector3((x)=> V =x, sv, ev, t, reT);
LFS.Vector4((x)=> V =x, sv, ev, t, reT);
LFS.Colour((x)=> V =x, sv, ev, t, reT);
LFS. Quaternion ((x)=> V =x, sv, ev, t, reT);
LFS.Bool((x)=> V =x, sv, ev, t, reT);
```

This is the form of the lerp call that takes both a rewind time and useSpeed. Note, when you start the lerp using this syntax, the LerpMode is automatically set to LerpMode.Rewind:

```
LFS.Int((x)=> V =x, sv, ev, t, reT, use);
LFS.Float((x)=> V =x, sv, ev, t, reT, use);
LFS.Double((x)=> V =x, sv, ev, t, reT, use);
LFS.Vector2((x)=> V =x, sv, ev, t, reT, use);
LFS.Vector3((x)=> V =x, sv, ev, t, reT, use);
LFS.Vector4((x)=> V =x, sv, ev, t, reT, use);
LFS.Colour((x)=> V =x, sv, ev, t, reT, use);
LFS. Quaternion ((x)=> V =x, sv, ev, t, reT, use);
```

This is the ultimate form of the lerp.

```
LFS.Int((x)=> V =x, sv, ev, t, f1, f2, reT, use, Mode, Type);
LFS.Float((x)=> V =x, sv, ev, t, f1, f2, reT, use, Mode, Type);
LFS.Double((x)=> V =x, sv, ev, t, f1, f2, reT, use, Mode, Type);
LFS.Vector2((x)=> V =x, sv, ev, t, f1, f2, reT, use, Mode, Type);
LFS.Vector3((x)=> V =x, sv, ev, t, f1, f2, reT, use, Mode, Type);
LFS.Vector4((x)=> V =x, sv, ev, t, f1, f2, reT, use, Mode, Type);
LFS.Colour((x)=> V =x, sv, ev, t, f1, f2, reT, use, Mode, Type);
LFS. Quaternion ((x)=> V =x, sv, ev, t, reT, use, Mode, Type);
```

To lerp transforms the syntax's are:

LFS.LerpRefTransforms(ObjToM,List,t,use,Type);

This will lerp the ObjToM through all of the points in List, and then it restarts again at the beginning.

ObjToM must be a Transform. List must be a Transform[] where the start and end transforms are on the same point.

LFS.LerpRefTransform(ObjToM,EndPoint,t,use,Type,Repeat,EndAtStart);

Here EndPoint must be a Transform variable and is where ObjToM will end up.

EndAtStart determines if ObjToM will end up at where it started. If it is set to true then it will.

Repeat states whether ObjToM will just go back and forth between its start position and the EndPoint position indefinitely(true) or just arrive and stop. If Repeat is set to true then it is recommended that EndAtStart is set to true as well.