Low Pass Filter

Vatio

This is the instruction for the Low Pass Filter package. It's an easy to use, wonderful tool that can smooth supplied data so that the core value remains, but all the noise is filtered out. The main application is to correct any input instabilities and random movements. With this script everything feels smooth and slick, making for a great user interaction. Still, the filter can be applied to any kind of data - statistics, jumpy physics, etc.

Short instruction for quick action

C#

C# usage

```
// Initialize the filter with:
LowPassFilter<[variable type]> lowPassFilter = new LowPassFilter<[
    variable type]>([alpha], [initial value]);

// Use it like:
[new value] = lowPassFilter.Append([value]);

// Get current value like:
[current value] = lowPassFilter.Get();
```

C# example

```
LowPassFilter<float> lowPassFilter = new LowPassFilter<float>(0.05f,
    0.2f);
lowPassFilter.Append(0.07f);
lowPassFilter.Append(0.35f);
lowPassFilter.Append(0.15f);
lowPassFilter.Get();    // this returns 0.199f
lowPassFilter.SetA(0.03f);
```

JavaScript

JavaScript usage

```
// Initialize the filter with:
var lowPassFilter = new LowPassFilter.<[variable type]>([alpha], [
    initial value]);

// Use it like:
[new value] = lowPassFilter.Append([value]);

// Get current value like:
[current value] = lowPassFilter.Get();
```

JavaScript example

```
var filter = new LowPassFilter.<float>(0.05f, 0.0f);
filter.Append(0.07f);
filter.Append(0.35f);
filter.Append(0.15f);
filter.Get();   // this returns 0.199f
filter.SetA(0.03f);
```

Boo

Boo usage

```
// Initialize the filter with:
lowPassFilter = LowPassFilter[of [variable type]]([alpha], [initial
    value])

// Use it like:
[new value] = lowPassFilter.Append([value])

// Get current value like:
[current value] = lowPassFilter.Get()
```

Boo example

```
filter = LowPassFilter[of float](0.05f, 0.0f)
filter.Append(0.07f)
filter.Append(0.35f)
filter.Append(0.15f)
filter.Get()    // this returns 0.199f
filter.SetA(0.03f)
```

Instruction

This is a generic low pass filter. It's an easy to use, wonderful tool that can smooth supplied data so that the core value remains, but all the noise is filtered out. The main application is to correct any input instabilities and random movements. With this script everything feels smooth and slick, making for a great user interaction. Still, the filter can be applied to any kind of data - statistics, jumpy physics, etc.

Example usages

- filtering all kinds of input, such as joysticks, accelerometers, gyros
- smoothing ragged data
- adding scripted inertia
- averaging data
- making sure everything looks and works pretty :)

The script works with both Unity free and Pro for all the target platforms. Although it's written in C#, it works from other scripting languages just as well. The package includes the source code with everything well commented and extendible if needed. Also there are three example scenes - one for desktop computers, one for mobile devices and one simple desktop game - each one completely written in C#, JavaScript and Boo, making for a total of 9 examples.

Supported variable types

- integer types
- floating point types
- vector types
- quaternions
- all other types that support arithmetic operations

Update guide

1. Delete the previous Low Pass Filter version before updating. Your project itself will work without any changes.
2. Import new version.

Usage

There are two ways of using the filter:

- normal mode - the user script supplies the data at any convenient time and the filter just compares it to the previous values
- auto-update mode - the filter will periodically pull the data from specific input and the user script will just need to read the filtered value

In the normal mode the user just needs to instantiate a LowPassFilter object, which has 4 basic methods:

- **LowPassFilter**

  This is the constructor for the filter.

  Parameters are:

  - a - smoothing factor, the lower the value, the more inertia filter has, it has to be in range [0, 1], for starters try 0.05f, and usually keep between 0.02f and 0.2f
  - initialValue - initial value for the filter, it should be set as close as possible to expected average value of the filtered variable

- **Append**

  This is the function adding new unfiltered value for the filter. It stores the new computed value, and also returns it.

  Parameters are:

  - input - the new value

  The function returns the new filtered value

- **Get**

  This is the getter for current filtered value

- **SetA**

  This is the setter for the smoothing factor. Use this setter to change smoothing factor during filter operation.

  Parameters are:

– a - smoothing factor, the lower the value, the more inertia filter has, it has to be in range [0, 1], for starters try 0.05f, and usually keep between 0.02f and 0.2f

C# example

```
LowPassFilter<float> lowPassFilter = new LowPassFilter<float>(0.05f,
   0.2f);
lowPassFilter.Append(0.07f);
lowPassFilter.Append(0.35f);
lowPassFilter.Append(0.15f);
lowPassFilter.Get();    // this returns 0.199f
lowPassFilter.SetA(0.03f);
```

JavaScript example

```
var filter = new LowPassFilter.<float>(0.05f, 0.0f);
filter.Append(0.07f);
filter.Append(0.35f);
filter.Append(0.15f);
filter.Get();   // this returns 0.199f
filter.SetA(0.03f);
```

Boo example

```
filter = LowPassFilter[of float](0.05f, 0.0f)
filter.Append(0.07f)
filter.Append(0.35f)
filter.Append(0.15f)
filter.Get()    // this returns 0.199f
filter.SetA(0.03f)
```

Auto-update mode - C# only

In the auto-update mode the user doesn't have to instantiate the filter itself. Instead he or she needs to define a class implementing the **ILowPassFilterInput** interface. The only method that needs to be implemented is:

- **Get**

  This is the getter for current unfiltered value

Then the user needs to add the **LowPassFilterMonoBehaviour** component to an object in the scene, and initialize it. It's methods are:

- **Set**

  This is the function initializing the component.

  Parameters are:

  - input - the object providing input to the filter, needs to be instantiated previous to invoking this method
  - a - smoothing factor, the lower the value, the more inertia filter has, it has to be in range [0, 1], for starters try 0.05f, and usually keep between 0.02f and 0.2f
  - initialValue - initial value for the filter, it should be set as close as possible to expected average value of the filtered variable

- **Get**

  This is the getter for the current filtered value.

Auto update example

```
public class MouseInput : ILowPassFilterInput<Vector3>
{
    public Vector3 Get()
    {
        return Input.mousePosition;
    }
}

LowPassFilterMonoBehaviour lowPassFilterMonoBehaviour = gameObject.
   AddComponent<LowPassFilterMonoBehaviour>();
lowPassFilterMonoBehaviour.Set<Vector3>(new MouseInput(), a, Vector3.
   zero);
    // Wait few frames in between to let the filter compute some values
lowPassFilterMonoBehaviour.Get<Vector3>();  // this returns the current
    filtered mouse position
```

Final note

Important: all filtered types need to support at least three arithmetic operations: addition, subtraction and multiplication.