LITE FPS COUNTER

Version 1.0.0 | OmniSAR Technologies

Introduction

Thank you for trying out Lite FPS Counter. We take performance seriously, so we developed our very own time probing tool. During its creation, the development was driven by some important requirements that sit at the very core of the philosophy that Lite FPS Counter was created after:

- **High Accuracy** - We wanted our tool to show the **real, actual** time budget in our simulation. We've tested many methods of time measurement and integration, but in the end, we've chosen what we think it's the most natural and accurate of them all: Given an amount of **time**, how many **frames** were drawn? As simple as that.
- **High Steadiness** - The resulting time average should be as steady as possible. By dividing the counted frames to the integrated time, we inherit the natural steadiness that comes with addressing a signal by individual segments as opposed to as continuous processing as a IIR or a moving average low-pass filter would require.
- **High Responsiveness** - Given sudden changes in the actual time average, the computed average should adapt quickly. By addressing the probed time samples in segments, we get the **maximum responsiveness** that a time integration system can get. Within the next framerate update (not to be confused with *frame update*), the component will display the new real average regardless of how big the change was.

Some methods measure the time on each frame and apply an IIR low-pass filter or a moving average filter to smooth out the time variations that naturally happen from one frame to another. While this method may look appealing, it comes with a compromise between steadiness and responsiveness. A long integration time will fail to adapt quickly to a new time category. For example, when the framerate changes from an average of 60 to an average of 15, it will take a considerable amount of time for the average framerate to catch up with the new average of 15.

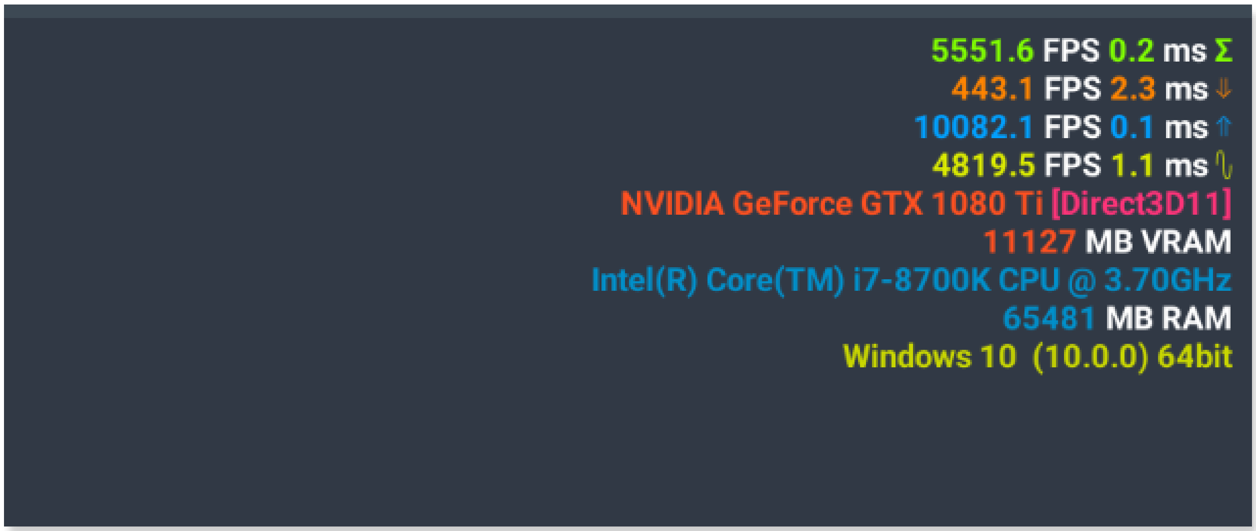
Other methods will average the instant time and the instant framerate independently, which is even worse, since the relation between time and framerate is non-linear and variations will weight differently in the time average than in the framerate average.

- **High Performance / Low Intrusiveness** - What is the point of having a measurement tool that tampers with the very thing it needs to measure? A considerable amount of effort was directed into making Lite FPS Counter as performant and as less intrusive as possible. Various techniques were used to make Lite FPS Counter stand to its name, including deferred updates, static one-time canvas dirtying and string caching where possible. The general strategy used is that only the things that need to be updated are updated.
- **High Reliability and Consistency** - We needed our measuring tool to be trustworthy and to **work as expected every single time we use it**. Take it from one scene to another, from one project to another, from one device to another and it should work great, both from a functional, as well as from a performance standpoint.
- **Diversified Deployment** - The fact that our tool does not contain a native plugin **is intentional**. Having no native plugins means that you can deploy your game and have Lite FPS Counter functional on any device running any operating system that can run Unity builds, essentially **any platform that Unity can deploy to**.
- **Independent** - A tool that is used just to measures your game's performance should not come with heavy dependencies. In fact, it should have **none**, period. Lite FPS Counter is a standalone package that does not even require the Standard Assets package. It is self-sustained and it has everything it needs to function properly in its own package.
- **Lightweight** - A measurement tool should be **as low profile as possible**, both from an execution performance standpoint, as well as from a working standpoint. While working on your game, the last thing you want to deal with is a package that takes ages to compile, slows down the editor while doing basically nothing or having its presence felt by any means. An ideal tool would basically be added to the scene and forgotten. Lite FPS Counter doesn't like to be forgotten, but it's a sacrifice we're willing to make.
- **Easy and Fast Installment** – We work on many projects and having a tool that can be **set up as fast and as easy as possible** is crucial and a total must for our daily work. The setup for Lite FPS Counter is so easy and so fast that we barely had to write documentation for it. Basically, we drop a prefab in the scene and we're done. That's it. Go back to what you were doing. For the more pretentious user, we've also included an "add game object" menu item as an alternative setup method that will be presented in the Scene Setup section of this document.

Performance Evaluation

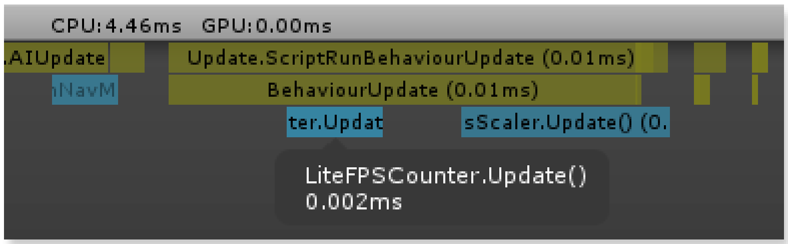
To measure the overhead of our tool, we've set up an absolutely empty scene in which the only things that run are Lite FPS Counter on its own canvas and a simple scene changing UI. We've used an update interval of 1 second for Lite FPS Counter. The **camera** only clears a solid color and **cull nothing**, the **Occlusion Culling** and **MSAA** are disabled. Although it would have been nice to strip down everything and let our tool to run at full speed, for sure there are some things Unity does under the hood that we can't disable. We're going to have to live with that.

Here is a screen capture showing **a staggering average of 5500+ frames** being drawn per second in 2560x1440. The update interval used here is 1 second, meaning that once every second Lite FPS Counter does some math and updates the UI.



```
5551.6 FPS 0.2 ms Σ
443.1 FPS 2.3 ms ⇓
10082.1 FPS 0.1 ms ⇑
4819.5 FPS 1.1 ms ∿
NVIDIA GeForce GTX 1080 Ti [Direct3D11]
11127 MB VRAM
Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
65481 MB RAM
Windows 10  (10.0.0) 64bit
```

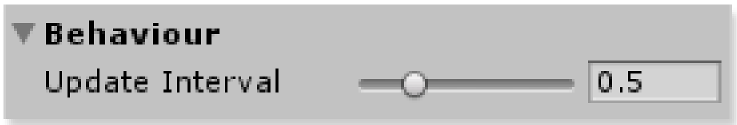
But how much does Lite FPS Counter really consume every frame?



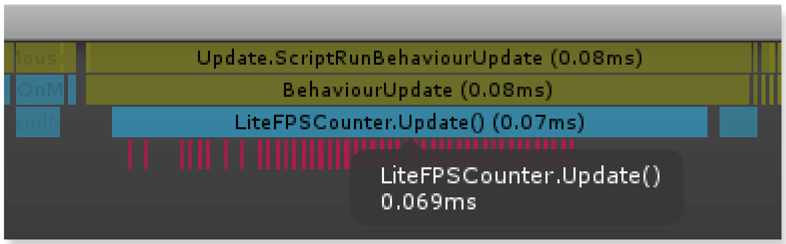
```
CPU:4.46ms   GPU:0.00ms
.AIUpdate   Update.ScriptRunBehaviourUpdate (0.01ms)
nNavM       BehaviourUpdate (0.01ms)
ter.Updat   sScaler.Update() (0.
LiteFPSCounter.Update()
0.002ms
```

In the profiler, we can see that on a regular basis, the Update function takes ~2 microseconds (0.002ms). If you're unsure what that means, it's **half a million frames per second**, yes - half a million. Arguably the per-frame performance hit is more than neglectable, but we'll take it into account anyways for the sake of being fair.

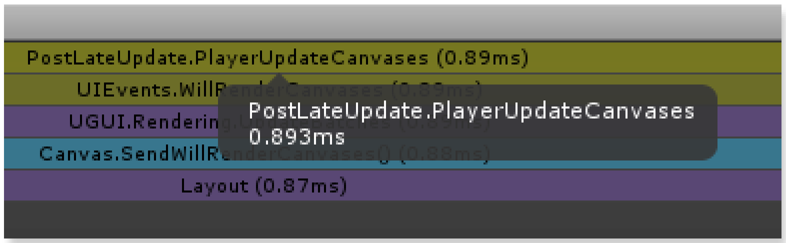
However, since nothing is really free, our tool does spend some time budget, but it does so only when updating its related UI. The tool can be configured to update more or less often by changing the **Update Interval** setting in the inspector.


Behaviour
Update Interval 0.5

Then how much does Lite FPS Counter consume overall?


Update.ScriptRunBehaviourUpdate (0.08ms)
BehaviourUpdate (0.08ms)
LiteFPSCounter.Update() (0.07ms)
LiteFPSCounter.Update()
0.069ms

Again, we turn to the integrated profiler and we see that the Update function is now spending 0.069 milliseconds (69 microseconds).


PostLateUpdate.PlayerUpdateCanvases (0.89ms)
UIEvents.WillRenderCanvases (0.89ms)
UGUI.Rendering
Canvas.SendWillRenderCanvases() (0.88ms)
Layout (0.87ms)
PostLateUpdate.PlayerUpdateCanvases
0.893ms

However, the UI update takes significantly longer - 0.893ms, but bear in mind that this figure includes also things not related to Lite FPS Counter.

If we add everything up, we get < 1ms, but we'll round up to 1ms, so that we also consider the 2 microseconds per frame our tool spends every frame.

Keep in mind however, **this 1ms is not per frame, but once per second**, so what does this mean to you? It means that only if your framerate is ridiculously high (like in the empty scene capture), you will start to experience framerate inaccuracy by adding the overhead of Lite FPS Counter.

To get a better idea of how framerate is affected, here's a table that shows the impact at different framerates.

Original FPS	Resulting FPS (with LFPSC overhead)	Delta / Drop FPS
10000	9990.01	9.99
5000	4995.005	4.995
1000	999.001	0.999
500	499.5	0.5
300	299.7	0.3
288	287.712	0.288
240	239.76	0.24
200	199.8	0.2
180	179.82	0.18
144	143.856	0.144
120	119.88	0.12
100	99.9	0.1
90	89.91	0.09
80	79.92	0.08
70	69.93	0.07
60	59.94	0.06
50	49.95	0.05
40	39.96	0.04
30	29.97	0.03
25	24.975	0.025
20	19.98	0.02
15	14.985	0.015
10	9.99	0.01
5	4.995	0.005
1	0.999	0.001

Table 1 - Framerate performance hit chart

The following formula was used for computing the delta FPS (FPS drop).

DELTA_FPS = FPS – (FPS * 1000 / (1000 + TIME_OVERHEAD_PER_SECOND))

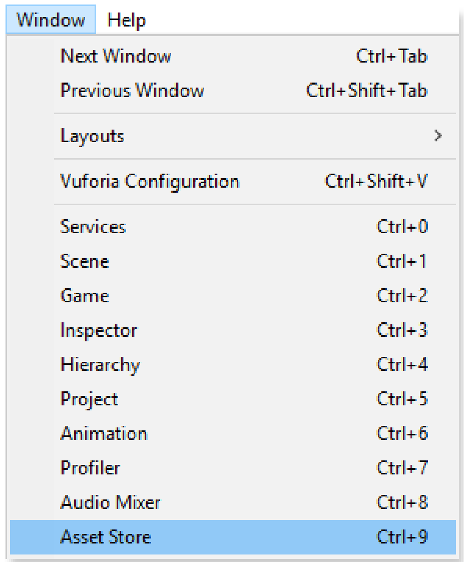
So, unless your game's framerate is ridiculously high to start with, the **performance overhead added by Lite FPS Counter is neglectable**.

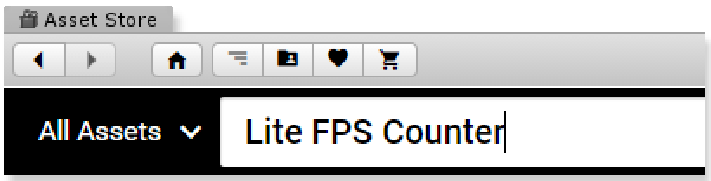
However, do take these values with a pinch of salt, as they are system-dependent.

Installation

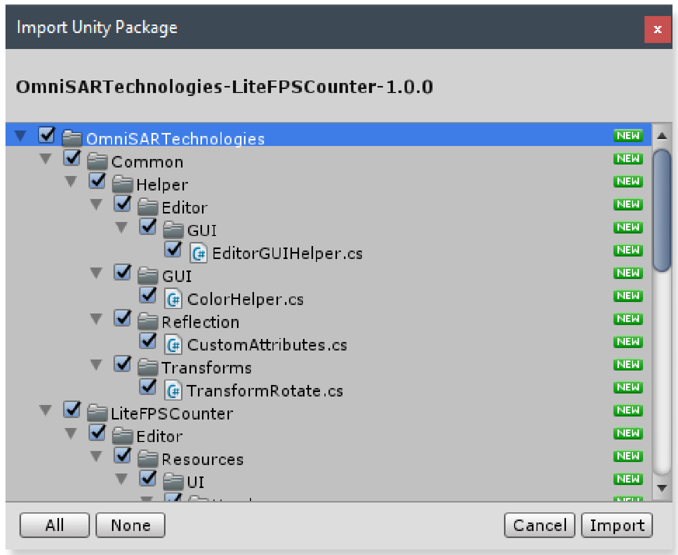
Installing from Unity Asset Store

Without a doubt, the most convenient way to install Lite FPS Counter is to get it from Asset Store right within the Unity editor. Just go to **Unity's main menu -> Window -> Asset Store** and search for **Lite FPS Counter** or **Lite FPS Counter (Pro)**.

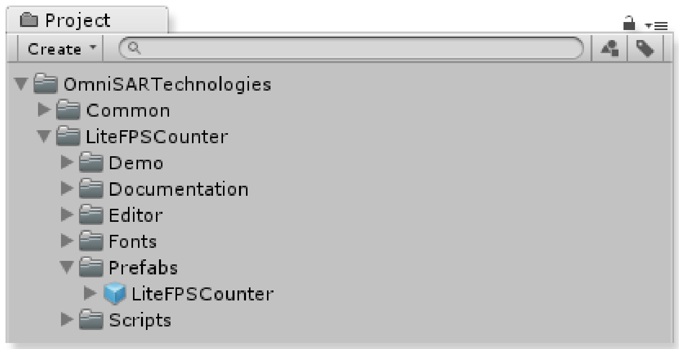




Download and **Import**. The **Import Unity Package** window will open. Make sure all the content is selected and click **Import**.

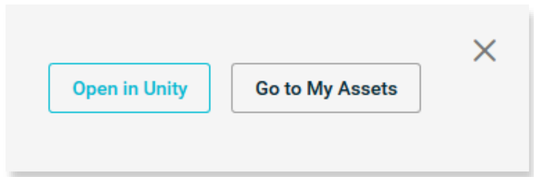


The importing process will start and at the end of it, you should be able to see the imported package content in your project structure.



```
OmniSARTechnologies
  Common
  LiteFPSCounter
    Demo
    Documentation
    Editor
    Fonts
    Prefabs
      LiteFPSCounter
    Scripts
```

Alternatively, you can follow the same procedure using a browser. Just go to Unity Asset Store and search for **Lite FPS Counter**. You'll be prompted with a popup that allows you to open the Unity package file with Unity.

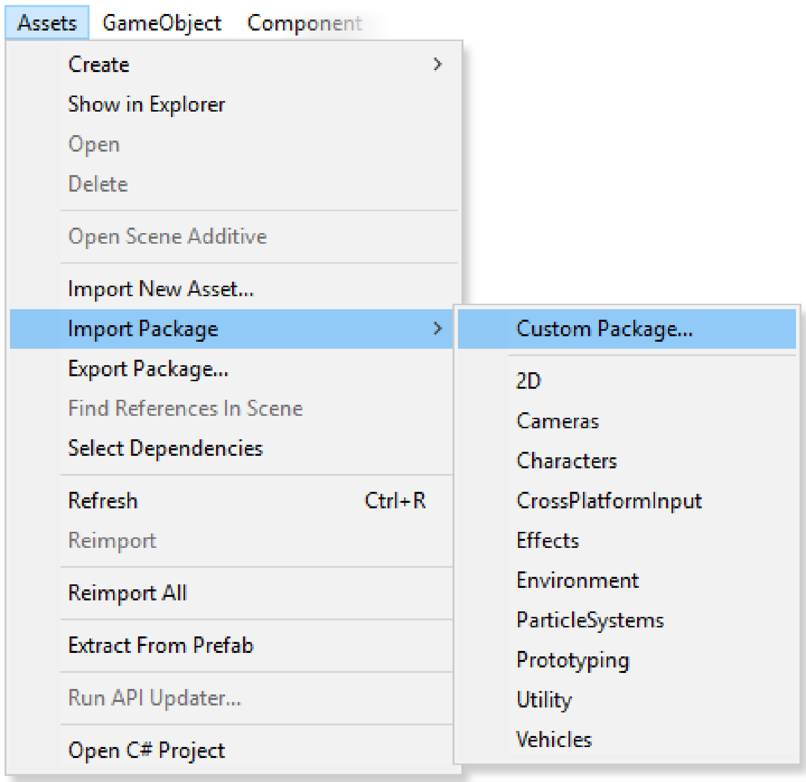


Installing from OmniSAR Technologies website

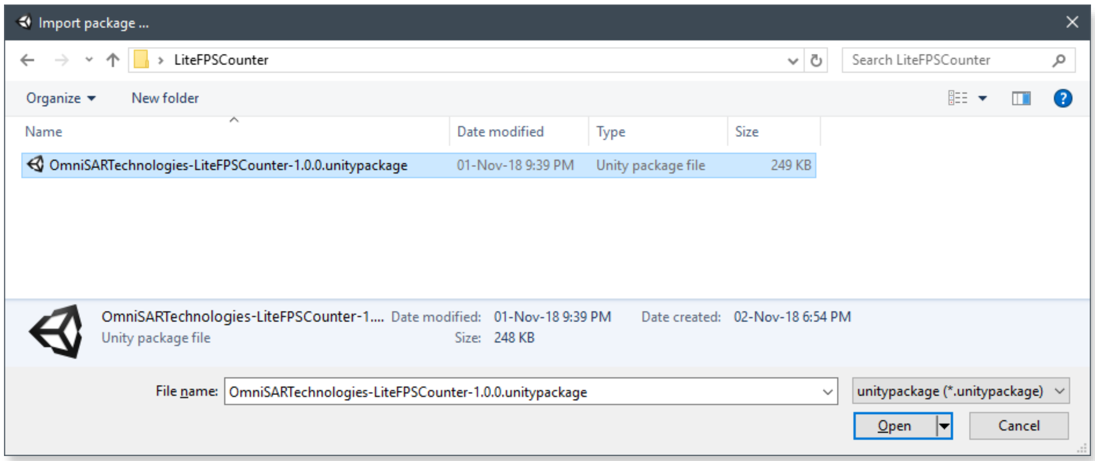
Go to https://www.omnisar.com/store/lite-fps-counter and download the archived package.

You will end up with a .7z file that you can extract anywhere you like. The archive should contain the Unity package file **OmniSARTechnologies-LiteFPSCounter-1.0.0.unitypackage** that you can import in your project.

To import the package into your project, follow the standard package importing procedure. Go to **Unity's main menu -> Asset -> Import Package -> Custom Package**.



In the **Import Package** dialog, go to where you've extracted the archive you downloaded from the **OmniSAR Technologies** website, select the **Lite FPS Counter unity package** and open it.

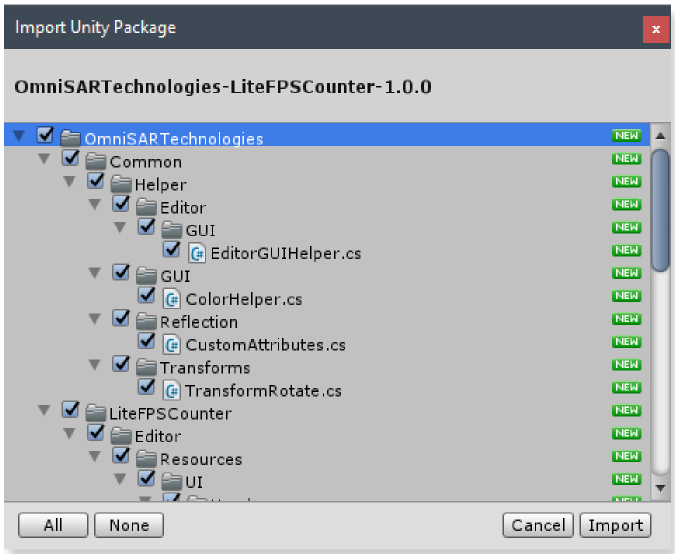


OmniSARTechnologies-LiteFPSCounter-1.0.0.unitypackage 01-Nov-18 9:39 PM Unity package file 249 KB

OmniSARTechnologies-LiteFPSCounter-1.... Date modified: 01-Nov-18 9:39 PM Date created: 02-Nov-18 6:54 PM
Unity package file Size: 248 KB

File name: OmniSARTechnologies-LiteFPSCounter-1.0.0.unitypackage
unitypackage (*.unitypackage)

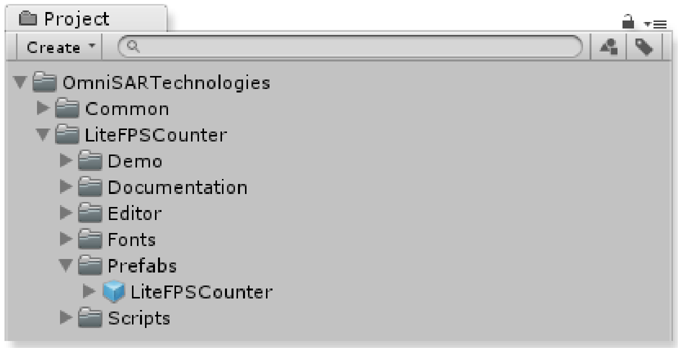
The **Import Unity Package** window will open. Make sure all the content is selected and click **Import**.



OmniSARTechnologies-LiteFPSCounter-1.0.0

```
✓ OmniSARTechnologies                NEW
  ✓ Common                           NEW
    ✓ Helper                         NEW
      ✓ Editor                       NEW
        ✓ GUI                        NEW
          ✓ EditorGUIHelper.cs       NEW
      ✓ GUI                          NEW
        ✓ ColorHelper.cs             NEW
      ✓ Reflection                   NEW
        ✓ CustomAttributes.cs        NEW
      ✓ Transforms                   NEW
        ✓ TransformRotate.cs         NEW
  ✓ LiteFPSCounter                   NEW
    ✓ Editor                         NEW
      ✓ Resources                    NEW
        ✓ UI                         NEW
```

The importing process will start and at the end of it, you should be able to see the imported package content in your project structure.



```
▼ OmniSARTechnologies
  ▶ Common
  ▼ LiteFPSCounter
    ▶ Demo
    ▶ Documentation
    ▶ Editor
    ▶ Fonts
    ▼ Prefabs
      ▶ LiteFPSCounter
    ▶ Scripts
```

Installing from other sources

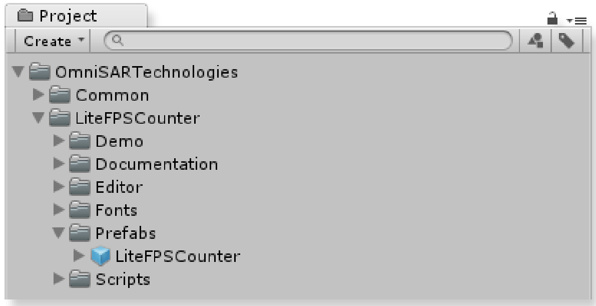
In case of a source other than Asset Store or OmniSAR Technology website, you may end up with either a Unity package file (.unitypackage) or the raw unpacked assets (the **OmniSARTechnology** directory containing all the assets of the Lite FPS Counter).

In case of a Unity package file (.unitypackage), follow the standard package importing procedure described in the Installing from the OmniSAR Technologies website section.

As a last resort, if all you have is the unpacked asset content, make sure you copy the **OmniSARTechnologies** directory in the **root for your project structure** (under **/Assets/**).

However, we strongly discourage importing the component's assets "by hand", as the integrity of the raw content could be compromised to begin with. We recommend installing Lite FPS Counter only from trusted sources like the Unity Asset Store or the OmniSAR Technologies website.

Regardless of your installation choice, your project structure should look like the following.



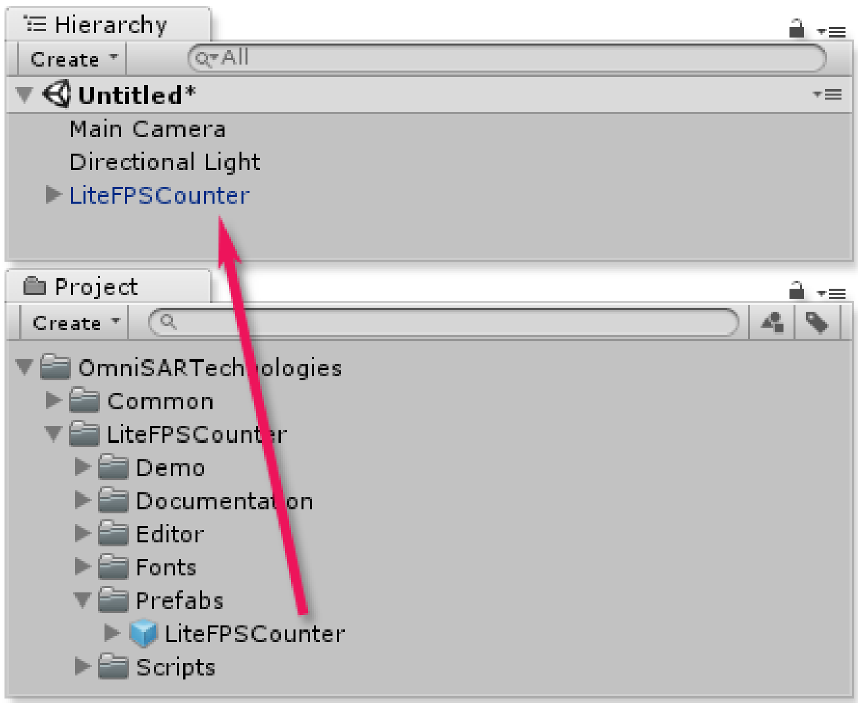
```
▼ OmniSARTechnologies
  ▶ Common
  ▼ LiteFPSCounter
    ▶ Demo
    ▶ Documentation
    ▶ Editor
    ▶ Fonts
    ▼ Prefabs
      ▶ LiteFPSCounter
    ▶ Scripts
```

Important! Do **NOT** move the **OmniSARTechnologies** directory somewhere else in the project structure or re-organize the package's structure in any way! It's very important that the Lite FPS Counter assets are placed in the locations they were designated to stay.

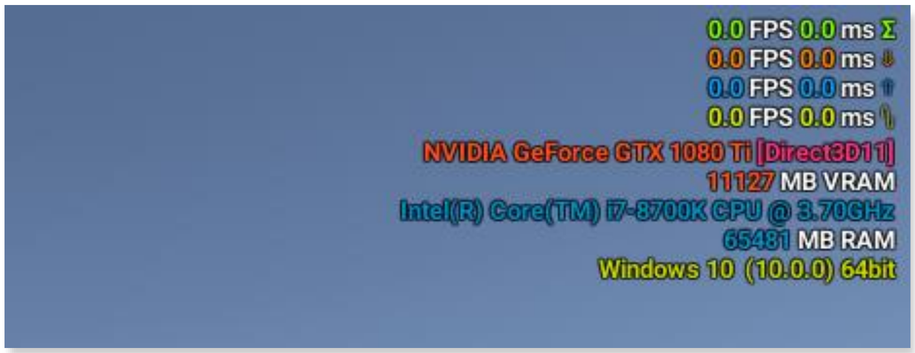
Scene Setup

Adding the Lite FPS Counter by prefab drag-and-drop

To add the Lite FPS Counter in a scene, you have two options. The first one consists of simply dragging and dropping the Lite FPS Counter prefab anywhere in the scene.



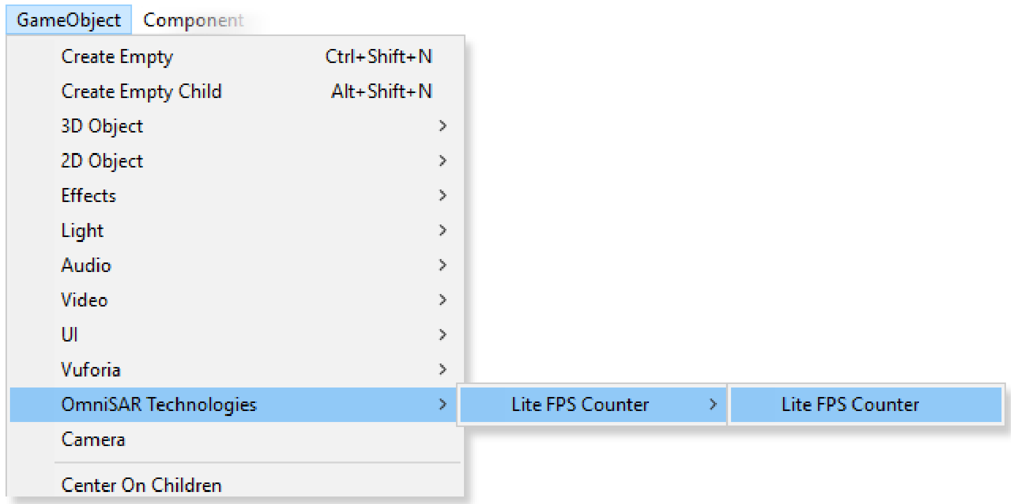
You should already be seeing the Lite FPS Counter UI in your **Game** view.



```
0.0 FPS 0.0 ms Σ
0.0 FPS 0.0 ms
0.0 FPS 0.0 ms
0.0 FPS 0.0 ms
NVIDIA GeForce GTX 1080 Ti [Direct3D11]
11127 MB VRAM
Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
65481 MB RAM
Windows 10 (10.0.0) 64bit
```

Adding the Lite FPS Counter using the Unity Game Object menu

The second option is to go to **Unity's main menu -> Game Object -> OmniSAR Technologies -> Lite FPS Counter -> ...** and chose Lite FPS Counter. A new Lite FPS Counter game object will be created in the scene and you should be seeing the Lite FPS Counter UI in the Game view.



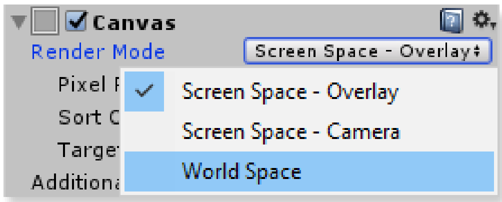
Please note that while dragging and dropping the prefab in the scene **will keep the link between the created scene game object and the prefab**, adding it using a menu option **will create an unlinked game object**. This will not affect by any means the functionality or performance of the Lite FPS Counter, but it's just something you may consider when setting up your scene.

Also note that adding the Lite FPS Counter by dragging and dropping the prefab in the scene comes with the potential risk of accidentally modifying the prefab if some values are tweaked and changes applied. Not necessarily a bad thing, but you'll lose the original prefab configuration.

If you're unsure of how prefabs work, use the **Game Object -> OmniSAR Technologies** menu to add the Lite FPS Counter in your scene.

Using Lite FPS Counter in VR

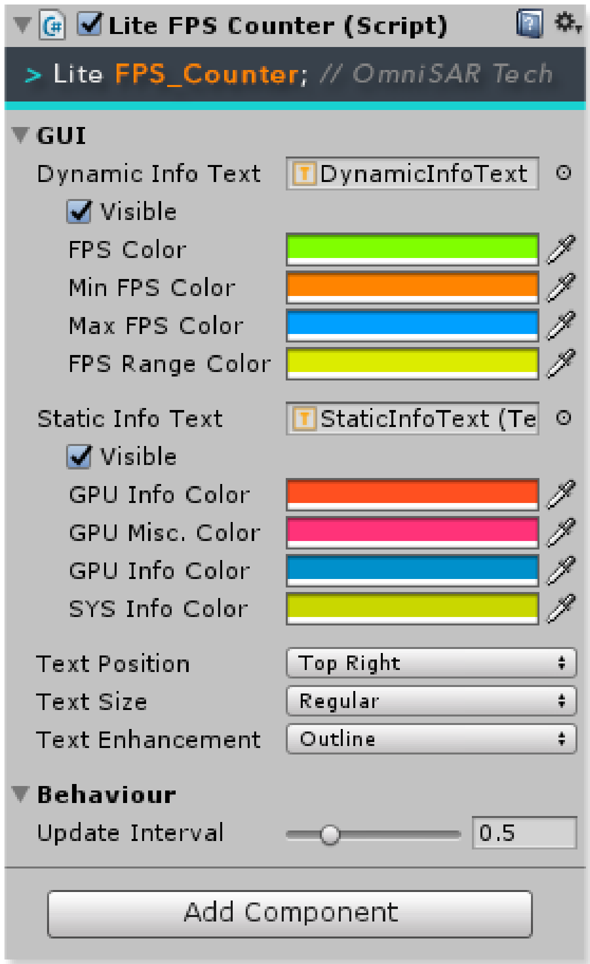
Add a Lite FPS Counter that has its own canvas. To make it work in VR, change Lite FPS Counter's **Canvas' Render Mode** to **World Space** (or **Screen Space - Camera**) and plug the camera of your choice that you need to render Lite FPS Counter with into the **Camera** reference.



Now you can move the canvas freely and you will see it in VR. Tip: Parent Lite FPS Counter's Canvas to your hand's transform for an interesting hand-HUD effect.

Configuration (Optional)

At this point, **the component should be up and running with no further configuration required**. However, there are some things you could change **if you wish to further customize it**. The component's inspector UI contains some settings you can tweak to your liking.


Lite FPS Counter (Script)

> Lite FPS_Counter; // OmniSAR Tech

GUI

Dynamic Info Text: DynamicInfoText
- ✓ Visible
- FPS Color: 🟩
- Min FPS Color: 🟧
- Max FPS Color: 🟦
- FPS Range Color: 🟨

Static Info Text: StaticInfoText (Te
- ✓ Visible
- GPU Info Color: 🟥
- GPU Misc. Color: 🟥
- GPU Info Color: 🟦
- SYS Info Color: 🟨

Text Position: Top Right
Text Size: Regular
Text Enhancement: Outline

Behaviour

Update Interval: 0.5

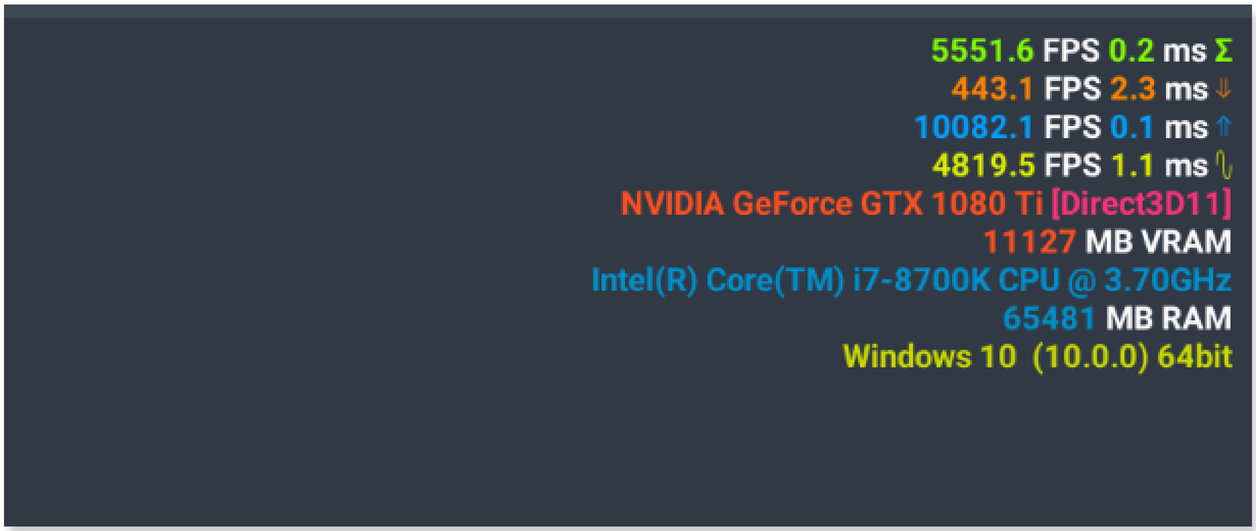
Dynamic Info Text - To mitigate the slow UI system, Lite FPS Counter splits its UI elements in 2 categories (static and dynamic), so that UI updates are kept to a minimum. This field represents a reference to the UI element that will draw the dynamic info like framerates and timing info. This reference is updated during the game execution periodically, with a frequency dictated by the **Update Interval** option.

Static Info Text - The static info text field represents a reference to the UI element that will draw the static information like system configuration. Since the system configuration doesn't change during the game execution, **this element is updated only once**, at the beginning of the game.

As a general rule, if you wish to use your own UI elements, make sure they are not expensive.

Visible* (Pro version only) - Controls the visibility of the referenced static / dynamic info text components.

Colors* (Pro version only) - Controls the colors of various information that is being displayed on the UI. A correspondence between these settings and the displayed information can be inferred by also taking a look at the following UI capture.



```
5551.6 FPS 0.2 ms Σ
443.1 FPS 2.3 ms ⇓
10082.1 FPS 0.1 ms ⇑
4819.5 FPS 1.1 ms ∿
NVIDIA GeForce GTX 1080 Ti [Direct3D11]
11127 MB VRAM
Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
65481 MB RAM
Windows 10  (10.0.0) 64bit
```

Text Position (Pro version only) - Controls where the text should be placed in screen-space. A range of (Left, Center / Right) x (Top, Center, Bottom) = 9 positions are available for selection.

Text Size (Pro version only) - Specifies how big or small the rendered text should be. Although the rendered text automatically scales to the rendering size, this setting adds an extra-advantage when accommodating certain pretentious game conditions or screen sizes.

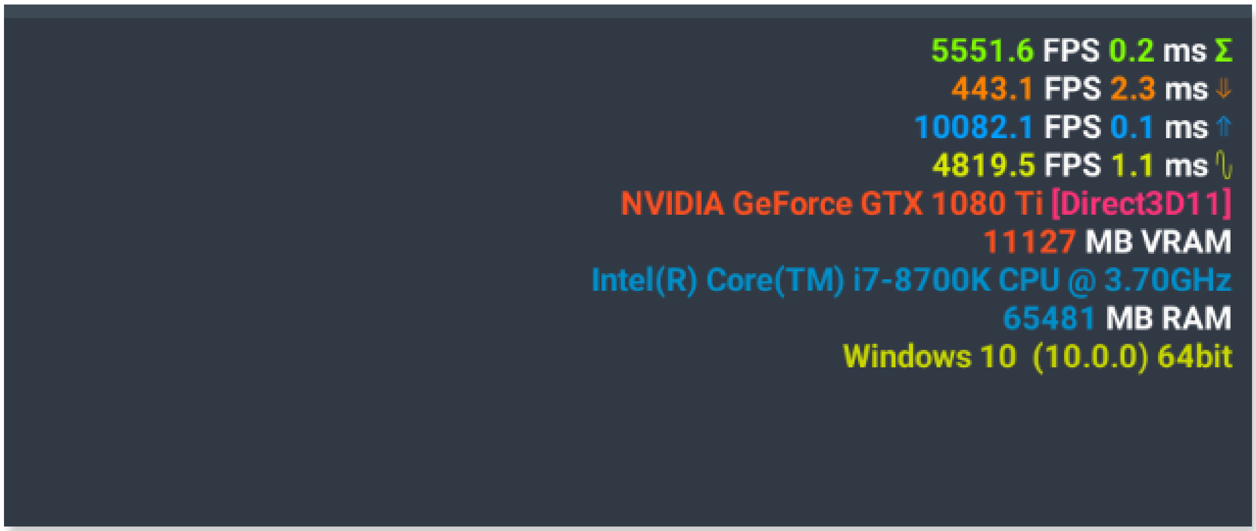
Text Enhancement (Pro version only) - Allows the selection of certain text enhancement techniques that can increase the text visibility / readability. Choose:

- **None** for the maximum performance, but sacrificing readability. With this option, the displayed info might be hard to read in certain conditions.
- **Shadow** as a fairly good compromise between visuals and performance. Although not as good as ***Outline***, the **Shadow** option still provides decent readability most of the time.
- **Outline** for the best visual enhancement, sacrificing a bit of performance (few ms ONCE every **Update Interval**). **This is the recommended option.**
- **All Combined** for even more pronounced readability, although not much different than ***Outline***.

Update Interval (Pro version only) - Specifies in seconds how often the component updates its UI. Refer to section Performance Evaluation to get an idea of how this option impacts the overall performance of Lite FPS Counter. In the free version, the **Update Interval** is 1 second.

Interpreting the UI

Besides the framerate, Lite FPS Counter displays some additional information.



```
                5551.6 FPS 0.2 ms Σ
                 443.1 FPS 2.3 ms ⇓
               10082.1 FPS 0.1 ms ⇑
                4819.5 FPS 1.1 ms ∿
   NVIDIA GeForce GTX 1080 Ti [Direct3D11]
                          11127 MB VRAM
Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
                           65481 MB RAM
              Windows 10  (10.0.0) 64bit
```

The displayed UI contains the following sections (top to bottom):

- **Timing info**
    - **Average framerate / Average frame time** - This is the average framerate and frame time recorded within the last **Update Interval**. This line is marked with the Greek letter Sigma (Σ) to indicate that the information represents an integration.
    - **Minimum framerate / Maximum frame time** - The lowest framerate and the highest frame time recorded during the last **Update Interval**. The line is marked with a down-arrow sign to indicate that the information represents a performance minimum.
    - **Maximum framerate / Minimum frame time** - The highest framerate and the lowest frame time recorded during the last **Update Interval**. The line is marked with an up-arrow sign to indicate that the information represents a performance maximum.
    - **Framerate fluctuation / Frame time fluctuation** - This line displays the half delta between the maximum and the minimum framerate and frame time. **Please note** that since the relation between time and framerate is non-linear, this line does not follow the inverse relation rule between framerate and frame time as the first 3 lines do. That is, it's normal that the framerate fluctuation is not always equal to 1000 / frame time fluctuation.
- **System configuration info**
    - **GPU name [Interface]** - This line displays the name and the graphics interface used.

- **GPU memory capacity** - Shows the amount of video graphics memory (VRAM) in megabytes the GPU is equipped with.
- **CPU name** - Shows the name of the CPU the system is equipped with.
- **CPU memory capacity** - Shows the amount of installed system memory (RAM) in megabytes.
- **Operating System** - Displays brief information about the running Operating System.

Product Roadmap

Here are some features we are working on and may be available in future updates:

- **Framerate and frame time histograms** - For now, Lite FPS Counter only shows minimal insights about the dynamics that naturally occur with frame rate and frame time (min, max and general fluctuation) which is a bit vague and does not depict a full image of what's really going on. A far more useful piece of information would be offered with **histograms**. The histograms would show a more accurate representation of **framerate and frame time values distribution**. If your histogram is not convergent around your average framerate, or worse - if the histograms show a standalone peak far from the average framerate, you'll know immediately that there is a periodic hiccup that your game experiences. Since the histogram would basically plot **framerate vs framerate frequency**, you'll know exactly how frequent and how long the hiccups are.
- **Framerate and frame time spectral analysis** - Doing a spectral analysis on the framerate and frame time values would give an even deeper insight about how often the framerate and frame time fluctuate. **An ideal spectral graph would be a flat line with a single spike at 0Hz.** This would mean that your framerate is as steady as it can be. A spike anywhere else in the graph would mean your framerate fluctuates with that exact frequency and this may be an indicator for some periodic events that induce periodic performance loss.
- **Conditional events** - Upon reaching certain conditions, events could be fired. This could be useful to either:
    - **Take in-game actions** like changing the quality settings on the fly to meet the new load.
    - **Issue a debug log call** and store the event for future analysis.

Plug

You can run a Windows demo to get a better idea of Lite FPS Counter in action:
https://www.dropbox.com/s/5soc1lgrgqekl50/LiteFPSCounterDemo-1.0.0-win_x86_64.7z?dl=0

Here's a link to this document, should you need it:
https://www.dropbox.com/s/jhr0b0se68lpgws/LiteFPSCounter-Manual.pdf?dl=0

For other OmniSAR goodies, visit:

- https://www.omnisar.com or
- our Asset Store publisher page

If RSS is your thing, here's the one that will keep you informed about **our Asset Store activity**:

publisher.assetstore.unity3d.com/feed/omni-sar-technologies/L2rkPIHuzAIoX267C25WOfkA86M/activity.rss

Ultimately, follow our **Facebook page** (**https://www.facebook.com/OmniSARTechnologies**) to get exciting news about the cool stuff that we do.

And remember, game development is fun!