# Unity Statistics VR HUD and Debug Tool

**Download package here:**
**https://github.com/ryanmimmersive/Unity_Stats_VR_HUD**
**Statistics HUD**
This tool was created for quick and easy issue identification and optimization inside VR, it aims to make the process easier while using HMDs. The Unity Editor provides useful information in its dropdown “Stats” window, however this overlay is not visible in VR. This results in having to remove the headset or partially wearing it to see these useful real-time values. This tool takes the most important values: FPS, batches, set-pass calls, tris and puts them in a world space canvas so they can be monitored easily in real-time while in VR. It also includes additional useful features for debugging.


The display as seen here matches closely that values shown in the stats window. Most importantly the FPS as displayed in the stats window attempts to emulate the FPS of a built project. It excludes the editor windows and only accounts for the game window in Unity to do this, standard approaches to creating FPS counters in Unity do not account for this. The FPS value shown on the HUD uses Unity Statistics to display the “Estimated FPS for Build” giving an estimation of the built FPS without having to do a length build to see this in VR.

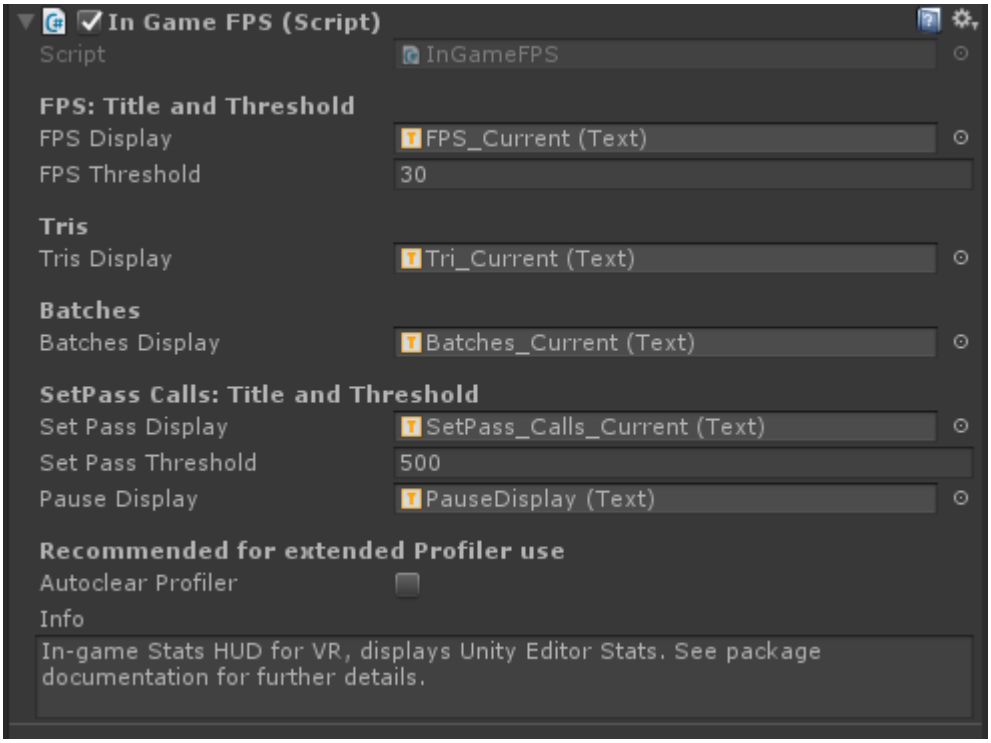
In the inspector an FPS threshold can be set, such as 60 FPS, when the estimated game FPS falls bellow this the FPS value will go red giving the user an indication that there has been a drop in FPS.

**Profiler**

Generally, the Profiler is hooked into Dev builds, this of course requires building the project to look for issues. This tool has been optimized to work alongside the profiler in the editor view, without building.

*When using the Profiler, the HUD "estimated FPS" will be affected by this process, unlike other editor windows.*

When working with the profiler it is recommended to use the "Autoclear Profiler" option. This will clear the profiler buffer after a certain point, instead of saving a log of every frame. This is to stop large FPS drops and performance issues caused by the Profiler chugging through a large amount of frame data.




**Autopause**

Pressing 'A' on the Oculus Touch controller will toggle the Autopause feature, this will automatically pause the game when FPS drops below your set FPS Threshold. This will allow easy checking of the HUD and Profiler without rushing to pause and check the data.

After unpausing the game Autopause will be disabled, simply press A to activate it again. It is recommended to set an FPS threshold lower that the traditional 60FPS as the profiler may affect estimated FPS and this will only pause the game to catch large spikes.

IMMERSIVE EDUCATION
LEARN THROUGH EXPERIENCE

**Summary**

- World space real-time Unity stats.
- Profiler optimization.
- FPS threshold for FPS drop notifications.
- FPS Autopause to allow easy removal of HMD and profiler spike checking.

**Code**

```
using UnityEngine;
using UnityEditor;
using UnityEngine.UI;
using System.Collections;

public class InGameFPS : MonoBehaviour
{
    private bool m_isShown = true;
    private bool dontDelete = true;
    private int m_FrameCounter;
    private float m_ClientTimeAccumulator;
    private float m_RenderTimeAccumulator;
    private float m_MaxTimeAccumulator;
    private float m_ClientFrameTime;
    private float m_RenderFrameTime;
    private float m_MaxFrameTime;

    [Header("FPS: Title and Threshold")]
    public Text FPSDisplay;
    public float FPSThreshold = 60f;
    private float FPS;
    private bool FPSCatcher = false;
    [Header("Tris")]
    public Text TrisDisplay;
    [Header("Batches")]
    public Text BatchesDisplay;
    [Header("SetPass Calls: Title and Threshold")]
    public Text SetPassDisplay;
    public float SetPassThreshold = 1000f;
    private bool SPCatcher = false;
    [Header("Press A to enable AutoPause when below FPS threshold")]
    private bool DropPause = false;
    public Text pauseDisplay;
    [Header("Recommended for extended Profiler use")]
    public bool autoclearProfiler = false;
    [TextArea(2, 10)]
    public string info;


    // IF UNPAUSED DELETE EXTRA PROFILER FRAMES
    private void OnApplicationFocus(bool focus)
    {
        if(focus == true)
        {
            dontDelete = false;
```

```
        }
    }
    // UPDATE
    public void Update()
    {
        // STOP PROFILER CHUGGING
        if (autoclearProfiler == true)
        {
            if (UnityEditorInternal.ProfilerDriver.firstFrameIndex > 0 && dontDelete
== false)
            {
                UnityEditorInternal.ProfilerDriver.ClearAllFrames();
            }
        }
        // TOGGLE AUTOPAUSE
        if (OVRInput.GetDown(OVRInput.Button.One)) // *** Edit this line for different
input methods ***
        {
            if (DropPause == false)
            {
                pauseDisplay.text = "AUTOPAUSE: ON";
                pauseDisplay.color = Color.red;
                DropPause = true;
            }
            else
            {
                pauseDisplay.text = "AUTOPAUSE: OFF";
                pauseDisplay.color = Color.white;
                DropPause = false;
            }
        }
        // IF FPS FALLS BELOW CHOSEN THRESHOLD
        if (FPS < FPSThreshold)
        {
            FPSDisplay.color = Color.red;
            FPSCatcher = false;
            if (DropPause == true)
            {
                StartCoroutine(WaitForPause());
            }
        }
       else
        {
            if (FPSCatcher == false)
            {
                FPSDisplay.color = Color.white;
                FPSCatcher = true;
            }
        }
        // IF SET PASS CALLS FALLS BELOW CHOSEN THRESHOLD
        if (UnityStats.setPassCalls > SetPassThreshold)
        {
            SetPassDisplay.color = Color.red;
            SPCatcher = false;
        }
        else
        {
            if (SPCatcher == false)
            {
                SetPassDisplay.color = Color.white;
                SPCatcher = true;
```

```
            }
        }
    }
    // PAUSE FUNCTION
    private IEnumerator WaitForPause()
    {
        dontDelete = true; // Dont clear the profiler
        pauseDisplay.text = "AUTOPAUSE: OFF";
        pauseDisplay.color = Color.white;
        DropPause = false;
        yield return new WaitForSeconds(0.2f);
        Debug.Break(); // Pause editor
    }
    // RUN
    void OnGUI()
    {
        if (m_isShown)
        {
            GameViewStatsGUI();
        }
    }
    // FORMAT TRI DISPLAY
    private string FormatNumber(int num)
    {
        if (num < 1000)
        {
            return num.ToString();
        }
        if (num < 1000000)
        {
            return ((double)num * 0.001).ToString("f1") + "k";
        }
        return ((double)num * 1E-06).ToString("f1") + "M";
    }
    // UPDATE FPS
    public void UpdateFrameTime()
    {
        float frameTime = UnityStats.frameTime;
        float renderTime = UnityStats.renderTime;
        m_ClientTimeAccumulator += frameTime;
        m_RenderTimeAccumulator += renderTime;
        m_MaxTimeAccumulator += Mathf.Max(frameTime, renderTime);
        m_FrameCounter++;
        bool flag = m_ClientFrameTime == 0f && m_RenderFrameTime == 0f;
        bool flag2 = m_FrameCounter > 30 || m_ClientTimeAccumulator > 0.3f ||
m_RenderTimeAccumulator > 0.3f;
        if (flag || flag2)
        {
            m_ClientFrameTime = m_ClientTimeAccumulator / (float)m_FrameCounter;
            m_RenderFrameTime = m_RenderTimeAccumulator / (float)m_FrameCounter;
            m_MaxFrameTime = m_MaxTimeAccumulator / (float)m_FrameCounter;
        }
        if (flag2)
        {
            m_ClientTimeAccumulator = 0f;
            m_RenderTimeAccumulator = 0f;
            m_MaxTimeAccumulator = 0f;
            m_FrameCounter = 0;
        }
    }
    // UPDATE STATS
```

```
    public void GameViewStatsGUI()
    {
        UpdateFrameTime();
        // Display FPS
        FPS = 1f / Mathf.Max(m_MaxFrameTime, 1E-05f);
        FPSDisplay.text = string.Format("{0:F1} FPS  ({1:F1}ms)", 1f /
Mathf.Max(m_MaxFrameTime, 1E-05f), m_MaxFrameTime * 1000f);
        // Display Batches
        BatchesDisplay.text = UnityStats.batches.ToString();
        // Display Tris
        TrisDisplay.text = FormatNumber(UnityStats.triangles);
        // Display SetPass
        SetPassDisplay.text = UnityStats.setPassCalls.ToString();
    }
       }
```