Performance Analysis

v. 0.4

This document contains information specific to application performance analysis. While this document is geared toward native application development, most of the tools presented here are also useful for improving performance in Android applications developed in Unity or other engines.

1. Application Performance

1.1 FPS Report

The number of application frames per second, the GPU time to render the last eye scene rendering, and the GPU time to to render the eye TimeWarp are reported to logcat once per second (for more information on logcat, see *Android Debugging*). Note that the GPU time reported does not include the time spent resolving the rendering back to main memory from on-chip memory, so it is an underestimate.

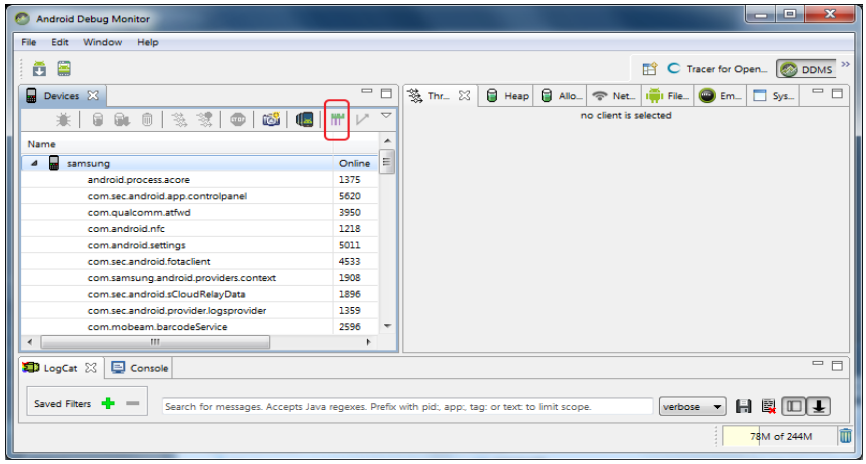
If the reported GPU time is over about 14 milliseconds, actual drawing is limiting the frame rate, and the resolution / geometry / shader complexity will need to be reduced to get back up to 60 FPS. If the GPU time is okay and the application is still not at 60 FPS, then the CPU is the bottleneck. In Unity, this could be either the UnityMain thread that runs scripts, or the UnityGfxDevice thread that issues OpenGL driver calls. Systrace is a good tool for investigating CPU utilization. If the UnityGfxDevice thread is taking longer blocks of time, reducing the number of draw call batches is the primary tool for improvement.

1.2 SysTrace

SysTrace is the profiling tool that comes with the Android Developer Tools (ADT) Bundle. SysTrace can record detailed logs of system activity that can be viewed in the Google Chrome browser.

With SysTrace, it is possible to see an overview of what the entire system is doing, rather than just a single app. This can be invaluable for resolving scheduling conflicts or finding out exactly why an app isn't performing as expected.

Under Windows: the simplest method for using SysTrace is to run the **monitor.bat** file that was installed with the ADT Bundle. This can be found in the ADT Bundle installation folder (e.g., C:\android\adt-bundle-windows-x86_64-20131030) under the sdk/tools folder. Double-click monitor.bat to start Android Debug Monitor.



[Android Debug Monitor]

Select the desired device in the left-hand column and click the icon highlighted in red above to toggle Systrace logging. A dialog will appear enabling selection of the output .html file for the trace. Once the trace is toggled off, the trace file can be viewed by opening it up in Google Chrome.

You can use the WASD keys to pan and zoom while navigating the HTML doc. For additional keyboard shortcuts, please refer to the following documentation:
http://developer.android.com/tools/help/systrace.html

1.3 NDK Profiler

The Android NDK profiler is a port of gprof for Android.

The latest version, which has been tested with this release, is 3.2 and can be downloaded from the following location:
https://code.google.com/p/android-ndk-profiler/

Once downloaded, unzip the package contents to your NDK sources path, e.g.: C:\Dev\Android\android-ndk-r9c\sources.

Add the NDK prebuilt tools to your PATH, e.g.:
C:\Dev\Android\android-ndk-r9c\toolchains\arm-linux-androideabi-4.8\prebuilt\windows-x86_64\bin.

Android Makefile Modifications

1. Compile with profiling information and define NDK_PROFILE

```
LOCAL_CFLAGS := -pg -DNDK_PROFILE
```

2. Link with the ndk-profiler library

```
LOCAL_STATIC_LIBRARIES := android-ndk-profiler
```

3. Import the android-ndk-profiler module

```
$(call import-module,android-ndk-profiler)
```

Source Code Modifications

Add calls to **monstartup** and **moncleanup** to your Init and Shutdown functions. Do not call monstartup or moncleanup more than once during the lifetime of your app.

```
#if defined( NDK_PROFILE )
extern "C" void monstartup( char const * );
extern "C" void moncleanup();
#endif

extern "C" {

void Java_oculus_VrActivity2_nativeSetAppInterface( JNIEnv * jni, jclass clazz ) {

#if defined( NDK_PROFILE )
        setenv( "CPUPROFILE_FREQUENCY", "500", 1 ); // interrupts per second, default 100
        monstartup( "libovrapp.so" );
#endif

        app->Init();
}

void Java_oculus_VrActivity2_nativeShutdown( JNIEnv *jni ) {

        app->Shutdown();

#if defined( NDK_PROFILE )
        moncleanup();
#endif
}

} // extern "C"
```

Manifest File Changes

You will need to add permission for your app to write to the SD card. The gprof output file is saved in /sdcard/gmon.out.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Profiling your App

To generate profiling data, run your app and trigger the moncleanup function call by pressing the Back button on your phone. Based on the state of your app, this will be triggered by `OnStop()` or `OnDestroy()`. Once moncleanup has been triggered, the profiling data will be written to your Android device at /sdcard/gmon.out.

Copy the gmon.out file to the folder where your project is located on your PC using the following command: `adb pull /sdcard/gmon.out`

To view the profile information, run the gprof tool, passing to it the non-stripped library, e.g.:

```
arm-linux-androideabi-gprof  obj/local/armeabi/libovrapp.so
```

For information on interpreting the gprof profile information, see the following: http://sourceware.org/binutils/docs/gprof/Output.html

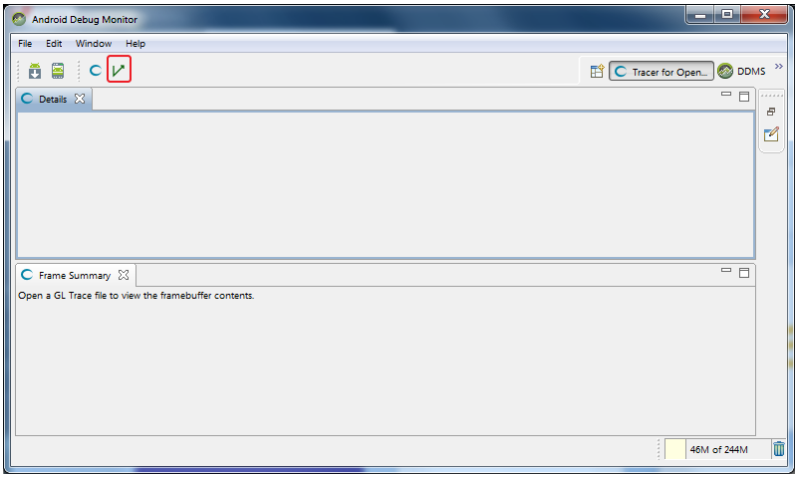
2. Rendering Performance: **Tracer for OpenGL ES**

Tracer is an additional profiling tool that comes with the ADT Bundle. It is used to capture OpenGL ES calls for an application.

Under Windows: the simplest method for using Tracer is to run the **monitor.bat** file that is installed with the ADT Bundle. This can be found in the ADT bundle installation folder (e.g., C:\android\adt-bundle-windows-x86_64-20131030) under the sdk/tools folder. Just double-click monitor.bat to start Android Debug Monitor.

Go to *Windows -> Open Perspective…* and select *Tracer for OpenGL ES*.

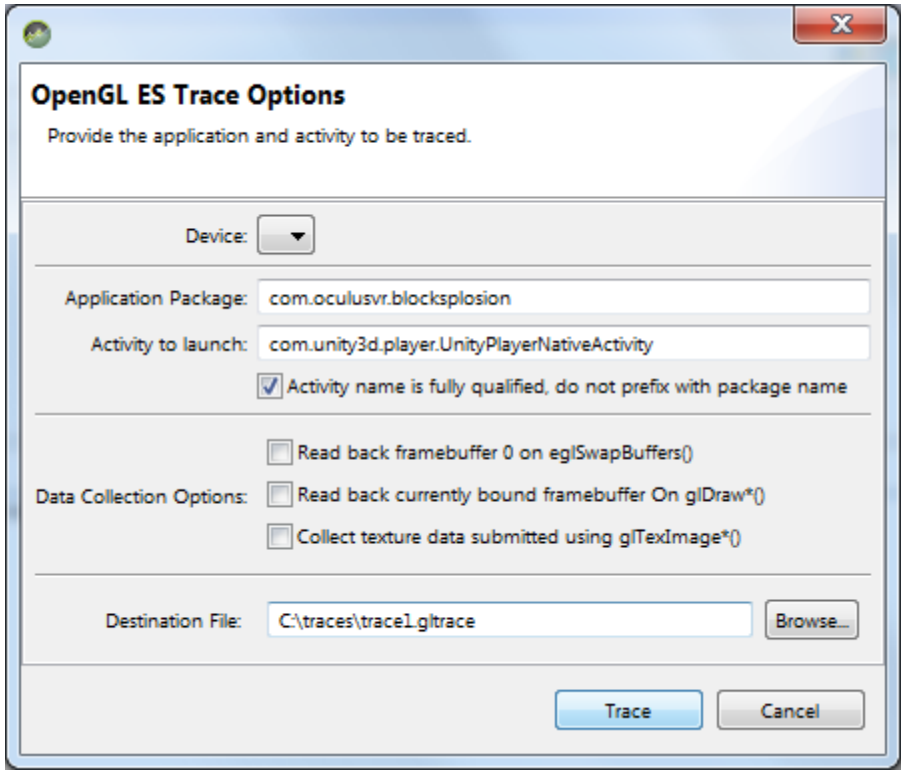
Click the Trace Capture button shown below.



[Tracer for OpenGL ES]

Fill in the Trace Options and select Trace.

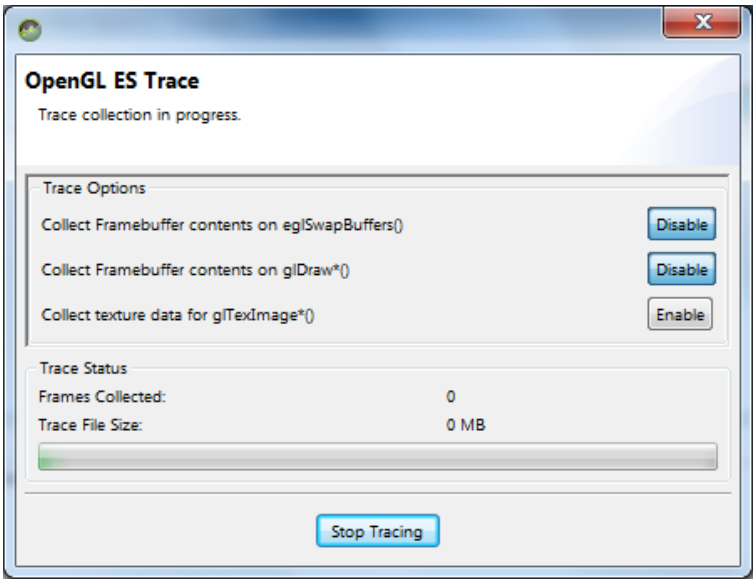
Note: A quick way to find the package name and Activity is to launch your app with logcat running. The Package Manager getActivityInfo line will display the information, e.g., com.Oculus.Integration/com.unity3d.player.UnityPlayerNativeActivity.



[Tracer for OpenGL ES]

Note: Selecting any Data Collection Options may cause the trace to become very large and slow to capture.

Tracer will launch your app and begin to capture frames.

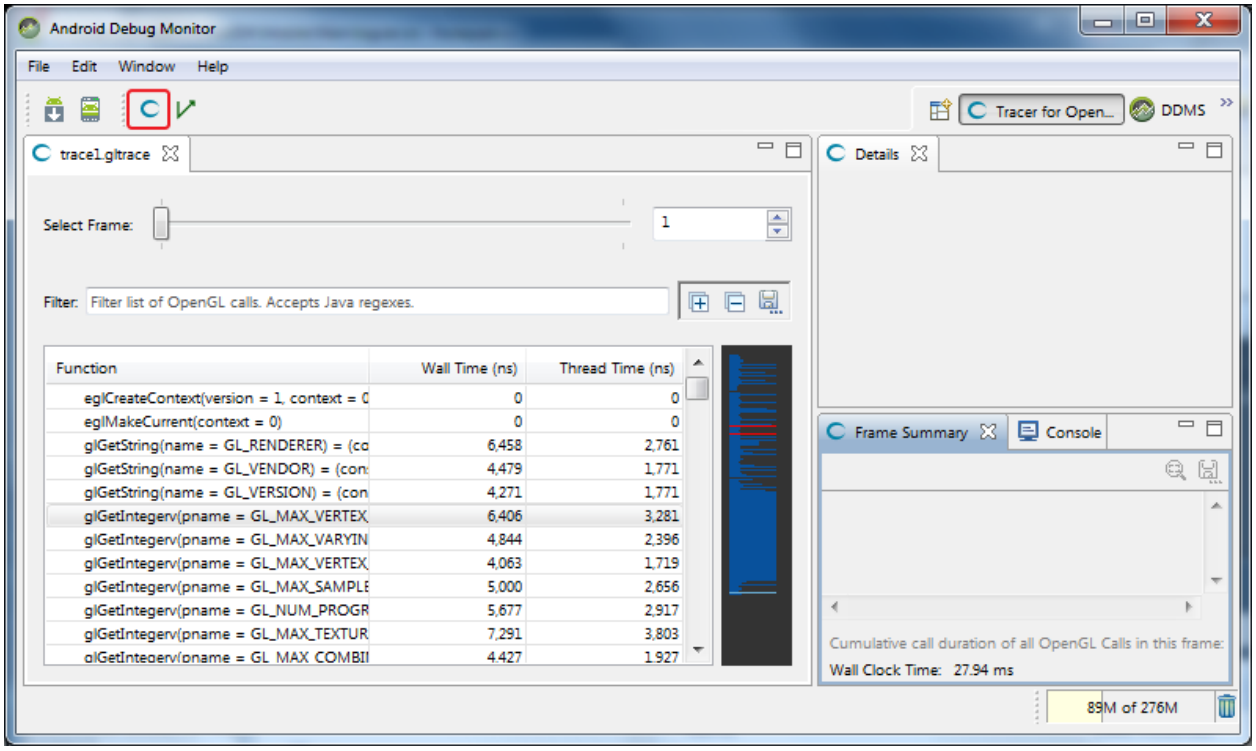


[Tracer for OpenGL ES]

Enable the following options:

- Collect Framebuffer contents on eglSwapBuffers()
- Collect Framebuffer contents on glDraw*()

Once you have collected enough frames, choose *Stop Tracing*.



[Tracer for OpenGL ES]

Select the button highlighted above in red to import the .gltrace file you just captured. This can be a lengthy process. Once the trace is loaded, you can view GL commands for every captured frame.

Last Update: Nov 10, 2014