Qualcomm Technologies, Inc.

Qualcomm® Snapdragon™ Virtual Reality SDK

User Guide

June 11, 2018

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision history

Revision	Date	Description
A	May 2016	Initial release
B	Sept 2016	VR SDK 1.1 Update
C	March 2017	VR SDK 2.0 Update
D	July 2017	VR SDK 2.1 Update
E	June 2018	VR SDK 3.0 Update

Contents

Tables

1 Introduction

1.1 Purpose

This document provides information about installing the Qualcomm® Snapdragon™ Virtual Reality (VR) SDK.

1.2 Background

High performance, low latency virtual reality on Android requires access to numerous new features and software optimizations on the target platform. The Snapdragon VR SDK provides access to these features which would otherwise be unavailable to developers.

In addition, the Snapdragon VR SDK implements many of the core low latency VR rendering functionality developers require to create high quality VR content.

The core Snapdragon VR SDK features include:

- Asynchronous timewarp
    - Barrel distortion
    - Chromatic aberration correction
    - Display stabilization/reprojection
    - Single buffered rendering
    - Layering (masks/overlays)
- 3DOF/6DOF sensor fusion (DSP, 1000 Hz)
- Stereo eye tracking (DSP, 120 Hz)
- CPU/GPU power management

2 Installation

To install the Snapdragon VR SDK, unzip the distribution package to any preferred location.

2.1 System requirements

A Snapdragon 835-based device running Android 7.1 loaded with the LA 2.0 (1463+) CRM, or later is required to utilize the Snapdragon VR SDK libraries.

2.2 SDK contents

The Snapdragon VR SDK contains the items listed in Table 2-1.

Table 2-1 SDK contents

/3rdParty	3rd party libraries used by the SDK
/controllers	Device services
/doc	SDK API documentation
/framework	Common utility code used by the SDK samples
/samples	Samples demonstrating use of the SVR SDK APIs
/svrApi	Native C/C++ headers and libraries for interfacing with the SDK APIs
/unity	Unity plug-in needed to create Unity applications utilizing the SDK

3 Using the SDK

3.1 Integrating with native OpenGL ES applications

3.1.1 SDK libraries

After installing the Snapdragon VR SDK, modify the application's "Android.mk" file to pull in "libsvrapi.so".

This is done using the "PREBUILT_SHARED_LIBRARY" and "LOCAL_SHARED_LIBRARIES" commands in the Android NDK build system.

See the Snapdragon VR sample or the Android NDK build documentation for more information.

Manually copy the file "svrApi.jar" from the Snapdragon VR SDK to the "libs" directory in your application.

3.1.2 Application manifest

In the application's build file "AndroidManifest.xml", change the "<application>" entry to include "android:hasCode="true"".

This tells the Android build system that there will be Java code in the native application.

If this is not done, the Android build system will not add the "svrApi.jar" file to the final APK.

3.1.3 Application Java activity

To include the Snapdragon VR SDK jar file in the final APK, your application must have at least a shell Java Activity class.

This can be nothing more than an Activity that extends "android.app.NativeActivity". Its only function can be to call "System.loadLibrary("svrapi")" and your application's library. Be sure to modify "AndroidManifest.xml" to change the Activity name to this new Java class.

3.1.4 Android activity lifecycle

Add the following Snapdragon VR calls to your native code:

android_main()

- Call "svrInitialize()" before entering the main loop
- Call "svrShutdown()" before the application exits

On APP_CMD_RESUME

- Call "svrBeginVr()" if VR has been initialized

On APP_CMD_PAUSE

Call "svrEndVr()" if VR has been initialized and "svrBeginVr()" has been called

3.1.5 Main application loop

At the beginning of each frame, call "svrGetPredictedDisplayTime()" to request the estimated time from the current point to the point when the frame being generated will appear on the display.

The predicted time should be supplied to the "svrGetPredictedHeadPose()" to get a quaternion representation of the current head orientation (and position if the device is equipped for 6DOF).

See the Snapdragon VR API documentation for further details on these functions.

3.1.6 Sample SimpleVR

The sample SimpleVR demonstrates the use of the Snapdragon VR SDK.

To compile and install the application.

- ➢ gradlew build

3.2 Unity integration

3.2.1 Scene setup

1. From a new or existing project, import the Snapdragon VR SDK package from **Assets** > **Import Package** > **Custom Package**, selecting the **/unity/svrUnityIntegration[32|64]bit.unitypackage** file from SDK release. Use 32bit for Unity 2017 and earlier; 64bit for Unity 2018 and later.
2. In the **Unity** dialog, leave all boxes checked and select **Import**.
3. Suggest **Close** and then **Reopen** Unity to sync references in prefabs.
4. Create an instance of a SvrCamera rig by dragging the prefab from **/Assets/SVR/Prefabs** into the scene.
5. Please try to avoid adding game objects to SvrCamera hierarchy to make SDK updates easier. Instead, use Transform SvrManager.Instance head and gaze to sync objects to head and eyes, respectively, instead.

Note that execution in the editor will display the message “No cameras rendering”. It is a known Unity issue that can be ignored.

3.2.2 Player settings

1. Select **File** >**Build Settings** > **Android** and choose **Player Settings**.
2. In the *Resolution and Presentation* section, set the *Default Orientation*.
   a. Landscape Left for Snapdragon 845
   b. Landscape Right for Snapdragon 835

3. In the *Other Settings* section, **Virtual Reality Supported** should not be set.

3.2.3 Quality settings

1. Select **Edit** > **Project Settings** > **Quality** to select the following recommended options:
   - *Anisotropic Textures* = **Per Texture**
   - *Anti Aliasing* = **Disabled**

     Anti Aliasing can be enabled in the properties of the SvrCamera node instantiated in the scene. Note that HDR must be disabled on the SvrCamera for anti-aliased eye buffers to be properly created.
   - *V Sync Count* = **Don't Sync**

3.2.4 Sample BoxWorld

The sample BoxWorld contains four unity scenes: boxWorld, overlayWorld, planeWorld and gridWorld. The Android BACK button or Windows Escape key will advance to the next scene and exit the application. In the unity editor, right mouse button will control the orientation, and center mouse button the position, of the head. SvrManager.Settings.TrackEyes is enabled for all but the first scene.

1. boxWorld.unity demonstrates the simple use of stereo left/right eye layers. In the SvrCamera hierarchy, only the 'Eye Left' and 'Eye Right' game objects are active - the others are disabled. Upon SvrCamera initialization, SvrEye components are allocated for the left and right eyes. Unity render textures are created and assigned to each unity camera. The eye target textures are passed to the svr sdk for display processing and timewarp reprojection.
2. overlayWorld.unity demonstrates the use of stereo overlay layers in addition to the stereo eye layers in the previous example. In the SvrCamera hierarchy, 'Overlay Left' and 'Overlay Right' are now active too. Upon SvrCamera initialization, SvrOverlay components are allocated for the left and right overlays. The overlay layers are displayed after the eye layers. Overlay layers are not timewarp reprojected. In this sample, overlays are used to demonstrate the display of objects that are position relative to the svr head. The Reticle in the center of the screen, Hat near the top and Pointer near the bottom are all displayed smoothly without timewarp reprojection induced judder. The Reticle and Pointer sync to the eyes direction.
3. planeWorld.unity demonstrates the use of SvrEye and SvrOverlay layer components used to display textures of imageType StandardTexture in addition to RenderTexture. In this sample, the SnapdragonLogo texture is displayed as a camera-space Reticle in the center of the screen and as a world-space object in the level. The SvrEye and SvrOverlay components are Added to the SvrCamera in the scene and the ImageTexture, ImageTransform and ImageCamera objects are assigned. This sample also demonstrates the use of multiple svr objects in the eye layers. The composite order of the objects within the eye or overlay layer is controlled by LayerDepth with larger values draw after smaller values and the overlay layer(s) are drawn after the eye layer(s). The Reticle is synced to the eyes direction.
4. gridWorld.unity demonstrates the use of foveated rendering with eye render targets greater than 1024 x 1024. The SvrEye layer components are added with ResolutionScaleFactor greater than one assigned. Foveated rendering is enabled with SvrManager.Settings Foveated Gain values greater than zero. Foveated Gain values greater than 1 will cause a reduction in bin resolution toward the extends while Area preserves more of the focal region. In this example, Foveated Gain (4, 4) and Area (2) is specified creating a subtle foveation that saves

approximately 20% of GPU draw. The foveated area Focal Point is synced to the eyes position.

3.2.5 Sample ControllerTest

Setup

- Install the ControllerTest apk
- Install the XimmerseControllerService apk
- Install the BTConfig[1.0.0-Flip] apk from https://github.com/Ximmerse/SDK_Flip/tree/master/Tools

Pairing

1. Launch the XimmerseControllerService app
2. Press and hold the Back button then press and hold the Start button on the controller
3. Select Bind in the XimmerseControllerService app
4. Release both buttons on the controller

ControllerTest - Demonstrates the use and integration of the xImmerse 3DoF controller.

XimmerseControllerService - Service that talks to Ximmerse Controller Daemon and provides the values to the SVR SDK in a format that is required by the SVR SDK.

BTConfig[1.0.0-Flip] – Ximmerse deamon that talks to the bluetooth controller.