**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 4**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC 1/SC 29/WG 4 m** **56727**

**Online – April 2021**

**Title: MIV / MPI on Android**

**Authors: J. Fleureau, F. Thudor, G. Briand, T. Tapie, R. Gendrot, R. Doré (Interdigital)**

**Abstract**

In this informative contribution, we present some early experiments dedicated to the deployment of a full MIV decoding experience on an Android device.

1. **Introduction**

The success of any standard is mainly measured by how much it is adopted and used by the consumer. It will be also the case for MIV and it is therefore interesting to see how such a specification may be implemented on an important class of devices: consumer tablets and smartphones. In this contribution, we propose to present an implementation of a MIV / MPI decoder and renderer on a regular Android tablet as well as some preliminary observations that may be helpful to tune deployment guidelines in a more concrete manner.

1. **Context**
   1. **Android Device**

To run our experiment on such class of devices, a **Google Pixel C tablet** has been chosen. It is not a last generation device, but it was a quite high-end tablet at the time it was released (2016). It especially embeds a NVIDIA Tegra GPU with an an advanced support of OpenGL ES 3.2 as well as dedicated hardware components to decode HEVC bitstreams.

Figure : Google Pixel C tablet

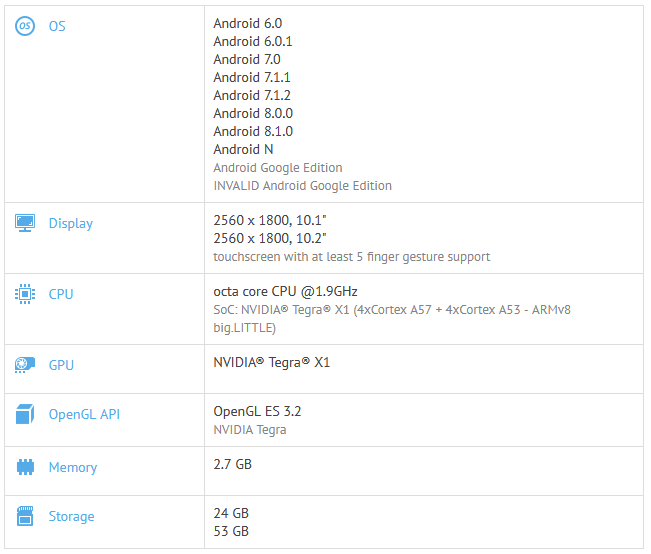


Figure : Google Pixel C features summary

However, as it is the case for many devices on the market, **10 bits HEVC bitstreams decoding is not supported** so far and therefore hardware decoding of 10 bits video stream is not possible.

* 1. **MPI Profile**

In the context of this experimentation, a restricted geometry profile (a.k.a. MPI profile) was chosen for the volumetric video coding. The main reasons of this choice are multiple:

* First, as it has been shown in previous experiments, an MPI-based synthesizer is **computationally cheap** and may well fit such class of devices.
* Second, as it was also mentioned in previous meetings, an MPI-based profile is quite well suited to overcome an **8-bit only video decoding** constraint as it only relies on texture and transparency atlases which naturally tolerate such low bit depths (it is not always the case of a geometry atlas).
* Third, an MPI-based encoding tends to show better **visual robustness at high QPs** because of a lesser sensitivity to coding of the transparency video bitstream. When targeting low bit rates, it is therefore another reason to prefer such a profile.

Furthermore, to limit the complexity at the decoding side and especially avoid having too much hardware decoding instances to manage in parallel, **a single atlas** configuration was preferred. Some more details will be provided further on that point.

* 1. **Test Content**

The **MpiFan content** was chosen to run our tests as it is the only official MPI content provided so far and even if it is a CGI scene, its complexity (thin objects, significant baseline, significant number of occlusions) is noteworthy. Moreover, and as a reminder, such a content is not properly handled so far by a “regular” MIV encoding procedure.



Figure : MpiFan

1. **Technical Details**

Implementing a real-time MIV decoder on a tablet / smartphone is challenging. Even with a single atlas, it requires to decode at least 2 significantly heavy video bitstreams (in addition to the MIV metadata) and then perform a view synthesis task.

To optimally achieve such an operation, video decoding as well as view synthesis must be **hardware accelerated**. Moreover, besides to be accelerated, these 2 critical operations must be closely related to avoid useless copy or GPU to CPU memory transfers.

The proposed solution has been developed in line with these preliminary comments under the shape of a C++ shared module with a dedicated exported interface allowing to drive the different embedded components from an external framework.

* 1. **MIV bitstream demuxing**

The first key component of the whole module is a demuxing operator which splits a regular MIV bitstream into its metadata-only component and the associated texture and transparency video sub-bitstreams. To perform this first task the test model (TMIV) was compiled (and slightly patched) for Android and used as an external library to perform this first parsing.

* 1. **HEVC Video decoding**

The demuxed raw video buffers are then provided to video decoder operators which should be instantiated for each video stream to decode. These decoding operators are the most demanding components in terms of computational resources and battery. On Android, a common API named MediaCodec allows for controlling the hardware decoding. It may be accessible either from the Java SDK or, partially, from the C++ NDK. The FFmpeg API also provides another wrapped interface to get access to the MediaCodec API which has been used in the context of this contribution.

Two different ways to make use of the MediaCodec hardware accelerated API may be considered with significant differences in terms of performances.

A first one that we can qualify of “partially accelerated” effectively performs a hardware decoding in a dedicated fast memory but then recover the decoded video buffer in the main CPU memory. This last step is computationally expansive and induces some complexity overhead. However, such a first way to do is easier to deploy in terms of coding effort and may be sufficient for some use cases (cf. V-PCC for instance).

A second way to do that we name “**fully accelerated**” in the following makes possible to keep the decoded video buffer in a fast / GPU memory which can then be retrieved with zero-copy as an **opaque OpenGL ES texture** to be used for a possible rendering task. This second approach is significantly faster (more details will be given in the following) but is definitively more complex to setup. It especially requires some Java calls which must be done by the means of a JNI (Java Native Interface) interoperability mechanism from a C++ code. Moreover, the opaque decoded texture may be rendered to be used or backup which requires dedicated graphics resources per decoder. Such resources must be allocated per decoder and should be globally synchronized which becomes **critical when too many atlases must be decoded in parallel**. Despite this overhead in terms of coding effort, the performances are definitively better and makes possible to decode MIV atlases with a sufficient resolution to achieve an acceptable visual quality.

* 1. **View synthesis**

Once decoded, the video sub-bitstreams are then provided (together with the decoded MIV metadata) to the synthesis stage. On Android, OpenGL ES is the preferred graphics accelerated framework to perform rendering tasks. Most devices now support **OpenGL ES 3.2** which offers functionalities very comparable to the one delivered by a modern desktop OpenGL.

The synthesis of an MPI-based coded atlas may be done very efficiently in a 1-pass procedure which basically blends the different layers of the MPI from the closest to the furthest. This **blending operation** is advantageously done by the mean of dedicated hardware capabilities such as automatic saturation on alpha (cf. GL\_ALPHA\_SATURATE) and instanced rendering.

A multi-scale **Laplacian inpainting** operator is also integrated as a post process which is also optimized by the mean of dedicated mipmapping operations.

* 1. **Unity framework**

The developed MIV decoding & synthesis shared module can be easily interfaced with any UI / Rendering framework. In the context of this experiment, Unity was chosen as the applicative framework and the developed C++ module was used as an external native plugin. Making use of Unity not only offers the possibility to get access to a very popular game engine but also to quite seamlessly and directly deploy on various targets such as Android devices including tablets / smartphones but also headsets (cf. Oculus Quest).

* 1. **Architecture overview**

The following figure provides an overview of the full operative framework.

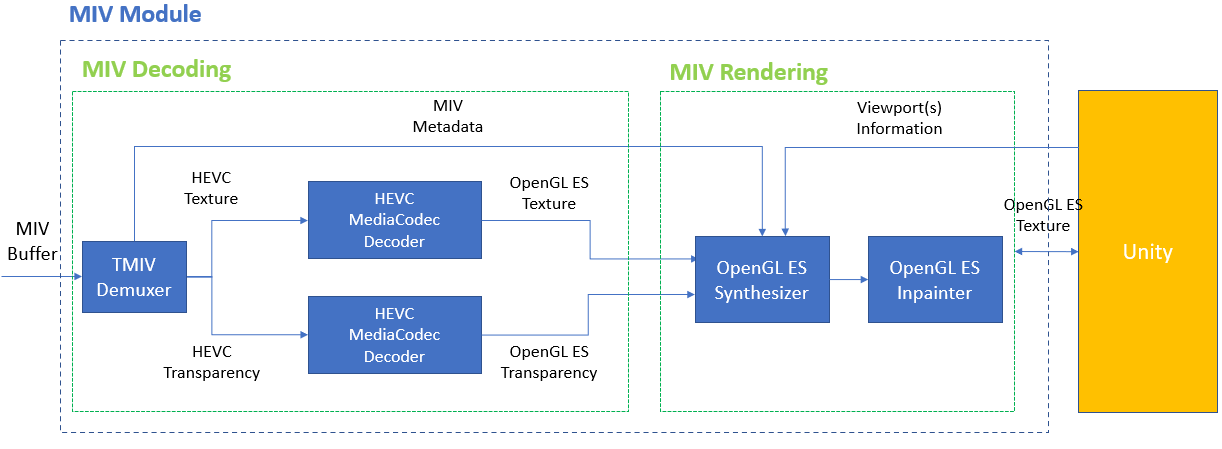


Figure : Architecture Overview

1. **Experimental results**

More than a simple proof of concept to show how possible it is to deploy a MIV experience on an Android device, the development of the proposed decoder was also a good opportunity to investigate a bit more the operating ranges in terms of bitrates, pixel rates, etc. that may be acceptable on such a class of devices.

* 1. **Software vs Hardware decoding**

In this first section we propose some quantitative tests we have initially made to measure the capabilities of an Android video decoding unit when it is setup in: 1) pure software decoding, 2) “partially accelerated” decoding and 3) “fully accelerated decoding”. To do so, 2 identical video bitstreams were decoded simultaneously for different pixel rates (and therefore bitrates) making use of the 3 decoding modalities beforementioned. Figure 5 provides the obtained results. The Y-axis show how many times faster than the real-time the decoder was. It is 1 if the decoding is real-time (or faster) and less than one it is slower than the real time. The X-axis provides the pixel rates of the (duplicated) video to decode.

Figure : Software vs Hardware decoding

A simple software decoding failed to decode real-time as soon as the duplicated video pixel rate is equivalent to a FHD (1920x1080) image. The “partially accelerated” mode collapsed later but significantly earlier than a “fully accelerated” decoder which success to decode real-time for a pixel rate of the duplicated video equivalent to a UHD (3840x2160) image.

This first test therefore clearly demonstrates the interest of performing a fully accelerated decoding to achieve reasonable atlas sizes at the decoding stage.

* 1. **Sensitivity to bitrate / pixel rate**

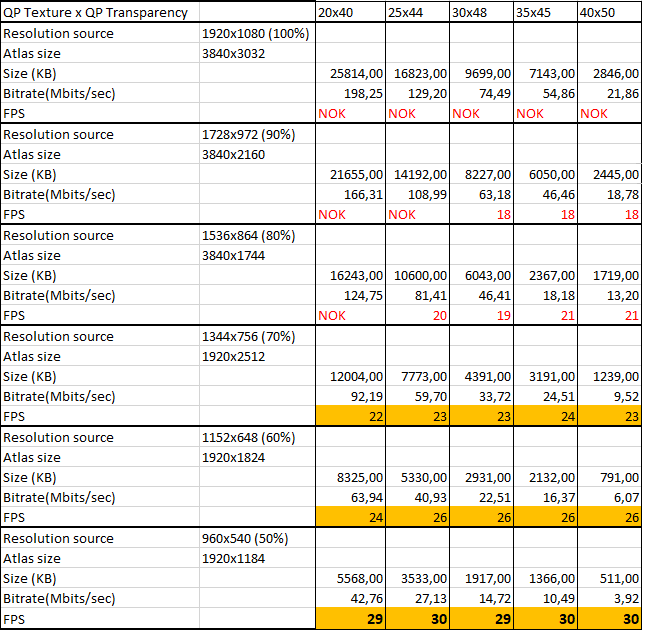
This second test aims at evaluating the performances of the full decoding / synthesis chain for various encoding configurations. More precisely, several MPIs have been generated from several downscaled version of Fan (O). These MPIs have then been encoded with the test model with (manually tuned) optimal atlas sizes for each scaled version. The associated video bitstreams have been in turn encoded at various bitrates by modifying the associated QPs. A set of bitstreams with different pixel rates and bitrates have been thus created and the performances (FPS) of the proposed MIV player have been measured on each of them. The following table sums up the obtained results.

Figure : Bitrate and pixel rate vs FPS

Such a table makes possible to determine which pixel rates and bitrates are accessible at a reasonable framerate on the device we used. Limitations may be achieved independently in pixel rate or in bitrates: a certain bitrate may be “decodable” at a certain resolution but not at another one (typically higher). It tends to show that a compromise has to be find in terms of pixel rate and bitrate to make the decoder properly run on such kind of devices.

As expected, the playing speed seems to decrease when the pixel rates and / or the bitrates increase. As soon as the pixel rate exceeds 10Mpixel per video stream, the player failed to run. However, it is capable to decode UHD atlases at 18FPS up to 60Mbps which is a quite impressive performance for such a device.

1. **Demonstration**

A video has been uploaded on the MPEG server to show a live capture of the proposed demonstrator. The first 32 frames of MpiFan have been encoded at 16Mbps making use of one UHD atlas. Some basic translations are interactively done by touching the left and right parts of the screen and demonstrate the interactive viewport update at 18~20FPS.



1. **Conclusions & Recommendations**

In this contribution, we have demonstrated how possible it is to deploy the MIV standard on quite low-end devices such as smartphones. We have shown that on a pretty old device, real-time decoding is close to be achieved with UHD atlases and bitstreams encoded at 60Mps.

Repeating the same kind of experiences with more modern devices would for sure improve the observed performances and prove that MIV is not reserved to high-end desktop computers only.

However, such a contribution may also serve for the coming phases of the standard dedicated to the definition of profiles. Target pixel rates as well as bitrates should be carefully defined to avoid excluding a large set of devices from a possible MIV deployment.

The number of simultaneous instantiated decoders should also be considered carefully because, when fully accelerated decoding is concerned, graphics resources are engaged per decoding instances and may become a critical point.

10-bits HEVC video decoding, even if it is possible on some modern devices, is not fully common so far on smartphone / tablets, and geometry coding on such kind of devices should be worth of interest.

From a more general point of view, we all know that relying on the increase of the computing capabilities to promote a standard is very dangerous today: environmental considerations and energy saving will be crucial concerns of tomorrow and we should guarantee that MIV is in line with such concerns.