**ISO/IEC JTC 1/SC 29/WG 11  
Coding of moving pictures and audio  
Convenorship: UNI (Italy)**

**Document type: Approved WG 11 document**

**Title: Requirements for Immersive Media Access and Delivery**

**Status: Approved**

**Date of document: 2019-07-12**

**Source: Requirements**

**Expected action:**

**No. of pages: 20**

**Email of convenor: leonardo@chiariglione.org**

**Committee URL: http://mpeg.chiariglione.org**

**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 11**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC 1/SC 29/WG 11 N18654**

**Gothenburg, SE – July 2019**

**Source:**

**Requirements for Immersive Media Access and Delivery**

1. **Introduction**

The present document captures thoughts and first draft requirements on the adaptive delivery and access for immersive media. The content of the document is the result of discussions in the System Subgroup during MPEG 123 and MPEG124 triggered by the input contribution m43753, "Streaming-First design for the MPEG-I project" [1] .

New, immersive services call for new ways of organizing, accessing, delivering and consuming media data. As a result, MPEG should assess what the implications are for its coding, encapsulation formats and content declaration technologies.

This document seeks to communicate the background and requirements, and refrains from phrasing solutions. The title of the activity is a working title, subject to change.

* 1. ***Background***

MPEG-encoded media historically retrieved from storage media (e.g. CD-I, DVD) or delivered using broadcast (e.g. ARIB, ATSC, DVB). Nowadays streaming has become the dominant force in the video industry; video streaming services drive the technical innovation in the media industry. MPEG Technologies (ISO BMFF, DASH) find massive adoptionfor unicast for live and on-demand services augmenting, complementing and replacing broadcast (ATSC, DVB, HbbTV, SCTE). Immersive experiences (VR/360, 3DoF+, etc.) favor unicast-based technologies such as DASH to tailor the data stream to the exact needs of the consuming user in real time.

* 1. ***New Applications***

The following are example of application bringing new challenges for streaming and accessing MPEG coded data:

* **Tiled 360 videos** in very high resolution
* Large **Point** **Clouds** that can be navigated in 6 DoF
* **Light fields** with lots and lots of small tiles
* A complicated **Scene** **Graph** with many objects to traverse
* Audio objects can be audible, or beyond the “**audio** **horizon**” in an immersive experience
* All likely retrieved from some sort of **cloud infrastructure**
* All of these can be available in **multiple quality/bitrate** variations
* All of those need to be **decoded and decrypted** with constrained devices at the receiver side.
  1. ***Trends***

Significant parts of the media data are **unique** **to the receiver**. Delivery shifts from sender-driven to **receiver-driven**. In addition, application requirements change dynamically in real-time, which makes **latency and a fast random access to the media** crucial aspects. However, at the same time, not all the components of an application have the same requirement in terms of end-to-end latency. In general, the amount of media elements contributing to a service is increasing and the user/application selects media according to user interaction and personalization in a dynamic fashion, and *therefore* the media data requires more **fine grain access**. Since mobile and wearable devices are core of the consumption of immersive media, the usage of **general purpose receiving platforms** (decoders, hardware decryptions, protocol stacks, renderers) should be leveraged to offer power and energy-efficient consumption.

* 1. ***Technical Challenges***

Since the representation of the immersive experience constitutes a large amount of data (several TBs expected), it is desirable to allow client to flexibly retrieve **parts** of a large body of media data from a **cloud** resource to create a coherent user experience under **constrained resources**, where:

* **constraints** exist like bandwidth, access latency, decode resources (and where these can fluctuate dynamically)
* the client in charge of making **trade-offs** given such constraints
* fast **response** **times** and **efficiency** are crucial for the QoE
* inherently, data is accessed and retrieved in multiple **parallel** **streams**
* this data may need to be **protected** and/or encrypted
* this data may need to be **cached** close to the user for the best experience
* the data is stored in the **cloud** in a distributed manner
  1. ***Media Access and Delivery Dimensions***

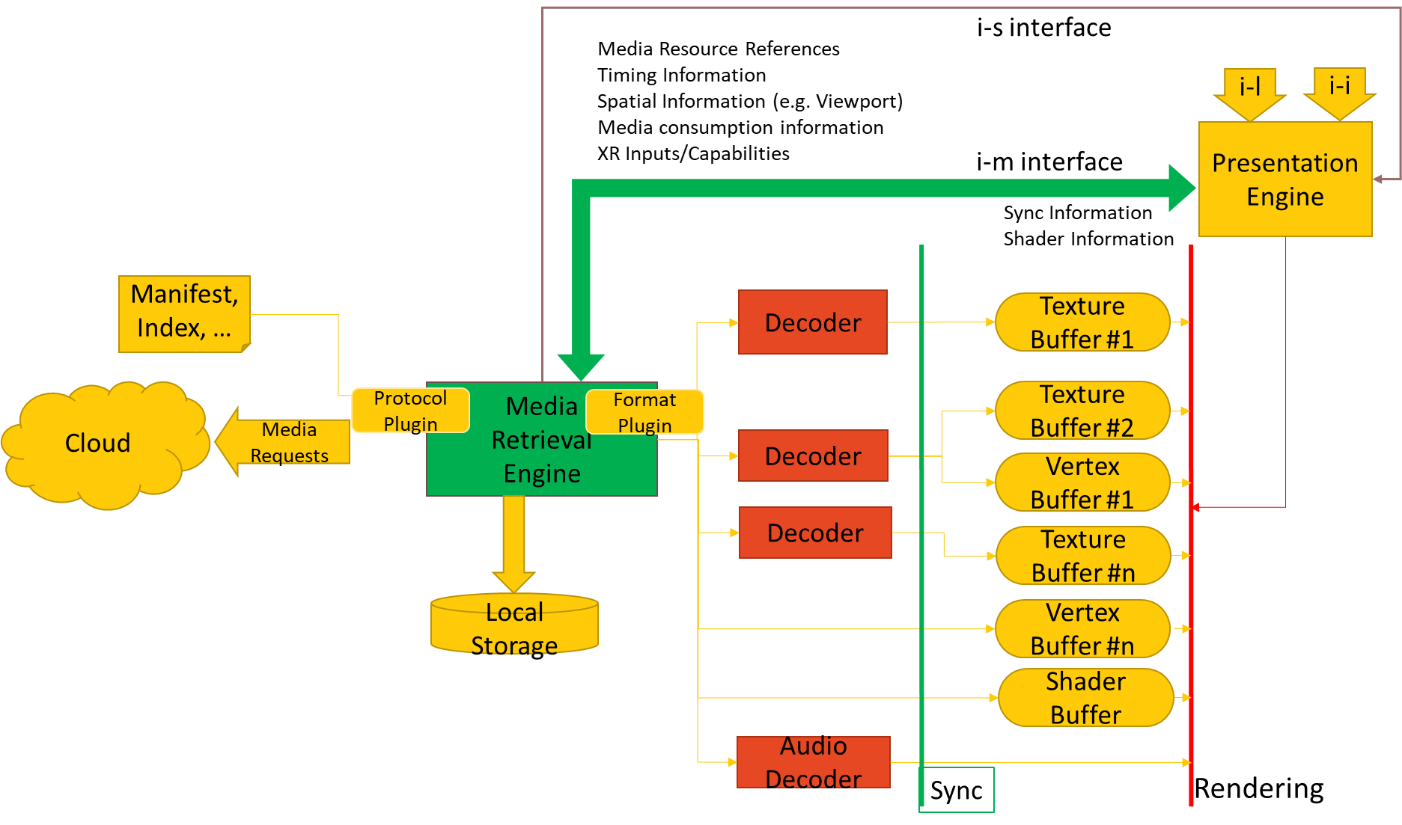
Traditionally, data has been organized to allow temporal access. This dimension will remain but will merely be one of the dimensions:

* **Temporal** random access – “as usual”
  + Different timelines
  + Addition of non-timed media
* **Spatial** random access – retrieving only the relevant parts of the media
  + Depending on user orientation
* **Quality** access – retrieving the suitable quality
  + Making quality/bitrate trade-offs in switching between quality levels
  + Depending on what is visible/audible
  + Depending on retrieval/device and resource constraints, including bandwidth, latency, decoder capability, things like video and audio reproduction capabilities (e.g. screen resolution and color space; speaker config)
* **Object** access – which objects to retrieve and possibly which parts of the objects
  + Descriptions, Nodes, etc.
  + Decoding capabilities, user preferences, etc.
  1. ***Design Goals and Working Assumptions***

To face the challenges of delivering immersive media, the media data stored on a cloud resource should offer a flexible, fast, timely and efficient **random access** where the media dimensions include: Spatial, Temporal, Quality, Bitrate, Objects. In conjunction, the **coding** and **encapsulation** **formats** of immersive media should by design factor in these dimensions such that the interface between the delivery and the coded data itself is lightweight and efficient. From an architecture point of view, there is a need to define a receiver model combining user interactions, decryption, decoding, and rendering along with a retrieval model that leverages these multiple random-access dimensions. These two models would help to better understand how a client could **dynamically** and **efficiently** **tailor** the experience for the user. Lastly, the conformance points between the cloud resource and the decoders to consume such experiences should be specified to ensure interoperability in the ecosystem.

* 1. ***Architectural View***

Figure 1 below depicts an architectural view of an immersive media system.



*Figure 1 - Architectural View of an Immersive Media Client (from [2] )*

* 1. ***Scope of the Work***
     1. **Overview**

This section lists the MPEG work areas that should take requirements for immersive delivery and access into account. We see this streaming-driven delivery philosophy affecting:

1. **Coded representation** of media
2. **Storage, Delivery and Access** of media
   * encapsulation in files
   * encapsulation “in flight”, i.e. during delivery
   * Physical organization of data
   * Access to physical data through indices and Manifests
3. **Data Model Declaration**:
   * Logical organization of data
   * media content (what is available)
   * Differentiation of objects, relationships, metadata, etc.
   * representations of those media (in what forms is it available, and how complex these are to consume; see under 4. below)
4. **Decoder** **Model**
   * for predictable processing (this is for discussion, but the intention is that a player knows the accumulated complexity of the objects it is retrieving, and this would need to be supported by appropriate metadata in the **Data Model**)
     1. **Coded Representations**

Immersive experiences give users the ability to move around and to explore the world around them. As a result, traditional coded video and audio representations have evolved by considering the 3-dimensional aspect immersive scenes. In addition, new types of media representation such as point clouds and object-based audio have emerged to better represent 3D objects in immersive media. The coded representation of these media should be such that it allows partial access with low latency access in all the dimensions mentioned in 1.5.

The following technologies of MPEG-I **coding** formats are relevant in this context:

* Visual
  + 2D video
  + 360 videos
  + 3DoF+ videos
  + Point Clouds
  + Light Field
  + Multi-object video
  + Multi-stream video (today tiles and RWP)
  + Any combinations of those
* Audio
  + Object-based 3D audio
  + HOA-based audio
  + 6DOF audio
    1. **Encapsulation for Storage**

OMAF specifies how to use the ISOBMFF standard to encapsulate MPEG-I coded representations for VR360 distribution.

* + 1. **Encapsulation for Delivery**

Current encapsulation formats for the delivery of MPEG-I media include ISOBMFF and MMT. OMAF specifies how to use these formats in 360 VR, and discussions are already underway on the use of ISOBMFF for point clouds. It is considered beneficial to develop requirements for the storage of point clouds before finally deciding on the exact physical and logical data structures for the storage, encapsulation and delivery of point cloud data.

* + 1. **Network Processing**

There is no doubt that network processing of media will play a significant role in the delivery of immersive media – it already does today for VR360. Tasks such as fusion of media (like stitching) or encoding fragmented media (tiling) are already performed in the cloud. For complicated immersive scenes, network elements can pre-process media before it gets delivered to a resource-constrained client. An example is a complicated 3D graphics element that gets collapsed to a 2D video before delivery. This implies that networked media processors:

* will ingest fragmented media
* will create fragmented media from “unfragmented” sources
* will publish such fragmented media
* will collapse complex media scenes to simpler representations in real-time
* and in doing so, they might create “unfragmented” representations for individual users
  + 1. **Content Declaration**

The current examples of content declaration formats in MPEG include MPEG DASH which defines the Media Presentation Description (MPD). OMAF leverages the MPD to announce and describe the availability of omnidirectional videos. MMT is also used to deliver OMAF content.

A (Hybrid) Scene Graph, as considered in the MPEG-I Scene Description activity, can also be considered as a declaration of immersive content.

1. **Usage Scenarios**
   1. ***Introduction***

The group has decided to study the decoder interface first, and the only usage scenario currently in this document focuses on this decoder interface.

* 1. ***Video Decoder Interfaces***
     1. **Overview**

Two logical interfaces are of interest in the discussion. The first one of interest, namely the interface accessing a video decoder, is placed between the media access engine and the decoding resources, as shown in Figure 1. Current decoders, even when they offer the use of tiles, constrain the use of the decoding resources – probably more than strictly necessary. This forces an immersive application to funnel all the compressed video data into one single bitstream, as currently defined in HEVC for instance. The second interface is the decoder output which retrieves the uncompressed video frames in the output frame buffer of the decoder. By adding metadata such as region-wise packing, the patched media objects can be identified and departitioned after the decoder output into individual objects (texture areas). Immersive applications would greatly benefit from moving away from the current single frame, single bitstream interface between the decoder and the media retrieval engine towards an object-based interface, and from lifting the one-to-one mapping between a frame and a display (wind the associated concept of a frame concept with a certain width and height of homogenous properties.

*Note: These “video objects” resemble the notion of visual objects in MPEG-4, where that standard envisioned a Level definition expressed in macroblocks/second. They are not meant to denote arbitrarily shaped video objects (although with point clouds, they may actually be just that).*

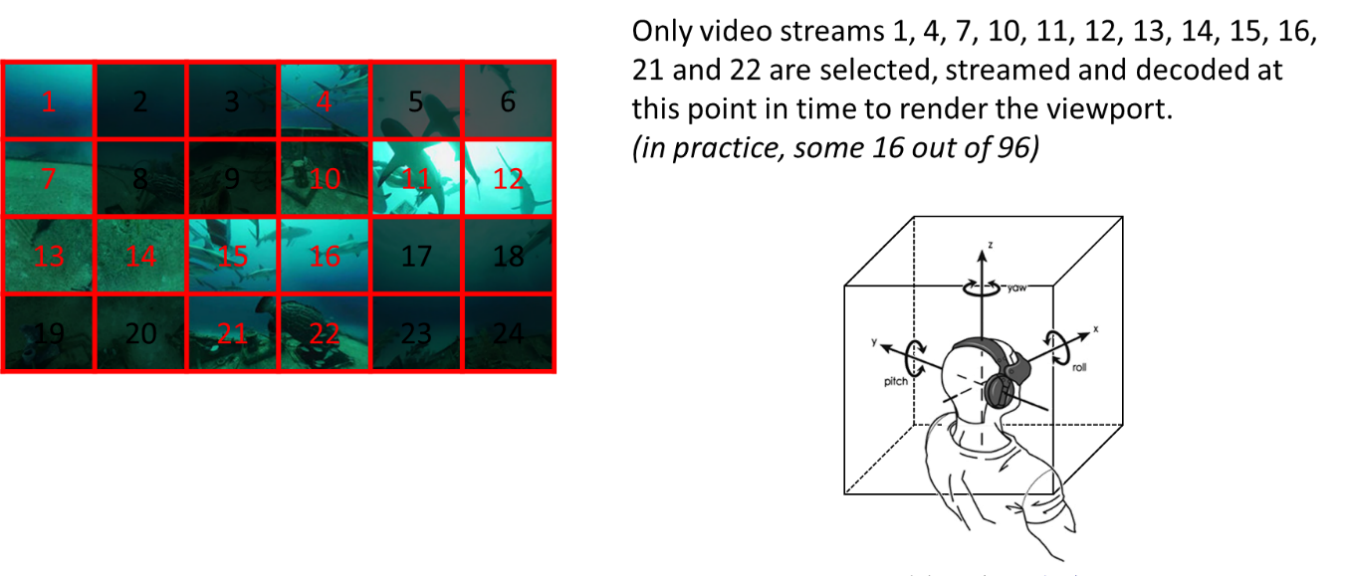
* + 1. **Usage Scenario #1: Viewport-optimized 360 video delivery**

In 360 video streaming, it is advantageous to optimize the bandwidth usage to the portion of the 360 video being watched by the user.



*Figure 2 - Viewport-optimized 360 streaming*

Figure 2 shows a source cubemap 360 video of 8K or more of resolution. Conceptually, the idea of viewport-optimized streaming is to generate independent video stream corresponding to small portions of the original video. For instance, Figure 3 shows a 2x2 tiling of the 8K source video which generate 24 independent video streams. From that set of video streams, the application would only select a subset of these high quality streaming order to render the current viewport in the highest quality possible. In addition, lower quality streams may also be selected to cover regions of the 360 video that are outside of the current viewport. This way network latency can be mitigated.



*Figure 3 - Video selection based on viewport position*

* + 1. **Usage Scenario #2: Synchronization of V-PCC bitstreams**

As of MPEG #124 [4] , V-PCC defines the input point cloud frame to be processed in a following manner.

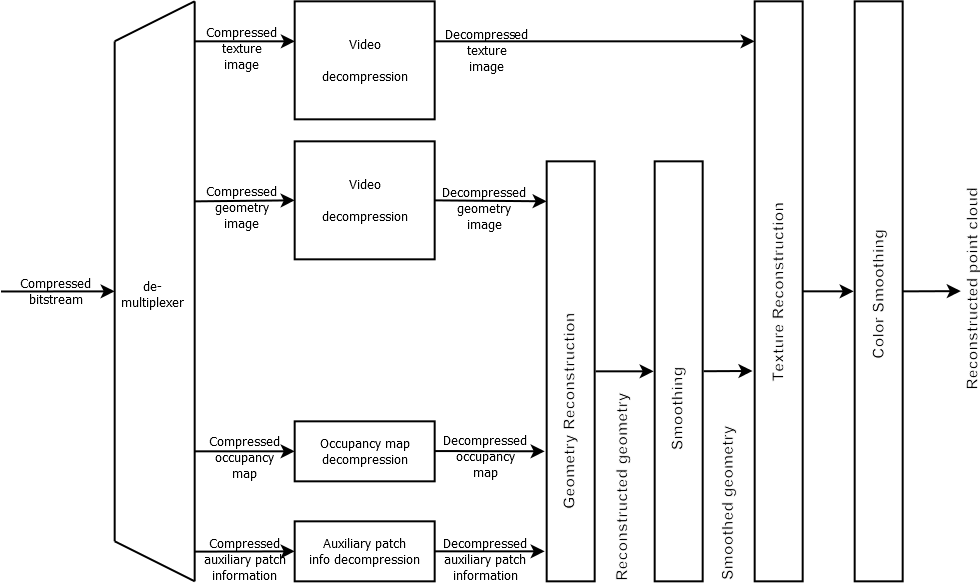
First the volumetric 3d data has to be represented as a set of 3d projections in different components. At the separation stage image is decomposed into far and near components for geometry and corresponding attributes components, in addition an occupancy map 2d image is created to indicate parts of an image that shall be used. The 2d projection is composed of independent patches based on geometry characteristics of the input point cloud frame.

Patch generation method, patch packing strategies and padding methods are out of scope of the standard and current implementations shall be described in best practices section of this document.

After the patches has been generated and 2d frames for video encoding were created the occupancy map, geometry information and the auxiliary information may be compressed. In order to compress the frame with texture information smoothing process is required based on reconstructed geometry and auxiliary patch information.

At the end of the process the separate bit streams are multiplexed into the output compressed binary file.

The block structure in Figure 4 shows the decoding process.



*Figure 4 - V-PCC TMC2 decoding structure*

In typical 2D video player applications, a single video stream is decoded and displayed. On Android, there are some Khronos graphics extensions that allow to stream the video to a texture. Video texture target is provided for the video decoder and latest video frame can be rendered. Player application can render at 60 fps and there is not much need for the application to know if a frame was skipped and displayed two times in a row.

In VR stereo 360 video players there can be two video tracks, one track for each left and right views. In this case, two video decoder instances decode two tracks and sync the frames between the tracks. However, problem exists when frames are skipped as the other eye could have already non-skipped frame data. In practice this is not a problem as eyes adapt to the situation especially if this happens very rarely.

* + 1. **Usage Scenario #3: VR application combining 360 videos and point cloud objects**

VR applications combine different visual elements to provide a rich and interactive experience. For instance, point clouds objects are usually placed inside a virtual environment. In the context of MPEG-I, the virtual environment maybe a 360 video and the object a video-coded point cloud as shown in Figure 5.

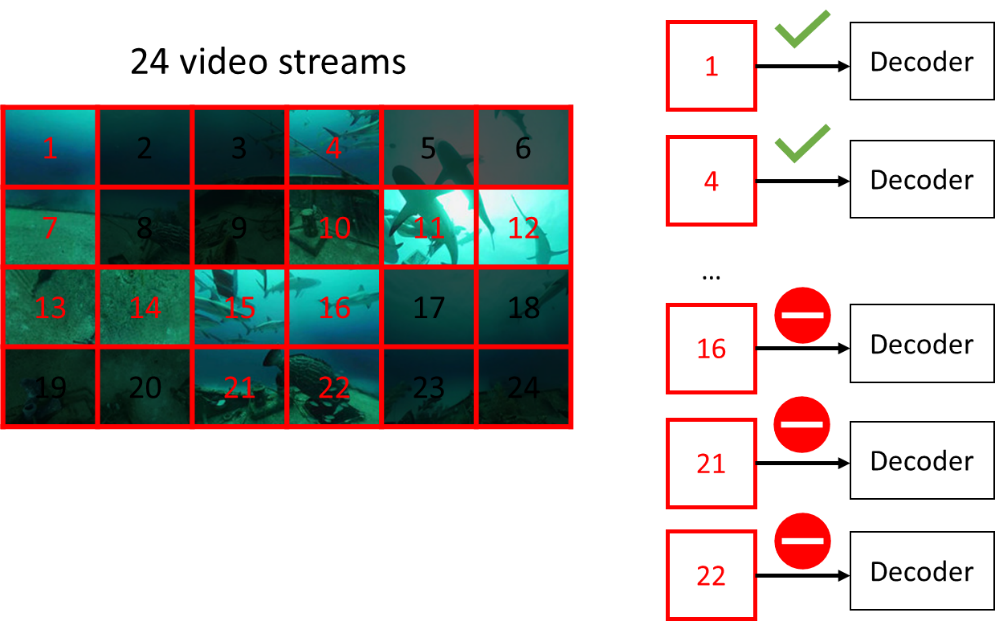
*Figure 5 - Example of a 360 video and a point cloud rendered together*

Relying on a single decoder has obvious advantages as that is what current (mobile) devices usually implement; reason why OMAF v1 goes to so much trouble to make viewport-dependent operation work with a single decoder. But the interface that single decoder could be defined much more flexibly. It should support the simultaneous decoding of these different video elements while they could have been produced separately in time and/or by different encoders, and that may have different video characteristics, framerate, bit depth, etc.

There are many other aspects to make this use case work, such as signaling position of the PC object in the 360 videos, but these do not relate to the video decoding interface and are left for further study.

1. **Identified Issues from Usage Scenarios**
   1. ***Problem #1: Limited Number of Parallel Decoders on Current Platforms***

In any of the listed usage scenarios, the application is expected to consume a certain number of independent video streams simultaneously for rendering the experience. The theoretical number of concurrent decoding instances typically exceeds the capacity offered by current decoding platform. It can be observed that typical end-user devices on the market in 2019 support up to 3 to 4 parallel decoding instances which falls by far short from the needs of these application. For instance, in the 360 video case presented in this document, the application would need up to 12 parallel decoders as shown in Figure 6 which is currently not supported by current end-user devices.



*Figure 6 - Lack of decoding resources for 360 video streaming*

* 1. ***Problem #2: Lack of Frame Accurate Time-locking Prior to GPU Buffer***

All the immersive applications listed in this document change the traditional media pipeline. Post decoding, the decoded frames are no longer pushed down to the display with a one-to-one mapping between output of the decoder and the input of the render. In between, the GPU collects the different decoded frames, process them and generate textures for the renderer.

This section gives details the issues of tight synchronization of different videos objects.

* 1. ***Problem #3: Dynamically using available decoding resources***

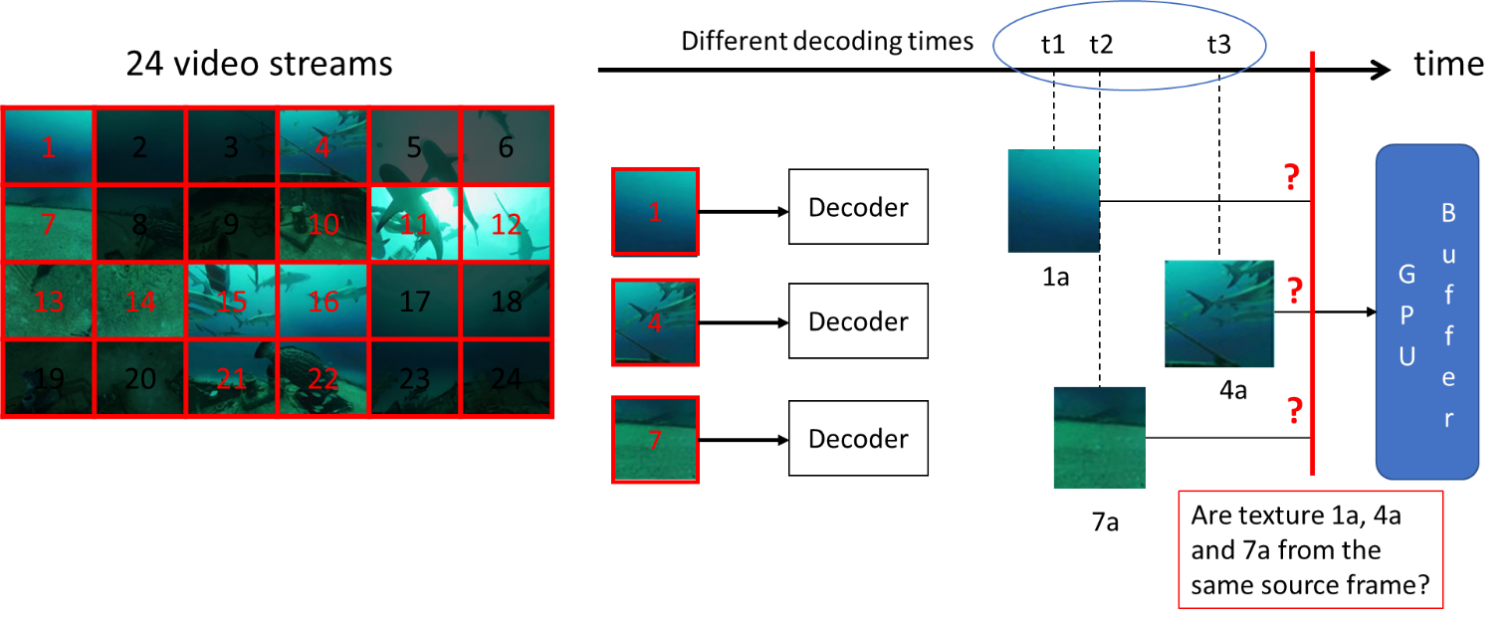
If parallel decoders are used, the decoders are statically and independently allocated for decoding resources, typically following the profile and level constraints of the bitstream. However, such a static allocation does not allow to change quality and number of objects during presentation, for example when the pose of the user change or when network resources change. It may also be the case that the decoding resources change, depending on the current power or thermal state of the device. It would be beneficial that available decoding resources can be shared dynamically across different decoding instances. Decoding complexity includes at the minimum:

* Bitrates
* Luma and Chroma samples per second
* Coded and Decoded Picture Buffers

It should be ensured that the aggregated resources are not exceeding the decoding resources. Additional dynamic configuration of decoding resources, for example by the usage of an API would be further beneficial.

* 1. ***Problem #4: Viewport-dependent 360 Video Streaming***

In viewport-optimized 360 streaming, the different parts of the source projected video need to be time-locked in the GPU buffer such that the texture in the GPU for only rendering visual samples that originally where in the same frame of the source content. In other words, there is the need of a frame-accurate synchronization of the several output of the coder prior to the GPU processing.



*Figure 7 - Frame time-locking for GPU texture buffer*

Figure 7 illustrates the challenges for time-locking the decoded pictures before the GPU processing.

* 1. ***Problem #5 Video-based Point Cloud Compression (V-PCC)***

This Section describes the challenges that were encountered in implementing MPEG’s Video-based Point Cloud Compression (V-PCC) on mobile operating systems.

* + 1. **Android**

In general, there is the common challenge of synching multiple video tracks with the current Android APIs. Android documentation provides the following information:

*SurfaceTexture (*[*https://developer.android.com/reference/android/graphics/SurfaceTexture*](https://developer.android.com/reference/android/graphics/SurfaceTexture)*)*

*"When updateTexImage() is called, the contents of the texture object specified when the SurfaceTexture was created are updated to contain the most recent image from the image stream.* ***This may cause some frames of the stream to be skipped.****"*

In V-PCC, frame skipping and syncing causes more difficulties than in regular video, as 5 video frames (2x texture, 2x depth, occupancy) must be synced. If any of the 5 frames were skipped, the application cannot display the point cloud at all, as all 5 decoded frames are needed for its reconstruction. Therefore, we need to store either copies of the reconstructed point clouds or the video frames. Both options would require extra copying of the data and therefore lower performance would be expected.

For this reason, the current V-PCC implementation only supports single layer decoding where 3 video streams must be synced (1x texture, 1x depth, occupancy). Android video texture extensions can be found here:

<https://www.khronos.org/registry/OpenGL/extensions/OES/OES_EGL_image_external_essl3.txt>

<https://www.khronos.org/registry/OpenGL/extensions/EXT/EXT_YUV_target.txt>

The image external extension is widely supported. However, it uses automatic YUV to RGB conversion and it’s not well specified how this works. Currently in V-PCC depth and occupancy only use luma video channel and therefore would not need any YUV to RGB conversion as we could just directly read the luma channel values. It was noticed that the current TMCv3 writes zero values to chroma channels for depth and geometry. This leads to invalid values as automatic YUV to RGB conversion is done by the extension. We propose that correct (128 instead of 0) chroma values should be used.

Newer Android YUV target extension allows fetching the different YUV components and also supports well defined automatic YUV to RGB conversions. Current implementation uses this extension for the flexibility. It’s not known how well this extension is adopted on multiple Android chipsets.

On Android it’s also possible to decode video frames directly to CPU memory by configuring the video decoding (AMediaCodec\_configure) with ANativeWindow = NULL. With this AMediaCodec\_getOutputBuffer provides YUV planes in NV12 format and these can be uploaded to textures by having Y plane as R8 format an UV planes as RG8 format. The benefit of this synchronous approach is that we can more easily create copies out of the decoded images. However, we discovered that the performance of this approach is well under 15 fps and therefore in practice not usable.

* + 1. **iOS**

The first issue to tackle with the iOS implementation was that ARKit does not support OpenGL ES 3.0. Therefore, it was decided that we only try V-PCC decoding without AR. Enabling AR would need a Metal based graphics rendering pipeline to be used. OpenGL ES APIs are being deprecated on iOS platform.

Video decoding on iOS was more straightforward than on the Android platform. Current video decoding uses VideoToolBox framework and in VTDecompressionSessionCreate with the following flags:

* kCVPixelBufferOpenGLESCompatibilityKey
* kCVPixelBufferOpenGLESTextureCacheCompatibilityKey

There is no need to upload textures as iOS provides texture caching (<https://developer.apple.com/documentation/corevideo/cvopenglestexturecache-q2r>) and video textures can be accessed with CVOpenGLESTextureCacheCreateTextureFromImage. Finally, YUV to RGB conversion is done in the shader code.

1. **Requirements**
   1. **Media Representation**
      1. **Video Decoding Interface (Approved)**

*This Section is considered mature*

1. The decoding interface shall allow multiple independent video objects to be decoded.
   1. The decoding interface shall not require rewriting of the bitstream(s) corresponding to these objects, by the application using the decoding resources.
      1. If a single multiplexed bitstream comprising multiple bitstream(s) of video objects is provided as input to the decoder interface, changing value fields related to the decoding operation in the bitstream(s) shall not be required to multiplex such bitstreams into a single multiplexed bitstream.

*Note: The multiplexed bitstream is not required to contain all video objects that were encoded. The application using the decoding resources selects which video objects are included into the multiplexed bitstream.*

* 1. It shall be possible to maintain absolute, frame-level sync between these decoded objects.
     1. The decoding interface shall provide a possibility to output decoded objects in decoding order (possibly out of presentation order).
     2. The video object bitstreams shall provide sufficient information, for example on the decoding reordering, such that, the decoding interface can be used to output decoded objects in presentation order with frame-level sync across video objects within available decoding resources.

*Note: The required decoding resources for handling multiple independently generated bitstreams for objects may increase if bitstreams have different decoding orders.*

* 1. The maximum number of simultaneously decoded objects shall be a function of the Level definition of the decoder.

*Note: This should allow for a number of objects that is at least an order of magnitude more than the amount of decoders that can typically be instantiated in a device today (e.g. 4).*

* 1. It shall be possible to simultaneously decode objects with different parameters:

1. different object sizes and aspect ratios
2. different framerates

*Note: For example, a 30 fps fallback layer in tiled VR streaming, ad a 90 fps foreground (viewport) layer.*

1. different resolutions (sampling density)

*Note: For example, a mix of 4K, 6K and 8K representations for different parts of the same 8K ERP or cubemap.*

1. different color space
2. different bit depth
   1. The decoding interface shall support decoding objects that have different image parameters, such as color space and bit depth.
   2. The decoding interface shall support maintaining the association of metadata with the decoded object.
      1. It shall be possible to keep objects in absolute (frame-level) sync with their associated metadata.
   3. It shall be possible to control whether decoded objects of the same timestamp are output separately or spatially arranged onto the same output picture. Selected control shall be persistent at sequence level.
   4. The conformance requirements for the decoding interface shall include a well-defined maximum decoding latency for all objects on a per-frame basis.
   5. The decoding interface shall support the dynamic and flexible usage of the decoding resources for different objects.
      1. The decoding interface shall allow to dynamically add and remove objects during a decoding session.
      2. The decoding interface shall allow to support communication between the decoding platform and the application on the currently available decoding resources.
   6. The specification shall enable signalling metadata associated with objects on required decoding resources, that allows scheduling the decoding of these multiple objects.
3. It shall be possible to do all of the above with protected content.
   1. It should be possible to mix protected and non-protected content

*Note: Whether support for different keys is required for these objects is for discussion, and it may not impact the decoding interface.*

1. The decoding interface shall allow efficient handling by graphics processors.

*Note: This might be done by, e.g., allowing decoded objects to be packed into texture buffer(s) as appropriate.*

The following requirement is perhaps not on the decoding interface, but is nevertheless important for the system:

It shall be possible to manipulate decoded objects without having access to the actual pixel information.

*Note: This is to be able to manipulate objects in a secure pipeline. Keeping with the tiling example: this would allow placing protected tiles in the right position on a sphere.*

* + 1. **Audio Decoding Interface**

*To be provided*

* 1. **Media Storage (draft)**

*This section is still being developed*

1. The specification shall support sub-division of immersive media content, e.g., into tiles, and encapsulation of sub-divided media content in MPEG storage formats (such as ISOBMFF tracks), to support:
   1. Parallel encoding and decoding with low cost in terms of bitrate overhead,
   2. Spatial random access to a region without having to decode the entire media bitstream, and
   3. ROI-based and viewport-dependent media processing, including delivery and rendering, for optimal network bandwidth and device resource utilization while maintaining satisfactory consumption experiences.
2. The specification shall support signaling of spatial relationship and sub-division relationship among related sub-divided immersive media content, in order to support tiled and viewport dependent immersive media processing
   1. Storage formats and metadata shall support logical structures to signal the spatial relationship and sub-division relationship among related sub-divided immersive media content in the storage format.
3. The specification shall support signaling of spatial relationship and sub-division relationship among related groups of sub-divided immersive media content, in order to support tiled and viewport dependent immersive media processing
   1. Storage formats and metadata shall support logical structures to signal the spatial relationship and sub-division relationship among related groups of sub-divided immersive media content in the storage format.
4. The specification shall support the use of entry points for accessing immersive media content, in a consistent manner like for accessing conventional, non-immersive media content.
   1. **Media Adaptation Requirements (draft)**

*This section is still under development*

1. It shall be possible for a client application displaying and playing a certain immersive scene to perform adaptation (e.g., to network conditions) across all the media components composing that scene:
   1. It shall be possible to determine the delivery cost of any coded representation of any media component
   2. It shall be possible to determine to compare two delivery costs across the media components
   3. It shall be possible to continuously make cost vs quality tradeoffs between any of these media components

*For example, the tradeoff may be expressed in bandwidth against number of points of a point cloud object.*

* 1. It shall be possible to perform adaptation coherently across all elements in the scene

*For instance, to prevent competing strategies between adaptation engines belonging to individual objects in it*

* 1. It shall be possible to signal that some media elements may not be retrieved if minimum quality cannot be reached.

1. It shall be possible to enable viewport-dependent streaming across all the media components composing the scene:
2. It shall be possible to produce and store a compact representation of the scene, along with related additional information (e.g.: relations, relative importance of objects and scene-level adaptation strategy)

*[Ed note: The requirement should be about what to retrieve in the first place and in which quality in the scene]*

1. It should be possible to reuse adaptation logic and engines in client implementation for all types of objects (as opposed to having multiple adaptation engines that might be coordinated but would still all be implemented individually)
2. It shall be possible for a client application displaying and playing a scene to dynamically manage the available decoding and rendering resources across all the media components composing that scene.
3. It shall be possible for a client application displaying and playing a scene to provide seamless transitions both in space and time of the media experience based on user interaction, such as changing viewports as well as based on changes of network conditions.
   * 1. *Examples are the avoidance of re-initializing decoding and decryption pipelines in case of such changes.*
4. It shall be possible for a client application displaying and playing a scene to manage the timelines of the individual objects of the scene relative to the playback timeline of the scene.

*[Ed Note: The playout start of some objects may be defined just-in-time, in a sense that it is only possible to know the relative timeline (wrt the 'master' playback timeline) of these objects at run-time.*

*Ed. Note: This is very close to a requirement on the Scene Description itself. It needs to be clear what the adaptation requirement is exactly.]*

* 1. **Data Model Declaration (draft)**

*Note: the requirements in this entire Section are an early Draft*

1. The specification shall support declaration metadata to identify individual objects and object fragments for efficient immersive media access and delivery.
   1. Identification for efficient immersive media access and delivery shall support the following types of metadata about the context of an object and object fragments:
      1. Temporal context
         1. the temporal position of the object either in absolute time relative time from any events such as beginning of the scene timeline
         2. indication of non-timed fragment whose temporal position is indefinite or decided by the user interaction
      2. Spatial context
         1. the spatial position of the fragment in 3D space either as absolute position or relative position to any objects in the rendered scene
      3. Quality context
         1. the quality of the object using objective measures
         2. the existence of alternative representations with different quality levels, including the indication of relative quality differences and levels
      4. Capability context
         1. the capabilities required to decode and the render objects
      5. Object context
         1. description about the object the fragment belongs to
      6. Interaction context
         1. Any dependency related to user interaction
   2. The specification shall support the storage and carriage of the declaration metadata such that it can be accessed before accessing any media data.
      1. The specification shall support transport of the declaration data both as a part of the file containing media resources and independent from that media data
      2. The specification shall support declaration data formats that are agnostic to the delivery method.
   3. **Delivery (draft)**
2. Delivery formats shall also support access to media with the same requirements as listed under 4.2 Media Storage. Unlike in stored formats, minimizing overhead *is* important for delivery.
3. It shall be possible to aggregate delivery units coming from different sources.

*[Ed Note need to figure out in which places the aggregation may happen.]*

1. Delivery formats shall support the delivery of fragments as small as a few hundreds of bytes – again while minimizing overhead.
2. Delivery formats shall support random access corresponding to a volume in a 3-dimensional space, a duration in time, a part of a 3D object.
3. **References**
4. ISO/IEC JTC1/SC29/WG11 MPEG2018/m43753, "Streaming-First design for the MPEG‑I project", Emmanuel Thomas, Rob Koenen (TNO), Ali C. Begen (OzU), Jill Boyce (Intel), July 2018, Ljubljana, SI
5. ISO/IEC JTC1/SC29/WG11/N18344, "MPEG-I Architectures", Thomas Stockhammer, Imed Bouazizi, Geneva, CH – March 2019
6. ISO/IEC JTC1/SC29/WG11 MPEG2019/M46074, "On implementing V-PCC standard", Mika Pesonen, Sebastian Schwarz, January 2019, Marrakesh, MA
7. ISO/IEC JTC1/SC29/WG11 MPEG2018/N18017, "V-PCC Codec description", October 2018, Macau, CN

**Annex A**

This annex is a placeholder for requirements that were not yet agreed and for aspects that need to be addressed at a later time.

**A.1 Network-based Based Processing**

A separate activity is underway in MPEG to define interfaces for Network-based Media Processing.

Network-based media processors will need to be able to ingest, produce and publish fragmented media, and they must also support collapsing such media to less-complex representations.

**A.2 Data Model Declaration**

This section provides still high-level requirements that try to capture the technical features needed to enable fragmented access and delivery.

Content declaration format(s) shall provide a description of the content along at least the following axes:

* position in space,
* position in time,
* object shape,
* range of qualities

Content declaration formats shall enable content selection along axes including at least:

* position in space,
* viewing frustum,
* decoding capabilities

Data Model formats shall enable adaptation along axes such as

* position in space,
* viewing frustum,
* available bandwidth,
* dynamic processing capabilities
* representation quality