



Technology under Consideration for ISO/IEC 23090-14

WG3 Scene Description BoG

MDS23201_WG03_N01048

Table of Contents

1. Extensions	1
1.1. MPEG_camera_control	1
1.1.1. General	1
1.1.2. Semantics	1
1.1.3. Processing Model	3
1.1.4. Example	3
1.2. Multi-user interactivity	4
1.2.1. Introduction	4
1.2.2. References	6
1.3. MPEG_material_acoustic	6
1.3.1. General	6
1.3.2. Semantics	6
1.3.3. Processing Model	9
2. ISOBMFF	11
2.1. Improvements for MPEG-I SD random access description	11
2.1.1. General	11
2.1.2. Characteristics of random access points of MPEG-I Scene Description	11
2.1.3. Description and processing of random access points	11
2.1.4. Proposed text improvements	12
2.2. On sample formats for lighting information	13
2.2.1. Introduction	13
2.2.2. Lighting information signalling	14
2.2.3. Proposals	15
3. Codec Support	24
3.1. Dynamic mesh support in scene description	24
3.1.1. Introduction	24
3.1.2. Design	24
3.1.3. Assets and Implementation	24
3.2. Support for multiple atlases for MIV applications (MPEG142)	25
3.2.1. Multiple atlases	25
3.2.2. References	32
3.3. On G-PCC support	33
3.3.1. Consideration on in-GPU processing	33
3.3.2. Proposal	34
3.3.3. Reference	35
3.3.4. Annex. Proposed MPEG extension	35
4. Data Formats	39
4.1. Support of glTF CBOR binary format	39

4.1.1. Problem Statement	39
4.1.2. Benefit of CBOR file/data format:	39
4.1.3. CBOR data size comparison example:	39
4.1.4. Use Cases	39
4.1.5. Potential Solutions	40
4.1.6. Open Issue Discussion	41
5. Interfaces	42
5.1. On DASH Dynamic Bitrate Adaption with Viewpoint Update	42
5.1.1. Problem Statement	42
5.1.2. Use Cases	42
5.1.3. Current Scene Description Support and Gaps	43
5.2. Supporting Multiple Viewers in the Media Access Function	44
5.2.1. General	44
5.2.2. Proposed Updates to MAF API	45
5.3. CoAP API support in MAF	46
5.3.1. General	46
5.3.2. MAF as CoAP Client	46
5.3.3. MAF as HTTP-CoAP Proxy	46
5.4. An Abstract API for Driving External Renderers	47
5.4.1. Render Lock-in API	47
6. MPEG-I Audio in Scene Description	49
6.1. Immersive audio extension	49
6.1.1. Introduction	49
6.1.2. Background	49
6.1.3. MPEG-I immersive audio support	50
6.1.4. References	54
6.2. MPEG-I Audio in Scene Description	54
6.2.1. General	54
6.3. Establishing a Mapping between Audio and MPEG-I Scenes	56
6.3.1. General	56
6.3.2. Extension for Audio Node Mapping	56
7. Reference Software	58
7.1. Thoughts on trimesh playback of AR scenes	58
7.1.1. General	58
7.1.2. AR Sessions recording and format	58
7.1.3. AR Session playback in trimesh	62
8. Interactivity framework	63
8.1. On event-based scene update	63
8.1.1. General	63
8.1.2. A use case for event based updates	64
8.1.3. JSON patch limitations	65

8.1.4. Semantics for event-based update	66
8.2. Physic Support.	67
8.2.1. Introduction	67
8.2.2. Analysis of the physic simulation consistency between game engines with the current parameters	68
8.2.3. Analysis with new physics parameters	69
8.2.4. Proposed changes to SD physic support	73
9. Collected problem statements and industry needs	78
9.1. On the support of real environment data.	78
9.1.1. General	78
9.1.2. Representation of the real environment.	78
9.1.3. Storing a representation of the real environment	79
9.1.4. Examples of framework for real environment handling	80
9.2. Semantic representation.	83
9.2.1. Semantic Expression for 3D contents	83
Appendix A: Disclaimer	85

Chapter 1. Extensions

1.1. MPEG_camera_control

Source: [m56337](#), [m57409](#)

1.1.1. General

The scene description may describe a set of paths through which the camera is allowed to move. The paths may be described as a set of anchor points that are connected through path segments. For enhanced expressiveness of the camera control, each path segment may be enhanced with a bounding volume that allows some freedom in motion along the path. The [Figure 1](#) depicts this behavior.

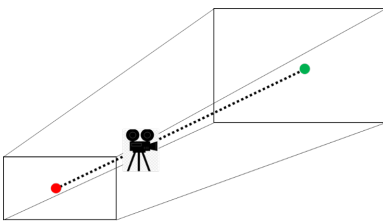


Figure 1. Example of Camera Path Segment with Bounding Volume

Example of Camera Path Segment with Bounding Volume The scene camera, and by consequence the viewer, will be able to move freely within the bounding volume along the path segment. The path segment may be described using more complex geometric forms to allow for finer control of the path.

Furthermore, the camera parameters may be constrained at each point along the path. The parameters are provided for every anchor point and then used together with an interpolation function to calculate the corresponding parameters for every point along the path segment.

In fact, the interpolation function applies to all parameters, including the bounding volume.

The camera control extension is a glTF 2.0 extension that defines camera control for a scene. The camera control extension is identified by “MPEG_camera_control” tag, which shall be included in the extensionsUsed and should be included in the extensionsRequired of the scene.

1.1.2. Semantics

The `MPEG_camera_control` extension shall be defined on `camera` elements. It contains the following properties:



*TODO : auto generate the semantics
schema is needed*

	Type	Description	Required
anchors	number	Number of anchor points in the camera paths.	No
segments	number	<p>The type of the bounding volume for the path segments. Possible types are:</p> <p>* BV_NONE: no bounding volume</p> <p>* BV_CONE: capped cone bounding volume, defined by a circle at each anchor point.</p> <p>* BV_CUBOID: a cuboid bounding volume, defined by size_x, size_y,size_z for each of the 2 faces containing the two anchor points.</p> <p>* BV_SPHEROID: a spherical bounding volume around each point along the path segment. The bounding volume is defined by the radius of the sphere in each dimension, radius_x, radius_y, radius_z.</p> <p>default: BV_NONE</p>	No
boundingVolume	number	<p>Quaternion describing the rotation of the scene in the anchor space. centerPosition and orientation are used as alternatives to transformation.</p> <p>default:false</p>	No

	Type	Description	Required
cameraIntrinsics	boolean	When set to true, indicates that the intrinsic camera parameters are modified at each anchor point. The parameters shall be provided based on the type of camera as defined in [glTF 2.0] as camera.perspective or camera.orthographic.	No
accessor	number	The index of the accessor or timed accessor that provides the camera control information.	No

The camera control information is structured as follows:

- For each anchor point, (x,y,z) coordinates of the anchor points as float numbers
- For each path segment, (i,j) indices of the first and second anchor point of the path segment as an integer
- If boundingVolume is BV_CONE, (r1,r2) radiuses of circle of first anchor point and second anchor point. If boundingVolume is BV_CUBOID, (anchor_idx,size_x,size_y,size_z) for each anchor point of the path segment. If boundingVolume is BV_SPHEROID, (r_x,r_y,r_z) as radius of the spheroid for each anchor point of the path segment.
- If cameraIntrinsics is true, the intrinsic parameter object.

1.1.3. Processing Model

The Presentation Engine shall support the MPEG_camera_control extension. If the scene provides camera control information, the Presentation Engine shall limit the camera movement to the indicated paths, so that the (x,y,z) coordinates of the camera always lie on a path segment or within the bounding volume of a path segment. The Presentation Engine may provide visual, acoustic, and/or haptic feedback to the viewer when they approach the boundary of the bounding volume.

1.1.4. Example



TODO : add example

Input needed

1.2. Multi-user interactivity

Source: [m64014](#)



The group invites for alternative solutions, possibly with also with different design choices, for example where scene description document is not modified, but other solutions are used. Alignment with 3GPP is encourage.

1.2.1. Introduction

We propose to address the shared indication and to delegate the generation of the scene update data to the application server, when the updates must be shared.

For the missing action type, an approach is proposed in the TUC document [6] that need further investigation that may be addressed in a future phase 3.

1.2.1.1. Solution Overview

Each user receives from the application server an initial scene description file containing a description of 3D scene objects and interactivity elements.

An event is defined as the activation of a set of triggers referenced in a behavior object as specified in the MPEG_scene_interactivity extension.

The following description refers to the [Figure 2](#).

When a set of triggers fire at one user's side (**step 2**). The behavior information is then sent to the application server (**step 3**).

The behavior information content and format are out of scope of MPEG-SD, but it may include:

- The index of the behavior in the behaviors array defined in the scene description file. The application server knows the scene and its interactivity features, and it will be able to launch related actions and generate scene updates based on the specified behavior (**step 4**).
- Additional information to perform the update, for instance for a user input trigger, a pose information related to where the gesture has been detected (a 2D position for a touch on a surface, the 3D position of a user's hand...)

The application server uses this information to generate and forward scene updates to the all users (**step 5**): A scene update content and format are out of scope of MPEG-SD, but it may contain:

- a patch (for instance a JSON patch [3] to update glTF scene graph) to be applied to the scene graph, adding, deleting, or modifying some nodes.
- an action to be executed by each user (play/stop a media file or an animation, activate a node). It can be a string describing an action as specified in [2] or the index of an action object in the actions array defined in the scene description file.
- The new state of some objects of the scene graph (activate, enabled, paused...)
- an identifier of the trigger information that caused the update.

The scene updates are then applied by all the users (**step 6**)

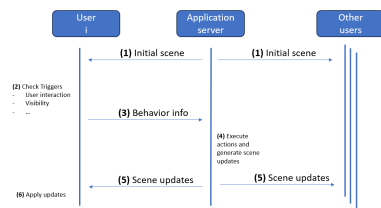


Figure 2. Shared behavior sequence diagram

1.2.1.2. MPEG-SD semantic for the “shared” parameter

We propose to add a new “shared” parameter to a behavior object: We could have added this parameter in a trigger or action object, but it would imply duplication if a same trigger/action is used for shared and non-shared updates.

The table 8.2-9 of the amd2 document should be modified as follow:

Table 1. semantic of behavior

Name	Type	Usage	Default	Description
triggers	array	M		Indices of the triggers in the triggers array considered for this behavior
actions	array	O	[]	Indices of the actions in the actions array considered for this behavior. The action list may be empty when the flag “shared” is set to true.
Shared	Boolean	O	false	Indicate if the behavior is to be process locally only (False) or if it must also impact the other connected users (True).
...				

The following text should be added in the section 8.2.3 of the amd2 document:

When a Presentation Engine parses a behavior object containing a “shared” parameter set to true, it checks the activation of the referenced trigger. When the triggers become active:

- it sends to the application server the behavior information.

When the presentation engine receives a scene update message from the application server, it updates its scene graph accordingly.

When a Presentation Engine parses a behavior object containing a “shared” parameter set to false or without a “shared” parameter, it checks the activation of the referenced trigger. When the triggers become active:

- If the actions list is empty, it raises an error.
- If the actions list is not empty, it executes each action that causes local changes only.

1.2.2. References

[1] Information technology - Coded representation of immersive media - Part14: Scene Description for MPEG media, ISO/IEC DIS 23090-14 :2021 (E)

[2] ISO/IEC JTC 1/SC 29/WG 3 N0797, Text of ISO/IEC 23090-14 CDAM 2: Support for Haptics, Augmented Reality, Avatars, Interactivity and Lighting, March 2023

[3] IETF JSON patch: <https://datatracker.ietf.org/doc/html/rfc6902/>

[4] Technology under Consideration for ISO/IEC 23090-14, May 2023

[5] ISO/IEC JTC 1/SC 29/WG 2 N00230, MPEG-I Phase 2 requirements, July 2022

[6] 3GPP TR26.998 Support of 5G Glass-type AR/MR devices, v18.0.0 (2022-12)

1.3. MPEG_material_acoustic

Source: [m64377](#)

1.3.1. General

The acoustic material extension adds support for acoustic materials to a scene. This extension may be used together with the MPEG_audio_spatial extension, but is not limited to that extension.

When present, the MPEG_material_acoustic extension shall be included as an extension to a material object as defined in ISO/IEC DIS 12113:2021.

For a primitive that is associated with a visual material, the acoustic material extension shall be attached to it.

1.3.2. Semantics

The definition of the MPEG_material_acoustic extension is provided in the following table.

Name	Type	Default	Usage	Description
frequencies	array		O	provides an array of MPEG_material_acoustic.frequency objects as defined in the next table.
accessor	integer		O	As an alternative, the frequency characteristics may be accessible through an accessor, which references a binary representation of the data in a buffer. The binary format of the elements is provided in table 3.

The definition of the MPEG_material_acoustic.frequency is provided in the following table.

Name	Type	Default	Usage	Description
frequency	number		M	The frequency for associated with the following coefficients, with values between 1 and 24000.
specularReflection	number	0.0	O	The specular reflection coefficient for this frequency, with a range of values between 0.0 and 1.0. Indicates the energy reflected back in a distinct outgoing direction.

Name	Type	Default	Usage	Description
diffuseScattering	number	0.0	0	The diffused scattering coefficient for this frequency, with a range of values between 0.0 and 1.0. Indicates the energy that is diffusely scattered back from the material.
transmission	number	0.0	0	The transmission coefficient for this frequency, with a range of values between 0.0 and 1.0. Indicates the energy which passes through the material without changing the direction of the sound.
coupling	number	0.0	0	The coupling coefficient for this frequency, with a range of values between 0.0 and 1.0. Indicates the energy which excites vibrations in the structure and is reemitted by the entire structure.

The binary format of the samples of the frequency characteristics is given in the following table.

Syntax	Length (bits)	Type	Semantics
frequency	16	uint(16)	The frequency for associated with the following coefficients, with values between 1 and 24000.

Syntax	Length (bits)	Type	Semantics
specularReflection	32	float	The specular reflection coefficient for this frequency, with a range of values between 0.0 and 1.0. Indicates the energy reflected back in a distinct outgoing direction.
diffuseScattering	32	float	The diffused scattering coefficient for this frequency, with a range of values between 0.0 and 1.0. Indicates the energy that is diffusely scattered back from the material.
transmission	32	float	The transmission coefficient for this frequency, with a range of values between 0.0 and 1.0. Indicates the energy which passes through the material without changing the direction of the sound.
coupling			The coupling coefficient for this frequency, with a range of values between 0.0 and 1.0. Indicates the energy which excites vibrations in the structure and is reemitted by the entire structure.

1.3.3. Processing Model

An acoustic material is described via that a list of elements, where each element holds four coefficients and an associated frequency.

The coefficients are:

- specular reflection, which represents the energy being reflected in a distinct outgoing direction from the direct sound.

- diffused scattering, which represents energy being diffusely scattering back from the material.
- transmission, which represents the energy that is passed through the material without changing the direction.
- coupling, which represents the energy that excites vibrations in the structure and is re-emitted by the entire structure.

The sum of these four coefficients, per frequency, must be less than or equal to 1, and be greater than or equal to 0. The difference between 1 and the sum of the four coefficients, per frequency, represents the energy that is dissipated into heat.

Chapter 2. ISOBMFF

2.1. Improvements for MPEG-I SD random access description

Source: [m58853](#)

2.1.1. General

For random access of the MPEG-I Scene Description data in a ISOBMFF file tracks, play of the track must start from either a sync sample or a redundant coding sample containing glTF JSON document. Draft FDIS of ISO/IEC 23090-14 Scene Description for MPEG Media indicates that glTF JSON documents shall be marked as sync samples and potential usage of redundant samples for random access but it does not provide detailed descriptions on how to process such samples for random access. This contribution proposes improvements on such description to avoid any confusion by the readers.

2.1.2. Characteristics of random access points of MPEG-I Scene Description

For traditional audio-visual media data, sync samples are simply considered as random access points as processing of a sync sample is same for a decoder playing a sync sample as the first sample and a decoder already processed other sync samples and non-sync samples. When a sync sample of traditional audio-visual media data is processed the result of previously processed samples does not have to be preserved as they are not used for decoding of a sync sample and a decoder is fully refreshed regardless of the status of the decoder before processing a sync sample. This processing model cannot be simply applied to the processing of a sync sample of scene description data as the status of Presentation Engine should not be fully refreshed and the status of Presentation Engine before processing a sync sample needs to be preserved for efficient processing. Therefore, appropriate processing model of sync sample of scene description needs to be described.

Table 1. Comparison of characteristics of sync samples characteristics of sync samples traditional audio-visual media scene description data dependency to the previous samples No No continuity of the decoder status No Yes

As shown in the Table 1, characteristics of sync sample of traditional audio-visual data and scene description data are different. For traditional audio-visual media, sync samples are not dependent to the previous samples and continuity of the data from the previous sample does not exist. However, for scene description data, sync samples are not dependent to the previous samples but continuity of the data from the previous sample may exist.

2.1.3. Description and processing of random access points

2.1.3.1. Random access points with sync samples

One type of random access point is sync sample. Currently, the specification is silent about the case of having a sync sample in the middle of a track and how such samples should be process by a Presentation Engine already in the processing of that track without breaking continuity of the

Presentation Engine. So, there must be description about how to process sync samples by a Presentation Engine already in the processing of a track. In this case, an ISOBMFF file track carrying scene description data can have more than one sync sample and all of each sync samples will contain a glTF JSON document which defines the status of the nodes at the presentation time of the sync sample. The Presentation Engine which has not processed any sample before the current sync sample can process a sync sample as normal scene description document. However, the Presentation Engine already processed any samples before the current sync sample in decoding order should process a sync sample as scene update even though document in the sample is not in the form of JSON patch. Therefore, the description about such processing model should be defined. Otherwise, there should be a restriction that only one sync sample is allowed in the track with MPEG-I Scene Description data.

2.1.3.2. Random access points with redundant coding

The other type of random access point is redundant coding sample. Currently, the specification mentions that the scene description data track can contain some non-sync samples which have `sample_has_redundancy` flag set to '1'. As such samples will be parsed by a Presentation Engine starting play from such sample and ignored by a Presentation Engine already in the processing of a track, this sample will not break continuity of a Presentation Engine already in the processing of a track. To use such samples as a random access point, such sample should carry a glTF JSON document and the document should have the description of a scene same as the scene at the composition time of that sample. In addition, it also needs to be mentioned that there should be no update of scene between the sample preceding such samples and the sample succeeding such samples.

Figure 3 shows an example with redundant samples for random access. In this example, a track with scene description data has two redundant samples denoted as R. The redundant sample R8 whose composition time is between U7 and U9 contains a glTF JSON document contains description of the scene at the time of the composition time of R8. The The Presentation Engine starting from middle of the track starts play either R5 or R8, then play U6 or U9, respectively. The The Presentation Engine starting from the beginning of the track starts play D0 and ignore R5 and R8. As the sample duration of U4 and U7 will be extended by sample duration of R5 and R8, respectively, the scene description information in U4 and U7 must consider that the Presentation Engine will play it longer than the duration of the sample containing it. For example, the animation of active scene of the Presentation Engine according to the animation samplers provided by the sample U4 and the samples before that sample may continue until it receives any updated animation samplers by the U6 sample or the samples after that sample.

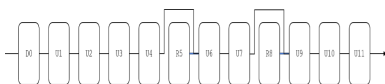


Figure 3. An example structure of scene description data with shadow sync samples

Therefore some additional description about the scene description for such samples should be provided.

2.1.4. Proposed text improvements

2.1.4.1. Sync Samples

It is proposed to add a section about processing of sync samples as follows.

Processing of sync sample

When no nodes in the currently active scene of the Presentation Engine matches a node in a glTF JSON document from a sync sample, the Presentation Engine shall add such node and interact with the MAF to fetch any new content associated with the scene update. When a node in the currently active scene of the Presentation Engine dose not match to any nodes in a glTF JSON document from a sync sample, such nodes shall be removed from the currently active scene of the Presentation Engine. When a node in the currently active scene of the Presentation Engine matches a node in a glTF JSON document from a sync sample, then the status of such node shall be updated to the status of the node described by the sync sample.

2.1.4.2. Redundant coding

It is proposed to improve a section about sample redundancies in section 8.7 of ISO/IEC 23090-14 as follows.

Sample redundancies

For all tracks defined in this document, if a sample has its sample_has_redundancy flag set to '1' and sample_depends_on flag set to '2', then it is expected that that sample contains a glTF JSON document describing the status of the scene at the compsoition time of that sample and would only be made available by the ISOBMFF parser to the Presentation Engine if the processing of the file starts with this sample. Otherwise, it is expected that the sample be ignored, and that processing of the current sample is continued beyond the duration of current sample for a duration equal to the duration of the ignored sample, as defined in ISO/IEC 14496-12.

If the scene description preceding the sample ignored, then the Presentation Engine should continue play of the currently active scene until it receives any updates from the next samples after the sample ignored. Therefore, the scene description in the sample immediately preceding the sample in decoding order whose sample_has_redundancy set to '1' and sample_depends_on set to '2' should consider that the Presentation Engine will play the scene beyond the duration of that sample by the amount of the duration of the next sample. In addition, the glTF JSON document in the sample whose sample_has sample_has_redundancy set to '1' and sample_depends_on set to '2' shall not introduce any scene description which make the status of active scene of a Presentation Engine different from the stauts of the active scene of a Presentation Engine played immediately preceding this sample during the time between the composition time of this sample and the composition time of immediately succeding sample.

2.2. On sample formats for lighting information

Source: [m65312](#)

2.2.1. Introduction

At MPEG #143, the SC29 WG03 Systems issued the Text of ISO/IEC 23090-14 DAM 2 Support for

haptics, augmented reality, avatars, interactivity and lighting (N00942).

Among other features, the amendment enables the signalling of lighting information in the scene description document as follows:

1. Image-based lighting
2. Punctual light sources

Both types of lighting information can either be explicitly signalled in the scene description as static information or be provided from accessors. For the image-based lighting, the extension `MPEG_lights_texture_based` provides references to accessors for the rotation, intensity and irradiances coefficients. For the punctual light sources, the extension `MPEG_light_punctual` provides references to accessor for the colour, intensity and range.

Since the specular images are suitable for storage in ISOBMFF files as static pictures or video sequences (e.g. like in test files captured using ARCore), but the current specifications lacks of the ability to store in such ISOBMFF the rest of the lighting information.

Therefore, this contribution proposes to define a sample format for all the lighting information such that the scene creator can store all this information in a unified way.

In the v2 of the document, following discussion in session, this contribution also provides alternative designs that were proposed. Those alternatives are:

- Defining the sample entry codes, e.g. ‘puli’, but not the sample format which is defined by the time accessor
- Defining a single sample entry code, e.g. ‘sdmt’, with samples containing different parameters.

Those three alternatives needs to be studied for the next meeting.

2.2.2. Lighting information signalling

Lighting extension	Attribute	Accessor type
<code>MPEG_lights_texture_based</code>	rotation	componentType = 5126 (float), type = VEC4, count = 1
<code>MPEG_lights_texture_based</code>	intensity	componentType = 5126 (float), type = SCALAR, count = 1
<code>MPEG_lights_texture_based</code>	irradiance	componentType = 5126 (float), type = SCALAR, count = 27
<code>MPEG_light_punctual</code>	color	componentType = 5126 (float), type = VEC3, count = 1

Lighting extension	Attribute	Accessor type
MPEG_light_punctual	Intensity	componentType = 5126 (float), type = SCALAR, count = 1
MPEG_light_punctual	range	componentType = 5126 (float), type = SCALAR, count = 1

2.2.3. Proposals

2.2.3.1. Option #1: Per metadata tracks

2.2.3.1.1. Design principles for file encapsulation

Principle #1: A light source is contained into one track

For a punctual light, there are three attributes. One approach is to have one track per attribute, another is to have one track providing the three attributes. We believe that parsing one track for all three attributes is friendlier for the application rather than getting all the information from multiple tracks.

Principle #2: Elements can be configured to be optional

In some cases, some attributes of a light source are varying over time and some are static for the duration of the scene. For instance, the intensity attribute of a light may change during a scene while the color attribute may remain the same. In this case, it would be inefficient to repeat the color information for every sample whenever the intensity does change. Therefore, it is desirable that the presence of the attribute in the sample is gated by a flag.

Principle #3: Light sources multiplexing in samples

Especially for punctual lights in virtual scenes, there can be several light sources to describe. To make the parsing simpler for the application, it is desirable to allow storing multiple light sources in the same tracks, although the content creator may still decide which light sources to group together. Therefore, it is desirable that the sample format allows for describing several light sources, i.e. enabling a “light source multiplexing”.

2.2.3.1.2. Illustration of proposed file structures

For punctual lights, the content creator can create one or more tracks (with sample entry code ‘puli’) for storing the related information.



Figure 4. Carriage of punctual light information in timed metadata track ('puli')

For texture-based lights, the content creator can create one or more tracks (with sample entry code ‘tbli’) for storing the related information. For the specular images since they are video sequences, conventional 2D video tracks are used.

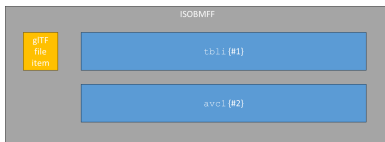


Figure 5. Carriage of texture-based light information in timed metadata track ('tbli')

2.2.3.1.3. Text proposal

2.2.3.1.3.1. Carriage format for lighting information

2.2.3.1.3.1.1. General

A timed metadata track can be used to provide the lighting information related to a given light source. The light source can be of two types, punctual as defined in [MPEG_light_punctual] or texture-based as defined in [MPEG_lights_texture_based]. The sample timing of the metadata track defines the time instant of a lighting sample to which the lighting information in the sample applies.

In the scene description document, the specified values are provided by referring to an accessor with MPEG_accessor_timed extension.

2.2.3.1.3.2. Punctual lights sample entry*

Definition*

Sample Entry Type: 'puli'

Container: Sample Description Box ('std')

Mandatory: No

Quantity: 0 or 1

A punctual light sample entry identifies a track containing lighting information related to punctual lights as defined in [MPEG_light_punctual].

2.2.3.1.3.2.1. Syntax

```

aligned(8) class PunctionalLLightSampleEntry( )
extends MetadataSampleEntry( 'puli' ) {
    unsigned int(16)    number_of_light_sources;
    unsigned int(1) color_is_static;
    unsigned int(1) intensity_is_static;
    unsigned int(1) range_is_static;
    bit(5) reserved;
    if(color_is_static == 1) {
        unsigned int(16)    color[3];
    }
    if(intensity_is_static == 1) {
        unsigned int(16)    intensity;
    }
    if(range_is_static == 1) {
        unsigned int(16)    range;
    }
}

```

2.2.3.1.3.2.2. Semantics*

number_of_light_sources specifies the number of light sources described in the samples.

color_is_static indicates that the color attribute is present in the sample entry and not in samples.

intensity_is_static indicates that the intensity attribute is present in the sample entry and not in samples.

range_is_static indicates that the range attribute is present in the sample entry and not in samples.

color is an array of three fixed-point 0.16 number that indicates the value of the color attribute of the light as defined in the color attribute of the KHR_lights_punctual extension.

intensity a fixed-point 0.16 number that indicates the value of the intensity attribute of the light as defined in the intensity attribute of the KHR_lights_punctual extension.

range a fixed-point 8.8 number that indicates the value of the range attribute of the light as defined in the range attribute of the KHR_lights_punctual extension.

2.2.3.1.3.3. Punctual lights sample format*

2.2.3.1.3.3.1. General*

The sample format includes the attributes of a punctual light for each light source described by the track.

2.2.3.1.3.3.2. Syntax*

```

class PunctualLightsInfo(
    int color_is_static,
    int intensity_is_static,
    int range_is_static) {
    if(color_is_static == 0) {
        unsigned int(16)    color[3];
    }
    if(intensity_is_static == 0) {
        unsigned int(16)    intensity;
    }
    if(range_is_static == 0) {
        unsigned int(16)    range;
    }
}

```

```

aligned(8) class PunctualLightsSample(
    int number_of_light_sources,
    int color_is_static,
    int intensity_is_static,
    int range_is_static) {
    PunctualLightsInfo light_info(
        color_is_static,
        intensity_is_static,
        range_is_static)[number_of_light_sources];
}

```

2.2.3.1.3.3.3. Semantics

color is an array of three fixed-point 0.16 number that indicates the value of the color attribute of the light as defined in the color attribute of the KHR_lights_punctual extension.

intensity a fixed-point 0.16 number that indicates the value of the intensity attribute of the light as defined in the intensity attribute of the KHR_lights_punctual extension.

range a fixed-point 8.8 number that indicates the value of the range attribute of the light as defined in the range attribute of the KHR_lights_punctual extension.

2.2.3.1.3.4. Texture-based lights sample entry*

2.2.3.1.3.4.1. Definition

Sample Entry Type: 'tbli'

Container: Sample Description Box ('stds')

Mandatory: No

Quantity: 0 or 1

A texture-based light sample entry identifies a track containing lighting information related to texture-based lights as defined in [MPEG_lights_texture_based].

2.2.3.1.3.4.2. Syntax

```
aligned(8) class TextureBasedLLightSampleEntry( )
extends MetadataSampleEntry( 'tbli' ) {
    unsigned int(16) number_of_light_sources;
    unsigned int(1) rotation_is_static;
    unsigned int(1) intensity_is_static;
    unsigned int(1) irradiance_coefficients_are_static;
    bit(5)          reserved;
    if(rotation_is_static == 1) {
        unsigned int(16) color[3];
    }
    if(intensity_is_static == 1) {
        unsigned int(16) intensity;
    }
    if(irradiance_coefficients_are_static == 1) {
        float(32)          irradiance_coefficients[27];
    }
}
```

2.2.3.1.3.4.3. Semantics

`number_of_light_sources` specifies the number of light sources described in the samples.

`rotation_is_static` indicates that the rotation attribute is present in the sample entry and not in samples.

`intensity_is_static` indicates that the intensity attribute is present in the sample entry and not in samples.

`irradiance_coefficients_are_static` indicates that the irradiance coefficients attribute are present in the sample entry and not in samples.

`rotation` is an array of four fixed-point 0.32 signed integer that indicates the value of the quaternion representing the rotation attribute of the light as defined in the rotation attribute of the `EXT_lights_image_based` extension.

`intensity` a fixed-point 0.16 number that indicates the value of the intensity attribute of the light as defined in the intensity attribute of the `EXT_lights_image_based` extension.

`irradiance_coefficients` is a sequence of 27 32-bit float numbers that indicates the value of the irradiance coefficients of the light as defined in the irradiance attribute of the `EXT_lights_image_based` extension.

2.2.3.1.3.5. Texture-based lights sample format

2.2.3.1.3.5.1. General

The sample format includes the attributes of a texture-based light for each light source described by the track. ===== Syntax

```

class TextureBasedLightsInfo(
    int rotation_is_static,
    int intensity_is_static,
    int irradiance_coefficients_are_static) {
    if(rotation_is_static == 0) {
        signed int(32) rotation[4];
    }
    if(intensity_is_static == 0) {
        unsigned int(16) intensity;
    }
    if(irradiance_coefficients_are_static == 0) {
        float(32) irradiance_coefficients[27];
    }
}

```

```

aligned(8) class TextureBasedLightsSample(
    int number_of_light_sources,
    int rotation_is_static,
    int intensity_is_static,
    int irradiance_coefficients_are_static) {
    TextureBasedLightsInfo light_info(
        rotation_is_static,
        intensity_is_static,
        irradiance_coefficients_are_static)[number_of_light_sources];
}

```

2.2.3.1.3.5.2. Semantics

rotation is an array of four fixed-point 0.32 signed integer that indicates the value of the quaternion representing the rotation attribute of the light as defined in the rotation attribute of the EXT_lights_image_based extension.

intensity a fixed-point 0.16 number that indicates the value of the intensity attribute of the light as defined in the intensity attribute of the EXT_lights_image_based extension.

irradiance_coefficients is a sequence of 27 32-bit float numbers that indicates the value of the irradiance coefficients of the light as defined in the irradiance attribute of the EXT_lights_image_based extension.

2.2.3.2. Option #2: Unspecified sample format

In this option, we would only define the sample entry (empty) and let the typed accessor in the glTF to describe how the samples are formed.

```

aligned(8) class PunctionalLLLightSampleEntry( )
    extends MetadataSampleEntry( 'puli' ) {
}

```



```
aligned(8) class TextureBasedLLightSampleEntry( )
extends MetadataSampleEntry( 'tbli' ) {
}
```

2.2.3.3. Option #3: Generic sample definition for SD timed metadata

In this option, we would define a single sample entry code and sample format that accommodates all the timed metadata defined in SD.

For instance, this is a possible sample entry definition when considering the punctual and texture-based lighting extension. Note that this should be extended to the other timed metadata present in SD v1 if we move forward with this approach.

```
class PunctionalLLightConfig( )
    unsigned int(16)    number_of_light_sources;
    unsigned int(1) color_is_static;
    unsigned int(1) intensity_is_static;
    unsigned int(1) range_is_static;
    bit(5)             reserved;
    if(color_is_static == 1) {
        unsigned int(16)    color[3];
    }
    if(intensity_is_static == 1) {
        unsigned int(16)    intensity;
    }
    if(range_is_static == 1) {
        unsigned int(16)    range;
    }
}
```

```
class TextureBasedLightingConfig( ) {
    unsigned int(16) number_of_light_sources;
    unsigned int(1) rotation_is_static;
    unsigned int(1) intensity_is_static;
    unsigned int(1) irradiance_coefficients_are_static;
    bit(5)             reserved;
    if(rotation_is_static == 1) {
        unsigned int(16)    color[3];
    }
    if(intensity_is_static == 1) {
        unsigned int(16)    intensity;
    }
    if(irradiance_coefficients_are_static == 1) {
        float(32)           irradiance_coefficients[27];
    }
}
```

```

aligned(8) class SceneDescriptionMetadataSampleEntry( )
extends MetadataSampleEntry( 'sdmt' ) {
    unsigned int(3) metadata_type;

    switch(metadata_type) {
        case 0:
            PunctionalLLightConfig config;
            return;
        case 1:
            TextureBasedLightingConfig config;
            return;
    }
}

```

Then the text would say that if `metadata_type` is equal to 0 then the sample is `PunctualLightsSample`, if equal to 1 then the sample is `TextureBasedLightsSample`.

```

class PunctualLightsInfo(
    int color_is_static,
    int intensity_is_static,
    int range_is_static) {
    if(color_is_static == 0) {
        unsigned int(16) color[3];
    }
    if(intensity_is_static == 0) {
        unsigned int(16) intensity;
    }
    if(range_is_static == 0) {
        unsigned int(16) range;
    }
}

```

```

aligned(8) class PunctualLightsSample(
    int number_of_light_sources,
    int color_is_static,
    int intensity_is_static,
    int range_is_static) {
    PunctualLightsInfo light_info(
        color_is_static,
        intensity_is_static,
        range_is_static)[number_of_light_sources];
}

```

```

class TextureBasedLightsInfo(
    int rotation_is_static,
    int intensity_is_static,
    int irradiance_coefficients_are_static) {
    if(rotation_is_static == 0) {
        signed int(32) rotation[4];
    }
    if(intensity_is_static == 0) {
        unsigned int(16) intensity;
    }
    if(irradiance_coefficients_are_static == 0) {
        float(32) irradiance_coefficients[27];
    }
}

```

```

aligned(8) class TextureBasedLightsSample(
    int number_of_light_sources,
    int rotation_is_static,
    int intensity_is_static,
    int irradiance_coefficients_are_static) {
    TextureBasedLightsInfo light_info(
        rotation_is_static,
        intensity_is_static,
        irradiance_coefficients_are_static)[number_of_light_sources];
}

```

Chapter 3. Codec Support

3.1. Dynamic mesh support in scene description

Source: [m57410](#)

3.1.1. Introduction

The support for dynamic meshes in scene description complements the support for dynamic point clouds. A dynamic mesh is a timed sequence of a mesh representation. A mesh consists of a set of attributes such as vertex positions, and normals. It also has connectivity information, usually in the form of a description of faces that usually are in triangular shape. A face is typically identified by its vertex indices. The faces are usually associated with a material, which is composed of a patch of texture and its light characteristics.

In this contribution, we describe the support for dynamic meshes in scene description.

3.1.2. Design

The support for dynamic meshes in the MPEG-I scene description is limited to the following features:

- Timed attributes such as vertex positions, normals, tangents, texture coordinates, ...
- Timed indices for indicating dynamic connectivity information
- Video texture for the mesh material

All other components of the dynamic mesh are assumed to remain unchanged (e.g. the material, the material properties, the mode, weights and morph targets, ...)

The support for dynamic meshes doesn't require the introduction of any new extensions. The timed attributes and indices are supported through providing a reference to a timed accessor, i.e. an accessor that provides the `MPEG_accessor_timed` extension.

The video texture is supported through referencing a texture that has the `MPEG_texture_video` extension, which in turn references a timed accessor.

3.1.3. Assets and Implementation

Adding support for timed meshes coincides with the start of the activity by the 3DG group on mesh coding. Similar to the point cloud support, the support for dynamic meshes can be done irrespective of whether the mesh is compressed or in raw format. Different pipeline variants may be created to handle decompression and reconstruction.

Initially, a single media pipeline is provided that handles mesh input in raw format based on the wavefront obj format. The assets provided by the mesh compression activity may be used for this purpose. We propose to use the football sequence in a scene description test scenario.

The only deviation is the compression of the texture image sequence into an HEVC bitstream that

can be used with the already supported video texture extension.

The dynamic mesh pipeline implements a file sequence reader that reads the obj file sequence one by one to generate the mesh frames.

Figure 6 depicts the setup:

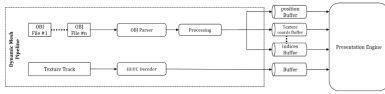


Figure 6. n/a

The Presentation Engine will synchronize the buffer access for each of the components of the mesh by synchronizing the buffer frame timestamps.

3.2. Support for multiple atlases for MIV applications (MPEG142)

Source: [m62515](#)

3.2.1. Multiple atlases

3.2.1.1. Motivation

A V3C bitstream can be decomposed into one or more atlas sub-bitstreams and their associated video sub-bitstreams. The video sub-bitstreams for each atlas may include video-coded occupancy, geometry, and attribute components. In the V3C parameter set (sub-clause 8.4.4.1 in [3]), `vps_atlas_count_minus1` plus 1 indicates the total number of atlases in the current bitstream. The value of `vps_atlas_count_minus1` is in the range of 0 to 63, inclusive.

With the proposal in Section 2.2.1 to support multiple atlases in the `MPEG_primitive_V3C` extension, MPEG-I SD remains future proof to any future derivation of V3C specification which may depend on multiple atlases along with common atlas data. One derived V3C specification in ISO/IEC 23090-12, specified the use of common atlas data which is common to atlases in the V3C bitstream.

3.2.1.2. Overview

The proposals take the following aspects into consideration:

- Logical grouping of the relevant syntax to describe an atlas in the `MPEG_primitive_V3C` extension.
- Use of `atlasID` property to identify the atlas identifier which is equal to `vps_atlas_id[k]` specified in 8.4.4.1 of ISO/IEC 23090-5[3]. In case there are multiple atlases in the V3C bitstream, `atlasID` provides a unique identifier stored in the bitstream to uniquely identify an atlas in `_MPEG_primitive_v3c` extension and establishes a corresponding relation with atlas definition in the bitstream.

3.2.1.3. Array of atlases

A new property is defined under the `_MPEG_primitive_V3C` extension named `atlases`. The `atlases` property is an array of components corresponding to an atlas. The length of the `atlases` array shall be equal to the number of atlases for a V3C object. The properties for an object in the `atlases` array describe the atlas data component and corresponding video-coded components such as attribute, occupancy, and geometry for a V3C object.

The `atlasID` property is an integer values, where each integer value refers to the `vps_atlas_id` specified in sub-clause 8.4.4 in [3] for each atlas in the V3C bitstream.

3.2.1.3.1. MPEG_primitive_V3C

glTF extension to specify support for V3C compressed primitives.

Table 2. `MPEG_primitive_V3C` Properties

	Type	Description	Required
atlases	<code>MPEG_primitive_V3C.atlas [1-*)</code>	An array of atlases	✓ Yes
_MPEG_V3C_CAD	<code>MPEG_primitive_V3C._MPEG_V3C_CAD</code>	This object lists different properties described for the Common Atlas Data in ISO/IEC 23090-5.	No
extensions	<code>object</code>	JSON object with extension-specific objects.	No
extras	<code>any</code>	Application-specific data.	No

Additional properties are allowed.

- **JSON schema:** `MPEG_primitive_V3C.schema.json`

3.2.1.3.1.1. MPEG_primitive_V3C.atlases

An array of atlases

- **Type:** `MPEG_primitive_V3C.atlas [1-*)`
- **Required:** ✓ Yes

3.2.1.3.1.2. MPEG_primitive_V3C._MPEG_V3C_CAD

This object lists different properties described for the Common Atlas Data in ISO/IEC 23090-5.

- **Type:** `MPEG_primitive_V3C._MPEG_V3C_CAD`

- **Required:** No

3.2.1.3.1.3. MPEG_primitive_V3C.extensions

JSON object with extension-specific objects.

- **Type:** *object*
- **Required:** No
- **Type of each property:** Extension

3.2.1.3.1.4. MPEG_primitive_V3C.extras

Application-specific data.

- **Type:** *any*
- **Required:** No

3.2.1.3.2. MPEG_primitive_V3C._MPEG_V3C_CAD

defines the common atlas data for a v3c object

Table 3. *MPEG_primitive_V3C._MPEG_V3C_CAD Properties*

	Type	Description	Required
MIV_view_parameters	<i>integer</i>	indicates the accessor index which is used to refer to the list of MIV view parameters.	✓ Yes
extensions	<i>object</i>	JSON object with extension-specific objects.	No
extras	<i>any</i>	Application-specific data.	No

Additional properties are allowed.

- **JSON schema:** *MPEG_primitive_V3C._MPEG_V3C_CAD.schema.json*

3.2.1.3.2.1. MPEG_primitive_V3C._MPEG_V3C_CAD.MIV_view_parameters

indicates the accessor index which is used to refer to the list of MIV view parameters.

- **Type:** *integer*
- **Required:** ✓ Yes
- **Minimum:** *>= 1*

3.2.1.3.2.2. MPEG_primitive_V3C._MPEG_V3C_CAD.extensions

JSON object with extension-specific objects.

- **Type:** `object`
- **Required:** No
- **Type of each property:** Extension

3.2.1.3.2.3. MPEG_primitive_V3C._MPEG_V3C_CAD.extras

Application-specific data.

- **Type:** `any`
- **Required:** No

3.2.1.3.3. MPEG_primitive_V3C.atlas

glTF extension to specify support for V3C compressed primitives.

Table 4. `MPEG_primitive_V3C.atlas` Properties

	Type	Description	Required
<code>_MPEG_V3C_CONFIG</code>	<code>integer</code>		✓ Yes
<code>_MPEG_V3C_AD</code>	<code>integer</code>		✓ Yes
<code>_MPEG_V3C_GVD_MAPS</code>	<code>integer [1-*)</code>	an array of references to video texture maps.	✓ Yes
<code>_MPEG_V3C_OVD_MAP</code>	<code>integer [0-*)</code>	a reference to a video texture that provides the occupancy map	No
<code>_MPEG_V3C_AVD</code>	<code>MPEG_primitive_V3C.attribute [0-*)</code>		No
<code>_MPEG_V3C_CAD</code>	<code>object</code>	This object lists different properties described for the Common Atlas Data in ISO/IEC 23090-5.	No
<code>extensions</code>	<code>object</code>	JSON object with extension-specific objects.	No
<code>extras</code>	<code>any</code>	Application-specific data.	No

Additional properties are allowed.

- **JSON schema:** `MPEG_primitive_V3C.atlas.schema.json`

3.2.1.3.3.1. MPEG_primitive_V3C.atlas._MPEG_V3C_CONFIG

- **Type:** `integer`
- **Required:** ✓ Yes
- **Minimum:** `>= 0`

3.2.1.3.3.2. MPEG_primitive_V3C.atlas._MPEG_V3C_AD

a reference to the accessor that points to the atlas data.

- **Type:** `integer`
- **Required:** ✓ Yes
- **Minimum:** `>= 0`

3.2.1.3.3.3. MPEG_primitive_V3C.atlas._MPEG_V3C_GVD_MAPS

an array of references to video textures that provide the geometry maps.

- **Type:** `integer [1-*)`
 - Each element in the array **MUST** be greater than or equal to `0`.
- **Required:** ✓ Yes

3.2.1.3.3.4. MPEG_primitive_V3C.atlas._MPEG_V3C_OVD_MAP

a reference to a video texture that provides the occupancy map

- **Type:** `integer [0-*)`
 - Each element in the array **MUST** be greater than or equal to `0`.
- **Required:** No

3.2.1.3.3.5. MPEG_primitive_V3C.atlas._MPEG_V3C_AVD

An array of references to the video textures that provide the attribute data

- **Type:** `MPEG_primitive_V3C.attribute [0-*)`
- **Required:** No

3.2.1.3.3.6. MPEG_primitive_V3C.atlas._MPEG_V3C_CAD

This object lists different properties described for the Common Atlas Data in ISO/IEC 23090-5.

- **Type:** `object`
- **Required:** No

3.2.1.3.3.7. MPEG_primitive_V3C.atlas.extensions

JSON object with extension-specific objects.

- **Type:** `object`
- **Required:** No
- **Type of each property:** Extension

3.2.1.3.3.8. MPEG_primitive_V3C.atlas.extras

Application-specific data.

- **Type:** `any`
- **Required:** No

3.2.1.3.4. MPEG_primitive_V3C.attribute

defines the attribute of a V3C object.

Table 5. MPEG_primitive_V3C.attribute Properties

	Type	Description	Required
type	<code>integer</code>	provides the type of the attribute.	No
maps	<code>integer [1-*)</code>		✓ Yes
extensions	<code>object</code>	JSON object with extension-specific objects.	No
extras	<code>any</code>	Application-specific data.	No

Additional properties are allowed.

- **JSON schema:** `MPEG_primitive_V3C.attribute.schema.json`

3.2.1.3.4.1. MPEG_primitive_V3C.attribute.type

provides the type of the attribute.

- **Type:** `integer`
- **Required:** No
- **Minimum:** `>= 0`
- **Maximum:** `<= 255`

3.2.1.3.4.2. MPEG_primitive_V3C.attribute.maps

provides the references to the corresponding video texture maps.

- **Type:** `integer [1-*)`

- Each element in the array **MUST** be greater than or equal to **0**.

- **Required:** ✓ Yes

3.2.1.3.4.3. MPEG_primitive_V3C.attribute.extensions

JSON object with extension-specific objects.

- **Type:** **object**
- **Required:** No
- **Type of each property:** Extension

3.2.1.3.4.4. MPEG_primitive_V3C.attribute.extras

Application-specific data.

- **Type:** **any**
- **Required:** No

Following is an example illustrating the use of the syntax described in [Section 3.2.1.3.3](#)

```

{
  "meshes": [{
    "name": "v3c_mesh",
    "primitives": [{
      "attributes": {
        "POSITION": 0,
        "COLOR_0": 1
      },
      "mode": 0,
      "extensions": {
        "MPEG_primitive_V3C": {
          "atlases": [{
            "atlasID": 1,
            "_MPEG_V3C_OVD_MAPS": [2],
            "_MPEG_V3C_GVD_MAPS": [3, 4],
            "_MPEG_V3C_AVD": [{
              "type": 0,
              "maps": [5, 6]
            }],
            {
              "type": 4,
              "maps": [7, 8]
            }
          ],
          "_MPEG_V3C_CONFIG": 9,
          "_MPEG_V3C_AD": {
            "buffer_format": "baseline",
            "accessor": 10
          }
        }],
        "_MPEG_V3C_CAD": {
          "MIV_view_parameters": 114
        }
      }
    }
  ]
}

```

3.2.2. References

- [1] m61138, "Support for multiple atlases for MIV application", MPEG 140, Mainz Meeting, October 2022.
- [2] WG7N00553, "Technologies under Consideration on Scene description", MPEG 141, Online, January 2023.
- [3] ISO/IEC 23090-5:2021 Information technology — Coded representation of immersive media — Part 5: Visual volumetric video-based coding (V3C) and video-based point cloud compression (V-

3.3. On G-PCC support

Source: [m63070](#)

3.3.1. Consideration on in-GPU processing

3.3.1.1. Geometry processing

The encoding process for geometry data of G-PCC bitstream is shown in [Figure 7](#).

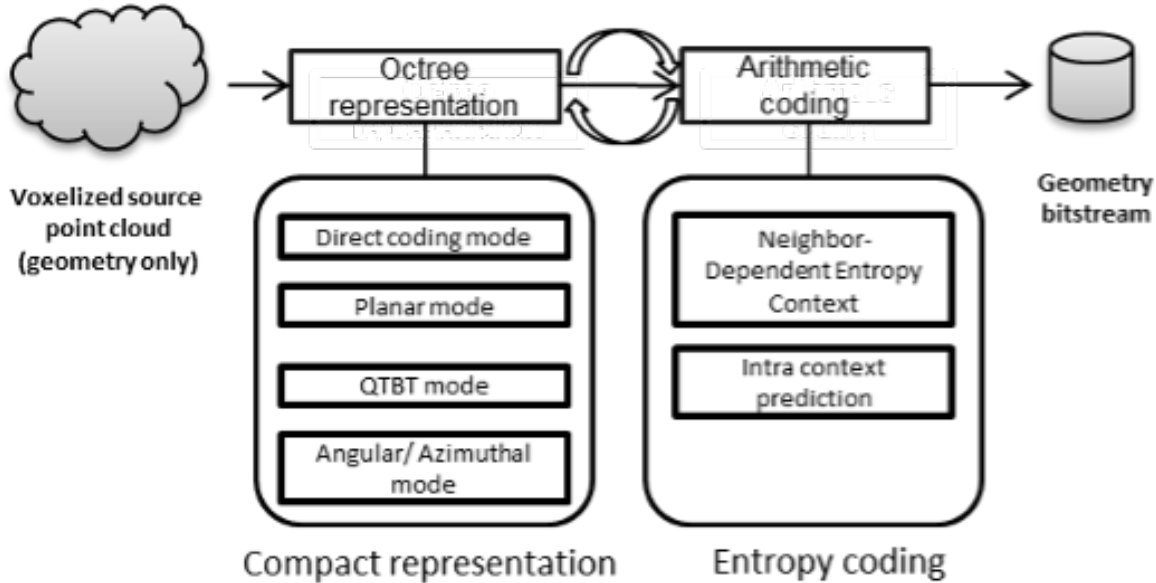


Figure 7. Encoding process for geometry data

Hence, the decoding process flow is as follows:

1. Entropy (CABAC) decode,
2. Octree restoration,
3. and finally, point cloud reconstruction

Here, entropy decoding is not so suitable for in-GPU processing, however, octree restoration can be accelerated by in-GPU processing as restoration of each node of octree can be processed in parallel.

Note that octree restoration and entropy decoding are not independently processed as both are processed in-loop manner when encoding and decoding. This means that both processes cannot be separately distributed to MAF and PE.

Based on the above facts, if whole G-PCC bitstream decoding process (entropy decoding and octree restoration) happens in PE, where basically assumed to equip CPU and GPU, then G-PCC decoding and reconstruction process can be more effective.

3.3.1.2. Attribute processing

The encoding process for attribute data of G-PCC bitstream is shown in [Figure 8](#).

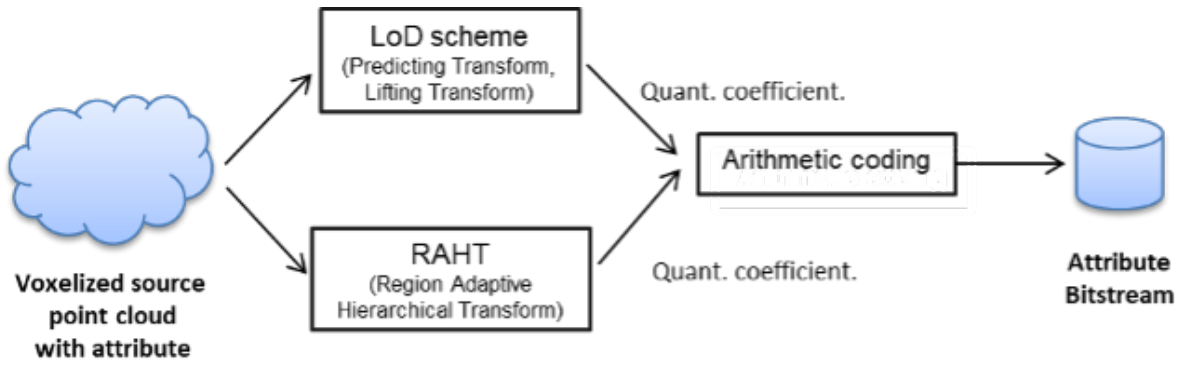


Figure 8. Encoding process for attribute data

Hence, the decoding process flow is as follows:

1. Entropy (CABAC) decode,
2. LoD scheme/RAHT decode,
3. and finally, point cloud reconstruction

There are two types of compression modes which can be selectively utilized depending on the characteristics of the attribute data. The LoD scheme at first re-organizes the points into a set of refinement levels as according to position relationship among points and then encodes each refinement level layer. The RHAT (Region Adaptive Hierarchical Transform) encodes by utilizing frequency conversion based on the density of points. When decoding, both modes are accelerated by parallel processing, and hence it would be more effective when in-GPU processing.

Note that whole attribute decoding process (entropy decoding and LoD scheme/RAHT decoding) needs to be processed after geometry decoding completed.

3.3.1.3. Summary

In summary, by defining the pipeline which is capable of in-GPU processing in PE for decoding geometry/attribute bitstream, it is expected that:

- for geometry, octree restoration process becomes more effective
- for attribute, LoD scheme and RHAT decoding process becomes more effective.

3.3.2. Proposal

Based on the consideration above, it is proposed the following pipeline, where coded geometry and attribute data are transmitted from MAF to PE via buffers, and then PE decodes both and reconstruct point cloud data by utilizing GPU.

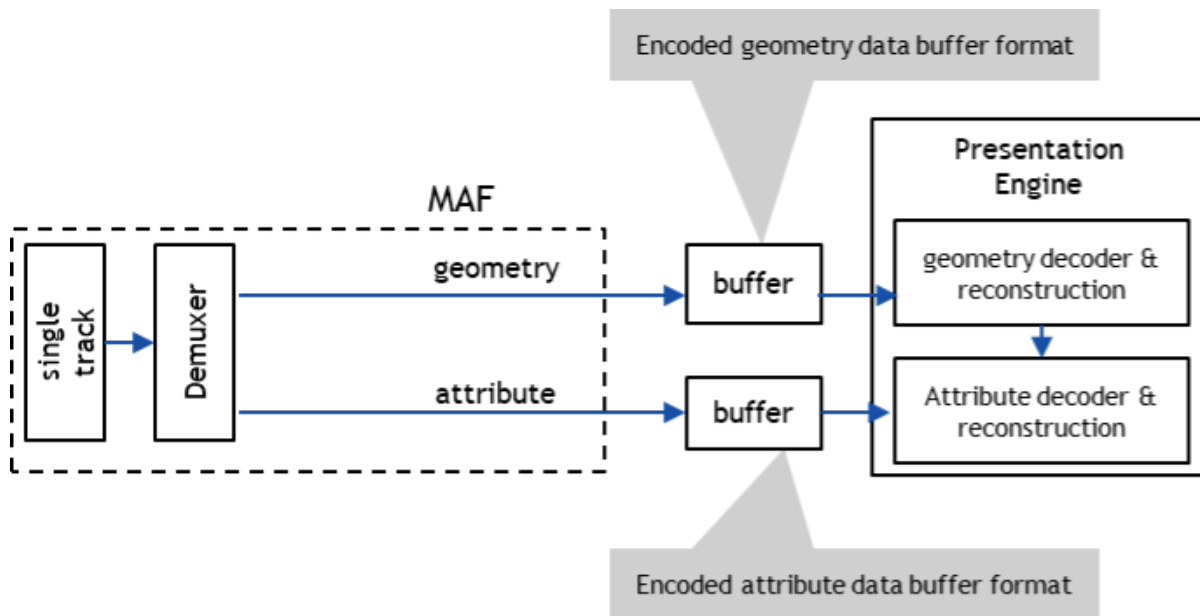


Figure 9. The proposed pipeline

The detail on the proposed extension is described in the annex of this contribution. If the proposed architecture is decided to be valuable, we will come up with the complete specification text.

3.3.3. Reference

1. "Potential improvements of ISO/IEC 23090-14 DAM 1 Support for immersive media codecs in scene description, N00795, MPEG online meeting, January 2023
2. "[SD] G-PCC support in Scene Description", m61856, MPEG online meeting, January 2023

3.3.4. Annex. Proposed MPEG extension

It is proposed new MPEG extension, MPEG_primitive_GPCC.

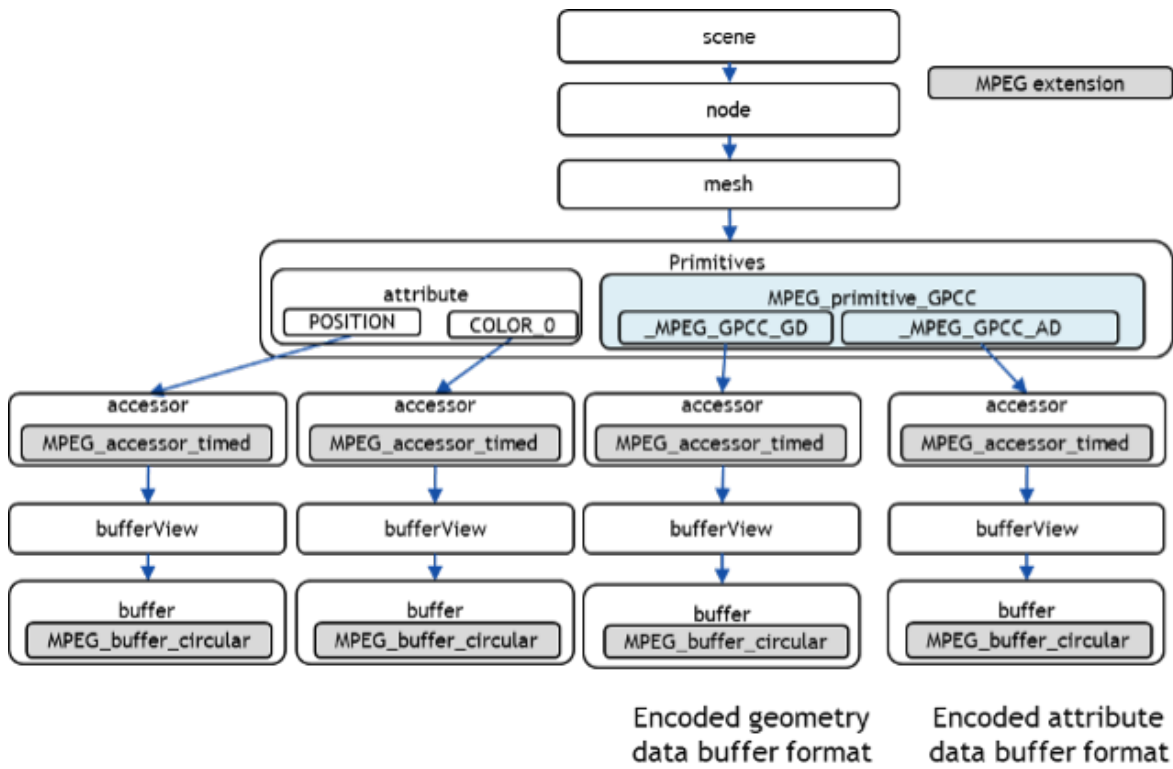


Figure 10. Overview of the *MPEG_primitive_GPCC*

Properties of *MPEG_primitive_GPCC*:

Name	Type	Default	Usage	Description
<i>_MPEG_GPCC_GD</i>	Integer	N/A	O	this component shall provide the index of the timed accessor, which corresponds to the G-PCC compressed geometry data buffer.
<i>_MPEG_GPCC_AD</i>	array(object)	N/A	O	this component shall provide an array of objects, each of which describing an attribute component of the G-PCC compressed mesh primitive.

Name	Type	Default	Usage	Description
Legend: For attributes: M=mandatory, O=optional, OD=optional with default value, CM=conditionally mandatory.				

Properties of _MPEG_GPCC_AD object:

Name	Type	Default	Usage	Description
type	uint8	0	O	provides the type of the attribute as defined by the “GPCC attribute types” in ISO/IEC 23090-9.
accessor	integer	N/A	M	This provides the index of the timed accessor that provides access to the attribute data buffer.
Legend: For attributes: M=mandatory, O=optional, OD=optional with default value, CM=conditionally mandatory.				

Encoded geometry data buffer format:

Field	Type	Description
tlv_count	uint16	tlv encapsulation structure
for(i=0; i<tlv_count; i++)		
tlv_encapsulation()		B.2.1 in ISO/IEC 23090-9

This buffer contains tlv_encapsulation sequence, which is defined in ISO/IEC 23090-9. For geometry

data buffer, only tlv_encapsulation data which tlv_type equals to 0, 1, 2, 5, 6 and 9 can be stored. For attribute data buffer, only tlv_encapsulation data which tlv_type equals to 3, 4, 7 and 8 are stored.

```
"meshes": [
  {
    "name": "g-pcc",
    "primitives": [
      {
        "attributes": {
          "POSITION": 0,
          "COLOR_0": 1
        },
        "mode": 0,
        "extensions": {
          "MPEG_primitive_GPCC": {
            "_MPEG_GPCC_GD": 3,
            "_MPEG_GPCC_AD": {
              "type": 0,
              "accessor": 2,
            },
          },
        },
      },
    ],
  },
]
```

Chapter 4. Data Formats

4.1. Support of glTF CBOR binary format

Source: [m56102](#)

4.1.1. Problem Statement

The Concise Binary Object Representation (CBOR), [IETF RFC 8949](#), represents a concise data format compared with the traditional JSON format. CBOR has similar data objects like JSON in a name/value pair format but in a binary and compact way, also with much more support with key-value types. The result file size is smaller than JSON, in some case, more than 50% of gain has been observed. CBOR is registered in IANA as “application/cbor”.

CBOR is chosen as one of the glTF interchangeable compressed file formats which also has been supported in KhronosGroup due to its compact data size and interchangeability with JSON.

4.1.2. Benefit of CBOR file/data format:

Since the support of CBOR by glTF is getting popular, it is reasonable to add such support into MPEG scene description for:

- Increasing glTF file format interoperability.
- Reducing file size for local storage or cache.
- Increase data transfer speed
- Reducing glTF file transfer latency with minimum processing power at MAF.

4.1.3. CBOR data size comparison example:

When there are lots of repeated data structure and types, CBOR shows a significant compression rate:

Table 6. *n/a*

Test.json	Test.cbor	Compression Rate
13MB	258Bytes	1:1000000

4.1.4. Use Cases

4.1.4.1. CBOR binary data associated with “url”

glTF supports an external binary data expressed inline in a binary data blob. As mentioned above, CBOR is registered in IANA as “application/cbor”. When CBOR is used, binary data may be associated directly under the “url” parameter as follows:

```
{
  "url": "application/cbor:xxxxxxx"
}
```

4.1.4.2. Using CBOR file instead of JSON

A compatible CBOR file (example.cbor) may be sent to MAF as an input instead of JSON (example.glTF). In this case, MAF should have capability to identify, parse and verify the data integrity of the input and parsed the glTF JSON format.

4.1.4.3. Using CBOR as local data storage

As shown in Section 1.1, CBOR may be used to compress glTF file size into local storage if file size is a concern.

4.1.5. Potential Solutions

4.1.5.1. Proposed CBOR Parser API

The proposed CBOR parser API may be used by MAF to translate CBOR input into glTF native supported JSON format. It may also be used as a file compressor to save the large glTF file into local storage or cache.

The CBOR parser API offers the following methods:

Table 7. Description of CBOR Parser API

Method	Brief Description
cbor2Json(FILE)	Convert a CBOR format into a JSON format
json2Cbor(FILE)	Convert a JSON format into a CBOR format
cbor2Json(Object)	Convert a CBOR data blob into a JSON format

The IDL description of this interface is provided in the following table:

```
interface InputFileParser {
  readonly attribute FILE inputFileName;
  readonly attribute FILE outputFileName;
  readonly attribute CBOR cborDataBlob;
  FILE cbor2Json()(FILE cborInput);
  FILE json2Cbor(FILE jsonInput);
  FILE cbor2Json(CBOR cborDataBlob);
  bool    save();
};
```

4.1.5.2. Proposed Test Cases

The testing of the proposed CBOR parser should be implemented under MAF. The use cases could

be the followings:

- If input glTF file is in CBOR format, the output shall be a glTF JSON by using `cbor2Json(FILE)` API
- If there is CBOR binary data specified in “url”, the output shall be a glTF JSON by applying `cbor2Json(Object)` API.
- For local storage or cache purpose, a glTF file is desired to save as a CBOR by using `json2Cbor()` and `save()` interface.

4.1.6. Open Issue Discussion

4.1.6.1. CBOR IPR

No IPR disclosures associated with [IETF RFC 8949](#).

4.1.6.2. CBOR data security

Unlike JSON, CBOR is a binary data serialization, which is not human-readable. It is a safe data format due to its binary nature.

4.1.6.3. Implementation

CBOR has been widely accepted and implemented. It has open-source implementations in most popular languages. (Python, C++, Java and etc).

4.1.6.4. Potential Data format issue

Currently we did not see any incompatible data type has been used in JSON which can not be converted to CBOR or vice versa. More testing may need to be done.

Chapter 5. Interfaces

5.1. On DASH Dynamic Bitrate Adaption with Viewpoint Update

Source: [m56094](#)

5.1.1. Problem Statement

DASH as an adaptive HTTP-based media streaming method enables a client to automatically adjust bitstream bitrate with predefined small bitstream segments based on network condition or buffer status. The advantage of switching up/down the bitrate quality can reduce re-buffer frequency resulting in a smooth playback experience.

The MPEG media extension, “MPEG_media”, enables scene description for playback DASH-based timed media. While the current design of DASH adaptive streaming is implementation-specific, the usage of DASH native switching does not provide optimal networking bandwidth usage in an immersive or 360 scene environments. For example, a view of a media play may not be always in the range of the current viewport, which may cause the unnecessary network resource waste. To provide a smooth timed media playback experience, it is essential to manage how network bandwidth is consumed.

In this contribution, we propose an extension to enable DASH-base timed media bitrate adaptation along with viewport update. In the glTF concept, this enables DASH-based media playback to automatically switch bitrate when the camera on and off focus on a timed media object. In turn, it improves a user’s quality of experience, increase network bandwidth efficiency.

5.1.2. Use Cases

The following scene objects are used for explanation of potential use cases.

Table 8. *n/a*

Asset	Description
A livingroom scene	A glTF asset that represents a living room.
A Big Buck Bunny video	DASH-based Big Buck Bunny video files
A Tears of Steal video	DASH-based Tears of Steal video files

5.1.2.1. One timed media playback

A simple use case is there is only one DASH-based timed media is played in a scene as shown in [Figure 11](#). Currently, the media is rendered based on the MPEG_media extension with configurable parameters such as autoplay, loop, etc. DASH adaptative streaming in this case is used within its native mechanism by switching bitrate based on either network condition or buffer status. The key observation in this case is that the video keeps playing even when the viewport is not in focus. In an adequate network environment, DASH switches to the highest bitrate possible without considering

the overall bandwidth consumption for a scene as a whole. In a less desirable network condition, with a camera's focus is on a set of relatively large bandwidth consumption scene objects such as PCC objects, the unnecessary bandwidth consumption from the ongoing timed media playback is not an optimal solution for view quality of the current viewport.



Figure 11. One DASH-based Timed Media Playback

5.1.2.2. More than one timed media playback

When there is more than one timed media is played at the same time, as shown in Figure 12, network bandwidth usage is similar to the use case in Section 5.1.2.1. However, the situation may get worse when all of the timed media are in a high-resolution setup. The lack of balancing network resources for each of the media play will worsen the view quality.

There are couple of scenarios in this use case:

- There is more than one DASH-based timed media in the current camera's viewport
- There are other DASH-based timed medias outside of camera's current viewport



Figure 12. Two DASH-based Timed Media Playback

Therefore, providing a means to MAF with configurable bandwidth usage for each of the DASH-based timed media may become a critical feature for scene description.

5.1.3. Current Scene Description Support and Gaps

5.1.3.1. Support of viewpoint data fetching

At this moment, the media access API provided in the MAF supports fetching based on “viewinfo” by using the following defined programming interface:

```
interface Pipeline {  
    ..  
    void    startFetching(TimeInfo timeInfo, ViewInfo viewInfo);  
};
```

The “ViewInfo” data structure is as follows:

```
interface ViewInfo {  
    attribute Pose pose;  
    attribute Transform objectPosition;  
};
```

By definition, the MAF may use the “viewinfo” to optimize the streaming of the requested media based on the camera’s view distance and orientation of the viewer. Currently, the following parameters are defined in “viewinfo”:

- Pose
- Transform

5.1.3.2. Gaps Analysis

It is unclear how API and “viewinfo” data structure specified in [Section 5.1.3.1](#) may be used to do the following:

- How exactly the “viewinfo” is used to identify there are one or more DASH-based timed media in the current viewport?
- How exactly the “viewinfo” is used to identify which media is current in focus of a viewpoint, in the case when there is more than one DASH-based timed media in the same viewport?
- How does the current MAF deal with DASH-based timed media fetching including both inside and outside of the current viewport? That is being said, from a system efficiency point of view, the current solution in the CD of 23090-12 does not consider the optimization of data fetching for DASH-based timed media.

5.2. Supporting Multiple Viewers in the Media Access Function

Source: [m58510](#)

5.2.1. General

In the Presentation Engine of the MPEG-I Scene Description architecture, the viewer’s view of the scene is determined by the camera used for rendering the scene from the viewer’s viewpoint. In many use cases, the Presentation Engine runs on the end user’s device and therefore there is only one viewer for the scene and one camera object is used at any given point in time for composition and rendering. Using the camera information provided by the Presentation Engine, the MAF can identify which objects in the scene are within the viewing frustum of the camera at a given time instance.

However, in some scenarios multiple cameras are used for rendering the scene from a number of viewpoints corresponding to different viewers of the same scene (e.g., in multi-viewer applications such as online conferencing applications with multiple users). In such scenarios, information about the cameras used to generate each viewer’s view of the scene, including both intrinsic and extrinsic camera parameters, are required by the MAF to identify and request the appropriate media or

media parts for each viewer.

Since a media pipeline is tightly coupled with the type of the media, it may not be desirable to have multiple media pipelines for the same content for different viewers. Rather, the MAF should allow a single media pipeline for a media content to be used for composition and rendering for different viewers.

5.2.2. Proposed Updates to MAF API

To support media fetching for multi-viewer applications, where each viewer may have their own extrinsic and intrinsic camera parameters, relevant methods in the MAF API and their definition should be updated as follows (updates are in **bold**).

5.2.2.1. Methods

Table 9. *n/a*

Methods	State after success	Description
startFetching()	ACTIVE	Once initialized and in READY state, the Presentation Engine may request the media pipeline to start fetching the requested data. The initialization may be performed using view information for one or more viewers.
updateView()	ACTIVE	Update the current view information. This function is called by the Presentation Engine to update the current view information, if the pose or object position have changed significantly enough to impact media access. It is not expected that every pose change will result in a call to this function. A call to this function shall include the view information for only those views whose parameters have significantly changed.

5.2.2.2. IDL for media pipeline

```

interface Pipeline {
    readonly attribute Buffer          buffers[];
    readonly attribute PipelineState  state;
    attribute          EventHandler  onstatechange;
    void    initialize.  (MediaInfo mediaInfo, BufferInfo bufferInfo[]);
    void    startFetching (TimeInfo timeInfo, ViewInfo viewInfo[]);
    void    updateView.  (ViewInfo viewInfo[]);
    void    stopFetching. ();
    void    destroy.     ();
};

```

5.3. CoAP API support in MAF

Source: [m56739](#)

5.3.1. General

The proposed APIs are assumed under a common CoAP implementation. Take video streaming from CoAP supported devices as an example, those devices are deployed and implemented as a CoAP server that captures, generates, and prepares video binary data (compressed or uncompressed).

5.3.2. MAF as CoAP Client

In this clause, the proposed MAF API in [Table 10](#) applies to the case where the MAF acts as a CoAP client to fetch timed media from the CoAP media server. The CoAP API offers the following methods:

Table 10. Description of CoAP Client API

Method	Brief Description
<code>fetch ()</code>	The MAF sends media resource request to a CoAP server
<code>receive ()</code>	The MAF receives the requested media resource from a CoAP server

5.3.3. MAF as HTTP-CoAP Proxy

In this clause, the proposed MAF API in [Table 11](#) applies to the case where the MAF acts as an HTTP-CoAP proxy.

Table 11. Description of HTTP-CoAP proxy API

Method	Brief Description
<code>hc()</code>	The MAF maps the HTTP requests to CoAP and forward them to CoAP Server

5.4. An Abstract API for Driving External Renderers

Source: [m65395](#)

5.4.1. Render Lock-in API

The Render Lock-in API is an abstract API that is offered by external renderers to align and synchronize their rendering state with the Presentation Engine. This API is used by the Presentation Engine to configure and update the status of the external renderer.

The following table describes the functionality provided by the Render Lock-in API:

Method	Description
init()	Initializes the external renderer by providing the related media source information and their corresponding buffers. It also establishes a session between the Presentation Engine and the external renderer.
configure()	<p>Configures the external renderer to establish an initial alignment and synchronization between the Presentation Engine and the external renderer.</p> <p>The information may include:</p> <ul style="list-style-type: none">• Sync up of the presentation timeline that is maintained by the Presentation Engine• Establishment of the node mapping between scene nodes and referenced elements that are available to the external renderer as part of the source bitstreams. By default, a mapping is assumed between the main camera node that represents the user and the user representation maintained by the external renderer.• Spatial alignment between the scene coordinate system and the coordinate system that is used by the external renderer. This may also include the scaling to align the bounding boxes of the spaces established by the scene description and the source bitstream.• Definition of other elements such as the XR spaces and AR anchors that are tracked by an XR runtime as part of an XR session that is owned by the Presentation Engine.
start() pause() resume() stop()	Allows the Presentation Engine to control the playback of selected media sources associated with the external renderer for interactivity purposes.

Method	Description
update()	Used by the Presentation Engine to update node positions and orientations for which there is a mapping with the external renderer. The transform TRS matrix is relative to the initial pose at the configuration time and is not incremental.
updateGraph()	The Presentation Engine uses the updateGraph function to add, update, or remove a set of nodes to the internal representation of the scene that is maintained by the external renderer.
registerCallback()	The Presentation Engine may provide a callback function to the external renderer to allow it to query the status of certain parameters at any time. This may for example include asking for the current user pose.

The following is a draft description for the API in IDL (ISO/IEC 19516):

```
interface RenderingLockin {
    void allocate(int count);
    void init();
    void configure();
    void start();
    void pause();
    void resume();
    void stop();
    update();
    void updateGraph();
    void registerCallback();
};
```

Chapter 6. MPEG-I Audio in Scene Description

6.1. Immersive audio extension

Source: [m63549](#)

6.1.1. Introduction

A support of spatial audio is provided in ISO/IEC 23090-14 [1] through the MPEG_audio_spatial extension based on the description of source, reverb and listener objects.

To allow a better audio immersion, MPEG-I WG6 immersive audio group has developed a dedicated Encoded Input Format (EIF) [1] to provide acoustic/audio properties in a scene graph for the MPEG immersive audio rendering.

Several WG3/WG6 joint meetings have been held since October to define how to manage in a consistent way both the immersive audio and the MPEG-I Scene Description scene graphs. As detailed in [2], two approaches have been identified for further investigations:

- A first approach based on a hybrid scene description has been selected to be the first target for developing an integrated architecture. As this approach supports the 2 scene graphs, a synchronization mechanism shall be defined through a dedicated API.
- A second approach based on a common scene description

Related to the second approach, a shadow scene concept [3] has been introduced at the MPEG#141 meeting in January 2023 to provide a way for describing invisible simplified geometries to be used by audio renderer. The main benefit of this approach is to share a common glTF-based semantic, but the addition of a new glTF “shadow” scene creates a second scene graph which requires spatial and temporal synchronizations with the graph of the main scene.

This contribution provides an alternative approach to the “shadow” scene concept to support immersive audio. As for the MPEG spatial audio support [1], it relies on a single shared scene graph thus eliminating the need for additional synchronization. This proposed approach is direct and consistent compared to the MPEG interactivity extension where invisible simplified geometries are already defined for collision detection for example.

Note: Further studies are required to ensure that all the audio/acoustic functionalities/features are supported.

6.1.2. Background

Virtual objects may have several representations, each of them targeting a dedicated renderer.

For a sake of illustration, a full VR experience is shown in [Figure 13](#) where a virtual car is moving inside a virtual environment which includes a wall. A user is equipped with a HMD to visualize the 3D virtual scene, an immersive audio headset to hear the motor and a pad controller to drive the

car.

The car and the wall have dedicated representations for audio and visual renderers:

- The car has a geometry for the audio source extent and another geometry for the visual renderer
- The wall has a geometry associated with an acoustic material for the audio renderer and another geometry for the visual renderer

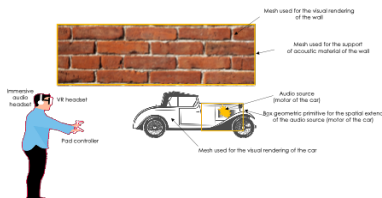


Figure 13. Virtual objects having dedicated representations for audio and visual renderers

Each object representation is dedicated to either the audio or visual renderer. For example, the geometry for the spatial extend of the audio source (motor of the car) shall not be considered by the visual renderer.

When the car is moving, its audio and visual representations shall be spatially and timely consistent.

6.1.3. MPEG-I immersive audio support

A preliminary approach to support MPEG-I immersive audio in a common scene graph is described in this section. Further studies are required to ensure that all acoustic functionalities/features are supported.

In [Table 12](#), we describe and compare the different capabilities of MPEG_audio_spatial and the MPEG-I Audio solution.

Table 12. Comparison between the different capabilities of MPEG_audio_spatial and the MPEG-I Audio solution

	MPEG_audio_spatial	MPEG-I Audio	New Extension
Audio Objects	<ul style="list-style-type: none"> • Listener: A representation of the listener in the scene, typically associated with the camera of the scene. • Source: An audio source that emits sounds in the scene. • Reverb: describes a reverb effect that can be applied to an audio source. 	Scene Objects include a Listener and Audio elements.	Inherit.
Audio Source Type	<ul style="list-style-type: none"> • Object: a mono-channel audio source • HOA 	Audio elements maybe: <ul style="list-style-type: none"> • Object Source • HOA Source 	Inherit.
Object Properties	Inherited from glTF. Velocity can be realized as a TRANSLATION animation. Animations can do more, e.g. scale and rotation.	Position, velocity, isStatic, parent.	Inherit.
Source properties	Pregain, playback speed, attenuation, referenceDistance, reverbFeed and reverbFeedgain, accessors.	Gain, directivity, directiveness, extent, refDistance, audioStream. And for HOA, additional info: group, Is6DoF, transitionDistance.	Inherit + guidelines for extents + better support for hidden geometries + support for HOA groups.
Effects	Reverberation effect.	Reverberation, early reflection, diffraction, portal, dispersion, fade-in/out.	Extend effects.
Scene types	Supports any type of scene. AR through AR anchoring extension.	AR or VR.	Inherit.
Geometry	Inherited from glTF2.0.	Built-in geometry definitions.	Inherit + better support for hidden meshes/primitives.

	MPEG_audio_spatial	MPEG-I Audio	New Extension
Materials	No support for acoustic materials	Support for materials with specular reflection, diffused scattering, transmission, and coupling.	Define acoustic materials.
Voxel Representation	Not supported	Voxel-based geometry and compression.	Add to the new extension.
Mesh compression	None.	Built-in	Add support for external mesh codecs such as V-DMC and Draco (Khronos extension).

As detailed in the MPEG-I Immersive Audio Encoder Input Format (EIF) document [1], audio/acoustic data may be provided at several parts of a scene graph:

- At global/scene level
- At object/node level
- At avatar/user representation
- At mesh primitive level

The following sections identifies new potential MPEG extensions at several levels of a glTF scene graph to support MPEG-I immersive audio as shown in [Figure 14](#) . Note that alternatively, a single extension, as is the case with MPEG_audio_spatial, might be defined instead.

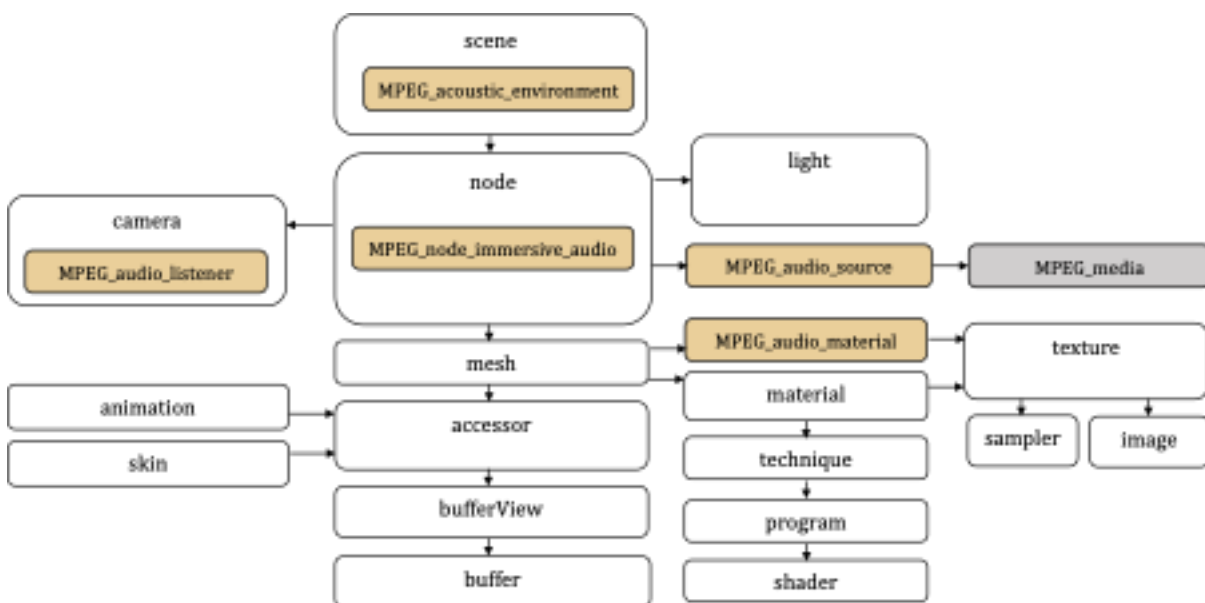


Figure 14. Proposed new MPEG glTF extensions to support MPEG-I immersive audio

6.1.3.1. Audio/acoustic data at global/scene level

The acoustic data relevant for the whole scene or for a specific spatial zone delimited by a static

geometry are defined as acoustic environment data in section 3.9 of EIF document [1]. An environment is characterized by acoustic parameters at defined positions such as:

- The 60 dB reverberation time (RT60)
- The pre-delay time
- The Diffuse-to-Direct-Ratio (DDR)

These acoustic environment data may be provided through a new “MPEG_acoustic_environment” glTF extension at scene level.

6.1.3.2. Audio/acoustic data at node level

A dedicated acoustic extension shall be defined at the node level to support the representation of the related 3D object for the audio renderer.

This new “MPEG_node_immersive_audio” extension typically provides a reference to a mesh geometry having an acoustic material. Thanks to referencing the mesh inside an audio-specific extension, we ensure that this mesh and the related material are only used by the audio renderer and are “invisible” for the visual renderer.

The audio data related to the source which emits sound into the virtual scene may also typically be provided at the node level (in line with the already-existing source object of the MPEG audio spatial extension [1]). The audio source takes benefit from the node position/orientation to define its pose.

The audio source parameters are defined in section 3.2 of EIF document [1] such as:

- The unique ID
- The signal which defines the corresponding audio stream
- The extent which defines a geometry for the spatial extent of the source perceived by the listener in an elevation/azimuth sector
 - As this extent geometry is referenced inside an audio-specific extension, we ensure that this mesh is only used by the audio renderer and is “invisible” for the visual renderer

These audio source data may be provided through a new “MPEG_audio_source” glTF extension at node level.

6.1.3.3. Audio/acoustic data at avatar/user representation level

Basically, an audio listener is implicitly attached to the user experiencing the XR application.

A dedicated MPEG avatar extension is currently being defined to describe the user representation for that XR experience. This extension is attached to a node having a camera component.

Therefore, we may also provide dedicated data related to the audio listener at the avatar node level through a new “MPEG_audio_listener” glTF extension. One potential parameter would be a unique identifier ID, in line with the already-existing listener object of the MPEG audio spatial extension [1])

6.1.3.4. Audio/acoustic material data at mesh primitive level

An acoustic material characterizes the acoustic behavior of surfaces of 3D object. This acoustic material is typically referenced by the mesh geometry provided within the “MPEG_node_immersive_audio” extension.

The parameters are frequency-dependent and are defined in section 3.8 of EIF document [1] such as:

- The specular reflection coefficient (r)
- The diffuse scattering coefficient (s)
- The transmission coefficient (t)
- The coupling coefficient (c)

These acoustic material data may be provided through a new “MPEG_audio_material” glTF extension at mesh primitive level.

6.1.4. References

[1] ISO/IEC 23090-14

[2] MPEG-I Immersive Audio Encoder Input Format v3, N0169, October 2022

[3] Considerations on MPEG-I audio and MPEG-I scene description architectures, N0186, February 2023

[4] Definition of Shadow Scenes, m62227, January 2023

6.2. MPEG-I Audio in Scene Description

Source: [m61180](#)

6.2.1. General

MPEG-I Immersive Audio has been specified in ISO/IEC 23090-4. The specification assumes the presence of an MPEG-I immersive audio renderer that will receive the MPEG-I audio bitstream, a set of MPEG-H audio streams, as well as information about some scene metadata, such as listener’s pose. It will then use the audio scene metadata in the MPEG-I audio bitstream, the decoded MPEG-H bitstreams, and the pose information to render the spatial audio.

[Figure 15](#) depicts the MPEG-I audio architecture:

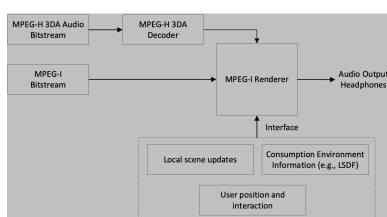


Figure 15. N/A

The MPEG-I render pipeline is depicted by [Figure 16](#):

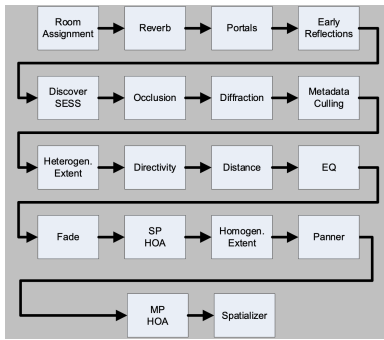


Figure 16. N/A

MPEG-I immersive audio relies on a new scene description format for the audio scene to establish the spatial relationships between the different audio sources.

Ideally, the audio scene metadata should be described as part of a common scene description that includes all media types: visual, audio, haptics, etc. The MPEG-I audio renderer would then be driven by scene metadata extracted from the common scene description.

However, if this is not possible, alternative options may be available. In the first option, the MPEG-I Presentation Engine will be provided with callbacks to allow it to update the audio scene based on information coming from the common scene description. This option is described by [Figure 17](#):

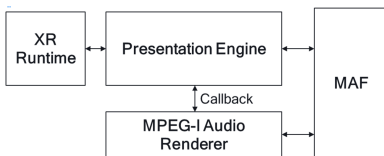


Figure 17. N/A

This option requires that the Presentation Engine gets all the extracted audio scene metadata, so that it can align it with the common scene description.

Another option would be to pre-process the MPEG-I immersive audio bitstream to align it with the common scene description. This option is depicted by [Figure 18](#):

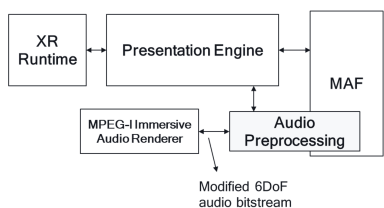


Figure 18. N/A

The pre-processing block may insert scene update MHAS packets to achieve the alignment of the audio scene with the common scene.

Yet another option could be that the common scene description completely overwrites the MPEG-I immersive audio scene with the spatial audio description in the scene description. In essence, it would just use the decoded MPEG-H streams as audio sources.

6.3. Establishing a Mapping between Audio and MPEG-I Scenes

Source: [m65378](#)

6.3.1. General

Systems and Audio groups are discussing the support of MPEG-I Audio in Scene Description. The groups have discussed several ways of achieving this goal, with the most agreed on option being the support of a separate MPEG-I audio stream that is referenced by the scene description document.

This approach is depicted by the following figure:

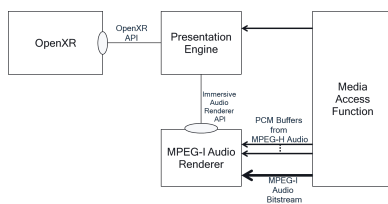


Figure 19. n/a

The MPEG-I Audio bitstream contains a description of the audio scene that is independent of the main scene description consumed by the Presentation Engine. In fact, this approach permits that these two scenes are created completely separately and independently. Proper rendering of both scenes to provide a consistent experience to the user becomes then extremely challenging.

To enable this approach, an alignment between the Presentation Engine and the Audio Renderer is essential. This alignment goes beyond the traditional time alignment but includes also spatial alignment.

6.3.2. Extension for Audio Node Mapping

6.3.2.1. General

The MPEG node mapping extension, identified by `MPEG_node_mapping`, establishes a mapping between the node in the scene description document and an external entity. An example is the mapping between a node that contains a car and an external audio node in an MPEG-I Audio bitstream, with a simplified geometry of that car and the attached audio sources. The following figure depicts that example:

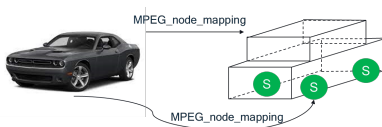


Figure 20. n/a

When present, the `MPEG_node_mapping` extension shall be included in a node object.

6.3.2.2. Semantics

The definition of all objects with the `MPEG_node_mapping` extension is provided in the following

table:

Name	Type	Default	Usage	Description
role	string	“urn:mpeg:sd:role:default”	O	An identifier of the role associated with this mapping. The role may for instance be “urn:mpeg:sd:role:audio-renderer” to indicate that the component is an audio renderer.
source	number	N/A	M	The index in the MPEG_media that provides the media resource that contains the mapped element.
referenceId	number	N/A	M	An identifier of the element in the referenced resource.
transform	array(number)	Identity	O	A 4x4 matrix that supplies the transform used to align the referenced element to the current node.
supportsInteractivity	boolean	false	O	Indicates if interactivity actions applied to the node should be exposed if an API is made available to the Presentation Engine by the renderer of the resource.

6.3.2.3. Processing Model

When processing the MPEG_node_mapping extension, the Presentation Engine shall identify nodes in the scene description that have a node mapping. The Presentation Engine shall determine if the component identified by the indicated role supports the Rendering Alignment API as defined in contribution m65395. If it does, the Presentation Engine shall pass the mapping information to the identified component.

The Presentation Engine shall then use the API to align the rendering with the component as configured over the API.

Chapter 7. Reference Software

7.1. Thoughts on trimesh playback of AR scenes

Source: [m60282](#)

7.1.1. General

The MPEG-I Scene Description standard relies and extends on the Khronos glTF format. While the primary goal of glTF is to represent 3D objects in virtual scenes, the MPEG-I SD work also aims at addressing AR applications wherein 3D objects are integrated into real-world scenes.

Given the requirement for test assets and reference software to guide the standardisation work of MPEG-I SD, this brings challenges to also include test assets for AR applications as well as their integration into the reference software, currently based on trimesh, while both glTF and trimesh are not originally developed for these AR applications.

Therefore, here we aim at starting the discussion on the feasibility of meeting this requirement and presents a possible approach. This approach comprises two main steps:

- Recording a real-world scene as an AR test asset using the AR Session recorder of Google ARCore
- Playing back the recorded an AR test asset inside trimesh (or other renderer)

7.1.2. AR Sessions recording and format

7.1.2.1. AR Session in Google ARCore

The Google ARCore framework provides an API to record an AR Session such that it can be played back at later time. By recording, the function effectively captures and stores the sensors information that are fed as input of the AR algorithms which power the AR application. This way, the playback function can later read those AR session files and recreate the device movement and sensing based on this file and no longer using direct sensor measurements.

This is depicted in [Figure 21](#) available in the [ARCore documentation](#).

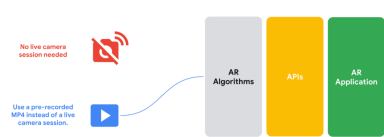


Figure 21. AR Session playback in ARCore

According to the documentation, the recorded AR Session will contain:

- Primary video track (CPU image track, i.e. not the video rendered on the screen)
- Camera depth map from hardware depth sensors, when available
- Gyrometer data
- Accelerometer data

- Custom/user event

7.1.2.2. AR Session file format

In order to test this capability, several recordings were made with ARCore compatible smartphones. The DepthLab Android application developed by Google [\[Ruofei et. al.\]](#)[\[DepthLab\]](#) was used to perform those quick tests. This application demonstrates the capabilities of the ARCore framework to application developers as well as provides a function to record the AR Session via the corresponding ARCore API.

Here are some dump information from the recorded files.

Track # 1 Info - TrackID 1 - TimeScale 90000 - Media Duration 00:00:29.107
Track has 2 edit lists: track duration is 00:00:29.134
Media Info: Language "und (und)" - Type "vide:avc1" - 869 samples
Visual Track layout: x=0 y=0 width=640 height=480
MPEG-4 Config: Visual Stream - ObjectTypeIndication 0x21
AVC/H264 Video - Visual Size 640 x 480
 AVC Info: 1 SPS - 1 PPS - Profile High @ Level 3
 NAL Unit length bits: 32
 SPS#1 hash: 03802E3BC1A1E33FE5B23E626E9E4D37369B6548
 PPS#1 hash: 85644534159E9C005D09E9AC5EACE302A792A46E
Self-synchronized
 RFC6381 Codec Parameters: avc1.64001e
 Average GOP length: 32 samples

Track # 2 Info - TrackID 2 - TimeScale 90000 - Media Duration 00:00:29.107
Track has 2 edit lists: track duration is 00:00:29.134
Media Info: Language "und (und)" - Type "meta:mett" - 869 samples
Textual Metadata Stream - mime application/arcore-video-0
 RFC6381 Codec Parameters: mett
 All samples are sync

Track # 3 Info - TrackID 3 - TimeScale 90000 - Media Duration 00:00:29.109
Media Info: Language "und (und)" - Type "meta:mett" - 5875 samples
Textual Metadata Stream - mime application/arcore-gyro
 RFC6381 Codec Parameters: mett
 All samples are sync

Track # 4 Info - TrackID 4 - TimeScale 90000 - Media Duration 00:00:29.109
Track has 2 edit lists: track duration is 00:00:29.109
Media Info: Language "und (und)" - Type "meta:mett" - 5875 samples
Textual Metadata Stream - mime application/arcore-accel
 RFC6381 Codec Parameters: mett
 All samples are sync

Track # 5 Info - TrackID 5 - TimeScale 90000 - Media Duration 00:00:27.575
Track has 2 edit lists: track duration is 00:00:28.327
Media Info: Language "und (und)" - Type "meta:mett" - 41 samples
Textual Metadata Stream - mime application/arcore-custom-event
 RFC6381 Codec Parameters: mett
 All samples are sync


```

Track # 1 Info - TrackID 1 - TimeScale 90000 - Media Duration 00:00:21.579
Track has 2 edit lists: track duration is 00:00:21.784
Media Info: Language "und (und)" - Type "vide:avc1" - 643 samples
Visual Track layout: x=0 y=0 width=640 height=480
MPEG-4 Config: Visual Stream - ObjectTypeIndication 0x21
AVC/H264 Video - Visual Size 640 x 480
    AVC Info: 1 SPS - 1 PPS - Profile High @ Level 3.1
    NAL Unit length bits: 32
    SPS#1 hash: 217A055E6A89F18FED4CDE98F4039A7B505ACC0B
    PPS#1 hash: 85644534159E9C005D09E9AC5EACE302A792A46E
Self-synchronized
    RFC6381 Codec Parameters: avc1.64001f
    Average GOP length: 32 samples

Track # 2 Info - TrackID 2 - TimeScale 90000 - Media Duration 00:00:21.579
Track has 2 edit lists: track duration is 00:00:21.784
Media Info: Language "und (und)" - Type "meta:mett" - 643 samples
Textual Metadata Stream - mime application/arcore-video-0
    RFC6381 Codec Parameters: mett
    All samples are sync

Track # 3 Info - TrackID 3 - TimeScale 90000 - Media Duration 00:00:21.581
Track has 2 edit lists: track duration is 00:00:21.585
Media Info: Language "und (und)" - Type "meta:mett" - 4444 samples
Textual Metadata Stream - mime application/arcore-gyro
    RFC6381 Codec Parameters: mett
    All samples are sync

Track # 4 Info - TrackID 4 - TimeScale 90000 - Media Duration 00:00:21.581
Media Info: Language "und (und)" - Type "meta:mett" - 4445 samples
Textual Metadata Stream - mime application/arcore-accel
    RFC6381 Codec Parameters: mett
    All samples are sync

Track # 5 Info - TrackID 5 - TimeScale 90000 - Media Duration 00:00:20.312
Track has 2 edit lists: track duration is 00:00:00.753
Media Info: Language "und (und)" - Type "meta:mett" - 28 samples
Textual Metadata Stream - mime application/arcore-custom-event
    RFC6381 Codec Parameters: mett
    All samples are sync

```

As can be seen from those dumps, the generated mp4 files contain: * The main video used for video processing * Gyroscopic data * Acceleration data * User actions (probably the custom-event track) * A mysterious track that has the same number of samples as the video track but only between 84 and 86 bytes per sample depending on the recording

Note that the smartphones used for the test recording were not equipped with depth sensors, e.g. ToF sensor, this should be the reason why there is no depth map video track as stated in the documentation “video file representing the camera’s depth map, recorded from the device’s

hardware depth sensor”.

[Ruofei et. al.] Du, Ruofei, Eric Turner, Maksym Dzitsiuk, Luca Prasso, Ivo Duarte, Jason Dourgarian, Joao Afonso et al. "DepthLab: Real-time 3D interaction with depth maps for mobile augmented reality." In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, pp. 829-843. 2020.

[DepthLab] DepthLab: Real-Time 3D Interaction With Depth Maps for Mobile Augmented Reality (augmentedperception.github.io), <https://augmentedperception.github.io/depthlab/>

7.1.3. AR Session playback in trimesh

As presented in clause [Section 7.1.2](#), the ARCore API provides the ability to record all the information pertaining to an AR session in terms of sensor data and user events.

From such a file, it should then be possible to:

- Determine the position of the smartphone camera over time (even absolute if GPS activated) using the rotation and displacement data.
- Create a point cloud frame/mesh frame from each recorded video frame based on the associated depth map. NOTE If no depth sensor is used for the recording, the depth map should be either generated via an algorithm or retrieved from the ARCore API and stored in the mp4 file using a custom made application.
- Position this point cloud frame/mesh frame in the scene over time.

Once this volumetric data corresponding to the AR Session is generated, this could constitute an AR test asset for MPEG-I Scene Description work which could be then played back in trimesh

Chapter 8. Interactivity framework

8.1. On event-based scene update

Source: [m61812](#)

8.1.1. General

In the 23090-14 DIS document, a scene update mechanism is proposed, with predefined timed updates: A special track in a media content (for instance an ISOBMFF file), provides timed samples that contain patch (i.e., [JSON patch](#)) to be apply to the original scene description file.

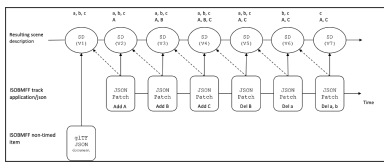


Figure 22. n/a

This mechanism handles pre-defined scene evolution but does not allow describing event-based update, following for instance a user action or any event that may occurred amongst the scene objects at any time. In the MPEG-I Scene Description output document on scene update [ISO/IEC JTC 1/SC 29/WG 3 N0315], a potential solution is presented for event-based scene updates : while a predefined timed scene update is in progress, an event may occur that updates the scene description. Several scenarios are then proposed: apply a patch and switch to a new timed samples track or apply a patch and skip one or more versions in the same track.

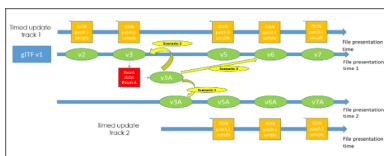


Figure 23. n/a

This mechanism is still strongly related to pre-defined scene evolutions and does not specify how the event that triggers the update is described in the scene description document.

Furthermore, it does not handle the case where the same event that creates a new node may be fired multiple times, like illustrated in the following diagram: A glTF scene contains a description of an event-based update mechanism with the same patch applied each time an event is fired. Some elements of the glTF scene are modified (adding, changing or removing nodes, meshes parameters) but not the event-based update description.

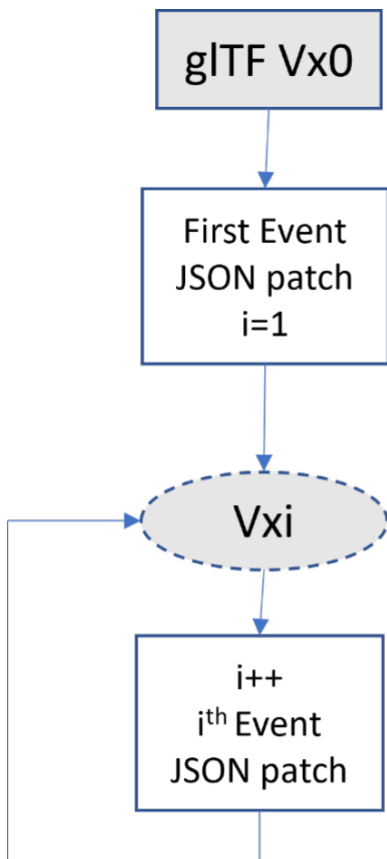


Figure 24. Event-based update diagram

8.1.2. A use case for event based updates

This update diagram is illustrated in the IDCC demo, presented during the last MPEG meeting in Mainz:

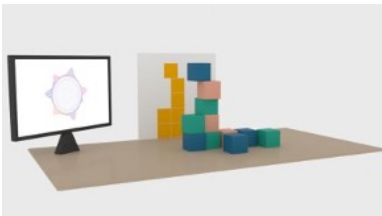


Figure 25. n/a



Figure 26. n/a

The demo presents a game application. An initial scene is first displayed, containing a plane surface, a TV screen displaying a video content and a vertical surface displaying a pattern. The user can add a new cube in the scene by touching the screen, in order to build a cubes stack that matches the displayed pattern. Each time a match occurs, a new scene is loaded with a new pattern and a new video. The game may be multiplayer with the same scene shared between all the connected clients. The scene is synchronized each time an update is performed in one client. A

game server handles the scene synchronization each time an update is performed by a client.

The creation of the cube and the loading of a new scene is currently implemented using proprietary solution, but it could be possible to build a mechanism in line with the MPEG-SD dynamic scene framework.

Two kinds of updates are triggered during the game:

1. During a game phase, each time the user touches the screen to create a cube in front of the pattern, a same scene update/patch is applied. The difference is the position of the user's finger that gives the position where the cube is created and from which it falls. Using the current scene update mechanism, with JSON patch, the creation of a new cube would be performed with 2 patch operations:
 - An “add” operation, that adds a new node in the glTF node array, for instance with a path equal to “/nodes/-“, i.e. a new node created at the end of the array. A new node created in the middle of the nodes array (i.e., with a path equal to “/nodes/2”) would leave the scene in an erroneous status and would need extra patch operations to fix it. We would face other issues if the new “cube” nodes must be created as children of another “cubesStack” node: We would not know in advance the index of the new node since it depends on the number of updates that have already been triggered.
 - A “place” operation that does not exist in the JSON patch specifications. We could use a “replace” operation to set the “translation” or/and “rotation” elements of the new node but:
 - Same as above, we do not know in advance the index of the new node!
 - The value to be applied must be retrieved from user's finger position on the screen! And there is no way to pass this value as an input to the “replace” operation.
2. When the cubes stack matches the pattern, a new scene is loaded with a new pattern:
 - It could be a JSON patch, removing the cube nodes and replacing the pattern with a new one. As above, we do not know the indexes of all the cube nodes and these indexes are needed to remove the nodes. If the nodes have been created as children of a unique parent node, we could just empty the children array of this node. The cube nodes description would remain in the description file.
 - It could be a complete update and a new glTF file is used.

8.1.3. JSON patch limitations

A JSON patch is not a “glTF patch” and does not consider all the characteristics of the JSON tree in a glTF scene description file and particularly the interdependence between elements of different branches of the glTF tree (a node referencing a mesh that references a material, or a node referencing one or more child nodes). It is fine if you know in advance the scene description you want to update and the resulting scene description: The JSON patch can be generated by comparing the 2 JSON description files.

For repetitive event-based updates as described in [Section 8.1.2](#), we don't know the resulting scene and care should be taken when writing the JSON patch. Furthermore, the application, that applies the patch, may need to perform extra operations to complete the update:

- check the consistency of the resulting glTF scene,
- get the index of an array item created with the “-“ JSON patch alias,
- perform extra glTF modifications not handled by JSON patches (set newly created nodes as child of another node, set JSON element to a value only determined at run-time...).

8.1.4. Semantics for event-based update

A new semantic is needed to describe event-based scene update: A semantic that would address the use case (related to pre-defined timed scene updates) as well as the new one introduced in [Section 8.1.2](#).

An approach would be to keep using the JSON patch mechanism, which is already used for the pre-defined timed scene updates. As explained above, the definition of extra parameters would then be required.

Furthermore, the description of the event and its relationship with the scene update could be described with the interactivity framework specified in [ISO/IEC JTC 1/SC 29/WG 3 N0725]. It defines a set of action types that can be executed following a trigger activation. As a reminder, the table above gives the action types that are already specified:

Table 13. Type of action

Action type	Description
“ACTION_ACTIVATE”	Set activation status of a node
“ACTION_TRANSFORM”	Set transform to a node
“ACTION_BLOCK”	Block the transform of a node
“ACTION_ANIMATION”	Select and control an animation
“ACTION_MEDIA”	Select and control a media
“ACTION_MANIPULATE”	Select a manipulate action
“ACTION_SET_MATERIAL”	Set new material to nodes
“ACTION_SET_HAPTIC”	Get haptic feedbacks on a set of nodes

An event-based scene update may be described in a glTF scene description file, using the interactivity extensions specified in [ISO/IEC JTC 1/SC 29/WG 3 N0725]: A trigger element may describe the event (for instance, a “TRIGGER_USER_INPUT” trigger, as defined in [ISO/IEC JTC 1/SC 29/WG 3 N0725]), and an action element (of a new type, to be defined) may describe the update information (a patch to be applied (an array of JSON patch operations) and other parameters used by the application to complete this update). Here is a list of such parameters that may be defined:

- Parameters to place one or more nodes in a position not known in advance. For instance, it may include a position information and a list of nodes. The position parameter may be related to a user input, or a user pose and may use the [OpenXR interaction profile path semantic](#). Each node to position may be identified by one of the patch operations that created or modified it.
- Parameters identifying one or more nodes to be used as parent of one or more newly created nodes. For instance, a list of parent nodes and a list of child nodes. Same as above, each child

node may be identified by one of the patch operations that created or modified it.

- Any other parameters that may be needed for other use cases: flag to share or not a local update with other connected users sharing the same scene, strategy in case the patch fails or gives an inconsistent glTF tree (rollback, fix...), ...

8.2. Physic Support

Source: [m65177](#)

8.2.1. Introduction

This contribution intends to confirm whether the physics support as defined in MPEG-I SD Amd2 (ISO/IEC 23090-14 DAM2: Support for Haptics, Augmented Reality, Avatars, Interactivity, MPEG-I Audio and Lighting) is sufficient or would benefit from some improvements.

[Section 8.2.2](#) of this contribution analyses the consistency of the physic simulations, between two game engines (Unity and Unreal), based on the parameters currently defined in Table 8.2-11. We illustrate the mapping of the MPEG-SD parameters to the game engines (Unity and Unreal) and to USD.

In [Section 8.2.3](#), based on the addition of new parameters, at node and/or scene level, we analyse whether a closer physic simulation between the game engines is obtained.

In [Section 8.2.4](#), proposed changes are provided to MPEG-I SD based on the results of section [Section 8.2.2](#) and [Section 8.2.3](#).

For reference, in the section 8.2.2.2 – Table 8.2-11 (Semantic of the MPEG_node_interactivity), the following physic parameters have been defined to support basic physic simulation from the scene creator (Table-1).

if (type == TRIGGER_COLLISION) {				
Collider	integer	M		the index of the mesh element that provides the collider geometry for the current node. The collider mesh may reference a material.
static	boolean	M		If True, the collider is defined as a static collider.
usePhysics	boolean	M		Indicates if the object shall be considered by the physics simulation.
if (usePhysics){				
useGravity	boolean	M		Indicates if the gravity affects the object

if (type == TRIGGER_COLLISION) {				
mass	number	M		Mass of the object in kilogram.
restitution	number	M		Provides the ratio of the final to initial relative velocity between two objects after they collide
staticFriction	number	M		Unitless static friction coefficient as defined in the Coulomb friction model. Friction is the quantity which prevents surfaces from sliding off each other. Static friction is used when the object is lying still. It will prevent the object from starting to move.
dynamicFriction	number	M		Unitless static friction coefficient as defined in the Coulomb friction model. When a large enough force is applied to the object, a dynamic friction is used, and will attempt to slow down the object while in contact with another.
}				
}				

Table 1 Collision trigger

8.2.2. Analysis of the physic simulation consistency between game engines with the current parameters

8.2.2.1. Parameter mapping

Parameters are mapped as shown in the following Table-2:

MPEG-I SD	Unity	Unreal
useGravity	useGravity	Enable Gravity
Mass	Mass	Mass
Restitution	Physic material > Bounciness	Physic material > Restitution
Static friction	Physic material > Static friction	Physic material > Static friction
Dynamic friction	Physic material > Dynamic friction	Physic material > Friction

Table 2: Physic parameters mapping with game engines

The mapping to USD, is provided in Table-3:

MPEG-I SD	USD
useGravity	UsdPhysicScene::GetGravityDirectionAttr()
Mass	UsdPhysicsMassAPI::GetMassAttr()
Restitution	UsdPhysicsMaterialAPI::GetRestitutionAttr()
Static friction	UsdPhysicsMaterialAPI::GetStaticFrictionAttr()
Dynamic friction	UsdPhysicsMaterialAPI::GetDynamicFrictionAttr()

Table 3: Physic parameters mapping with USD

In each of the following analysis, the other physic parameters of each game engine are kept unchanged from their initial values.

8.2.2.2. Consistency analysis

The two videos “Unity_scene_00-a” and “Unreal_scene_00-a” illustrate the effect of restitution (Table 4). They demonstrate that from the current parameters, the results of the simulation are similar, but there is room for improvement to have more closely resembling simulation on both engines.

Video	Parameters
Unity_scene_00-a Unreal_scene_00-a	useGravity = true mass = 1.0 restitution = 0.5 static friction = 0.6 dynamic friction = 0.6

Table 4: Test video 1

As expected, the friction parameters weren’t critical in this simulation since the ball has a very small amount of contact points with the floor.

8.2.3. Analysis with new physics parameters

8.2.3.1. Addition of new physic parameters at node level

To improves the correlation of the simulation between both engines, 3 new parameters are introduced:

- Linear damping: Defines the linear drag coefficient (rate of decrease of the linear velocity over time)
- Angular damping: Defines the angular drag coefficient (rate of decrease of the angular velocity over time)
- Collision detection mode: Defines the collision detection calculation mode. It can be discrete (calculated once a frame) or dynamic (more sub-iteration per frame)

It results in the following mapping (Table-5):

MPEG-I SD	Unity	Unreal
linearDamping	Drag	Linear damping
angularDamping	Angular drag	Angular damping
collisionDetectionMode	Collision detection mode	Use CCD (Continuous collision detection)

Table 5: Physic parameters mapping with game engines

The mapping to USD is provided in the following table:

MPEG-I SD	USD
linearDamping	UsdPhysicsDriveAPI::GetDampingAttr()
angularDamping	UsdPhysicsDriveAPI::GetDampingAttr()
collisionDetectionMode	N/A

Table 6: Physic parameters mapping with USD

The following videos illustrate the use of these 3 new parameters, with the same scene as in the previous section.

Video	Parameters
Unity_scene_01-a Unreal_scene_01-a	useGravity = true mass = 1.0 restitution = 0.5 static friction = 0.6 dynamic friction = 0.6 linear damping = 0.2 angular damping = 0.05 collisionDetectionMode = Continuous dynamic

Table 7: Test video 2

They demonstrate that the simulation is very close from an engine to another.

Same test with the restitution set to 0.8 instead of 0.5:

Video	Parameters
Unity_scene_01-b Unreal_scene_01-b	useGravity = true mass = 1.0 restitution = 0.8 static friction = 0.6 dynamic friction = 0.6 linear damping = 0.2 angular damping = 0.05 collisionDetectionMode = Continuous dynamic

Table 8: Test videos

The update of the restitution parameter value is understood by both game engines, and a very close simulation is obtained with both values.

The goal of the following test (with a new scene) is to check that the friction parameter also leads to the same simulation on both engines. In the first simulation (Unity_scene_03-a and Unreal_scene_03-a) there is no friction, and on the second simulation (Unity_scene_03-b and Unreal_scene_03-b), the dynamic friction is set to 1.0 (Table 9).

Video	Parameters
Unity_scene_03-a Unreal_scene_03-a	useGravity = true mass = 1.0 restitution = 0.3 static friction = 0.0 dynamic friction = 0.0 linear damping = 0.2 angular damping = 0.05 collisionDetectionMode = Continuous dynamic
Unity_scene_03-b Unreal_scene_03-b	useGravity = true mass = 1.0 restitution = 0.4 static friction = 0.0 dynamic friction = 1.0 linear damping = 0.2 angular damping = 0.05 collisionDetectionMode = Continuous dynamic

Table 9: Test videos

Those two simulations (03-a, and 03-b) are very close to each other.

8.2.3.2. Addition of new physic parameters at scene level

In addition to the parameters of section 2, we have experimented with further parameters, added at the scene level. The results provide a near identical simulation between the two engines.

The previous test with Unity_scene_03-a video, is used with two extra parameters defined as follow:

- Physic max frame time: Determine the interval on which the physic engine should run (in second)
- Bounce threshold: A contact with a relative velocity below this threshold will not bounce.

The mapping is the following:

MPEG-I SD	Unity	Unreal
physicMaxFrameTime	Fixed timestep	Max Physics Delta Time

MPEG-I SD	Unity	Unreal
bounceThreshold	Bounce Threshold	Bounce Threshold Velocity

Table 10: Physic parameters mapping with game engines

MPEG-I SD	USD
physicMaxFrameTime	timeCodesPerSecond
bounceThreshold	N/A

Table 11: Physic parameter mapping with USD

Firstly, a simulation is launched with `physicMaxFrameTime` set to 0.1 (i.e., 10 physic frame calculations per seconds) instead of 0.02 (Unity default) and the bounce threshold set to 2.0 (Unity default)

Video	Parameters
physic_tick_rate	useGravity = true mass = 1.0 restitution = 0.3 static friction = 0.0 dynamic friction = 0.0 linear damping = 0.2 angular damping = 0.05 collisionDetectionMode = Continuous dynamic physicMaxFrameTime = 0.1 bounceThreshold = 2.0

Table 12: Test video for physic tick rate

Secondly, a simulation is launched with a bounce threshold set to 15.0 (instead of 2.0) and reset the `physicMaxFrameTime` set to 0.02 (50 calculations per second).

bounce_threshold	useGravity = true mass = 1.0 restitution = 0.3 static friction = 0.0 dynamic friction = 0.0 linear damping = 0.2 angular damping = 0.05 collisionDetectionMode = Continuous dynamic physicMaxFrameTime = 0.02 bounceThreshold = 15.0
------------------	---

Table 13: Test video for bounce threshold.

As the videos show, there is a great impact on game engine when changing these parameters, which can lead to finer control of the targeted simulation and parameters adjustments if needed.

In addition to the `physic_tick_rate` and `bounce_threshold`, to check the gravity on each game engine, a physic simulation using the same gravity value (i.e: moon gravity simulation) is launched:

- gravity: determine the gravity for the whole scene

It results in the following mapping:

MPEG-I SD	Unity	Unreal
gravity	Gravity	Gravity

MPEG-I SD	USD
gravity	UsdPhysicScene::GetGravityDirectionAttr()

8.2.4. Proposed changes to SD physic support

8.2.4.1. Update of the General section of Interactivity (section 8.2.1 of the DAM)

Interactivity is supported at the scene level and at the node level through the definition of two extensions `MPEG_scene_interactivity` and `MPEG_node_interactivity`.

When present, the `MPEG_scene_interactivity` extension shall be included as extension to the scene object.

When present, the `MPEG_node_interactivity` extension shall be included as extension to node object.

The `MPEG_node_interactivity` extension is used to complement the interactivity defined at the scene level. One particular case is the definition of the parameters for the physics engine. That is, when an `MPEG_node_interactivity` extension contains a trigger of type `TRIGGER_COLLISION` without being referenced by a trigger of type `TRIGGER_COLLISION` at the `MPEG_scene_interactivity` extension, this node shall not be considered for collision detection and instead only be used by the physics engine.

Note: when a full physics engine will be defined, the physics parameters provided in the `MPEG_interactivity` node extension will be skipped.

8.2.4.2. Semantic update at node level

To have a closely related simulation between game engines, the following optional parameters are added at the `MPEG_node_interactivity` extension (node level):

- `collisionDetectionMode`
- `linearDamping`
- `angularDamping`

if (type == TRIGGER_COLLISION) {				
Collider	integer	M		the index of the mesh element that provides the collider geometry for the current node. .
static	boolean	M		If True, the collider is defined as a static collider.
usePhysics	boolean	M		Indicates if the object shall be considered by the physics simulation.
if (usePhysics) {				
collisionDetectionMode	Enum	0	N/A	Define the collision detection calculation mode, can be DISCRETE (0) or CONTINUOUS DYNAMIC (1)
needPreciseCollisionDetection	Boolean	O	false	If true, the physics engine should handle the collision detection more accurately by increasing the detection rate for this node.
linearDamping	Number	O	0	Define the linear drag coefficient which corresponds to the rate of decrease of the linear velocity over time. The value shall be in the range [0,1], where 0 indicates no damping and 1 would result in linear motion ceasing immediately upon collision.

if (type == TRIGGER_COLLISION) {				
angularDamping	number	O	0	Defines the angular drag coefficient which corresponds to the rate of decrease of the angular velocity over time. The value shall be in the range [0,1], where 0 indicates no damping and 1 would result in angular motion ceasing immediately upon collision.
useGravity	boolean	M		Indicates if the gravity affects the object
mass	number	M		Mass of the object in kilogram.
restitution	number	M		Provides the ratio of the final to initial relative velocity between two objects after they collide
staticFriction	number	M		Unitless static friction coefficient as defined in the Coulomb friction model. Friction is the quantity which prevents surfaces from sliding off each other. Static friction is used when the object is lying still. It will prevent the object from starting to move.
dynamicFriction	Number	M		Unitless static friction coefficient as defined in the Coulomb friction model. When a large enough force is applied to the object, a dynamic friction is used, and will attempt to slow down the object while in contact with another.
}				

Table 14: New semantic proposal at MPEG_node_interactivity level

8.2.4.3. Semantic update at scene level

To provide a near identical simulation between game engines, the following optional parameters are added at the MPEG_scene_interactivity extension level (scene level):

- enablePhysicHighPrecision
- gravity
-
- physicMaxFrameTime
- bounceThreshold

Name	Type	Usage	Default	Description
enablePhysicHighPrecision recommendedPhysicsHighPrecision	Boolean	O	false	Determines whether the application should enable a more deterministic and precise physic simulation
gravity	Number	O	-9.81	Determine the gravity for the whole scene
recommendedPhysicsFrameRate	Number	O	50	Provides the recommended frame rate at which the Physics Engine should operate.
bounceThreshold	number	O	1	A contact with a relative velocity below this threshold will not result in a bounce.
triggers	array	M	[]	Contains the definition of all the triggers used in that scene
actions	array	M	[]	Contains the definition of all the actions used in that scene
behaviors	array	M	[]	Contains the definition of all the behaviors used in that scene. A behavior is composed of a pair of (triggers, actions), control parameters of triggers and actions, a priority weight and an optional interrupt action

8.2.4.4. Update of the Processing model section of interactivity (section 8.2.3 of the DAM)

If the scene description document contains a description of physics properties based on another physics model, then that physics model shall take precedence in the processing of the scene.

Otherwise, the application shall handle a physics simulation if the usePhysics Boolean is TRUE on any of the collision trigger extensions defined at the node level. When a collision occurs between two nodes, the application should calculate the combination of the restitution, static friction and dynamic friction values based on the values provided by the collision trigger extension of the two nodes.

Chapter 9. Collected problem statements and industry needs

9.1. On the support of real environment data

Source: [m61811](#)

9.1.1. General

In Augmented Reality (AR) experiences, virtual content is seamlessly inserted into the user's real environment using optical or video-see-through devices. The knowledge of the user's real environment is then required for:

- * The positioning of the virtual objects based on AR anchors
- * Consistent handling of collisions between virtual and real objects
- * Consistent rendering of virtual and real objects including occlusion and lighting/shadowing aspects

This contribution provides an overview of how real environment data are handled (captured, computed, stored and loaded) in some AR frameworks and proposes to investigate the support of real environment data in MPEG-I Scene Description for transmission purposes.

9.1.2. Representation of the real environment

As shown in [Figure 27](#), the real environment data are computed from embedded-sensor raw data. An AR device may have several embedded sensors to scan the user environment, such as color camera(s) and Light Detection and Ranging (LiDAR). The generated raw data are typically point clouds, depth maps, pictures. An Inertial Measurement Unit (IMU) is also required to estimate the current pose of the AR device when acquiring these data. Based on these sensor raw data, a representation of the real environment is computed and the resulting real environment data may have various formats:

- A single mesh, optionally textured, issued from a spatial mapping computation
- A semantic representation, optionally associated with a mesh segmentation, issued from a scene understanding computation
- A real light mapping

Depending on the AR experiences, the most appropriate representation of the real environment is computed:

- A single mesh representation may be sufficient for coherent collision handling and lighting
- A semantic representation (e.g. “desk”, “laptop”, “screen”, “floor”, “ceiling”, “wall”) may be required for the definition of advanced anchoring and/or interaction
- A mesh segmentation is required for individual real object handling, such as object removal in a diminished reality application

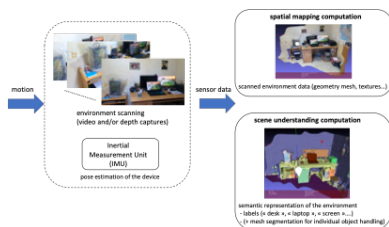


Figure 27. Computation of real environment data

The computation of the real environment data may either be done locally in the AR device or remotely in a Spatial Computing Server. In the case of remote computation, the transmission of such kind of data is in line with the Spatial Computing Server (SCS) requirements for eXtended reality (XR) of the MPEG-I Phase 2 requirement document especially the requirement #134:

“The SCS shall provide XR Spatial Description in a standard representation format (e.g. scene description) upon request of XR devices (UEs) on different platforms (desktop and mobile).”

9.1.3. Storing a representation of the real environment

The process of scanning the real environment and generating the corresponding representation may be done prior to runtime. This approach is often related to quasi-static environment and has the following main advantages:

- Availability of the real environment data at the beginning of the AR session
- Resource optimization of the AR devices resulting to power savings as no or limited scans are required at runtime
- Support of low-end AR devices having no efficient sensors
- Consistency of the representation of a shared real environment between several heterogeneous AR devices
- Ability to build a scalable library of real environments (rooms, buildings, cities...)

Note: Having an initial scan may also be relevant for time-evolving real environments. Updating some parts of the initial scan could be less time-consuming than performing a complete scan.

Generating real environment data before runtime requires efficient storage. Storing real environment data in the Cloud has been investigated by ETSI Augmented Reality Framework (ARF). As shown in Figure 28, a World Knowledge server is located in the Cloud and stores the real environment data to be used by

- a Vision Engine for AR anchoring positioning/localization aspects
- a 3D Rendering Engine for consistent collision handling and rendering between virtual and real objects

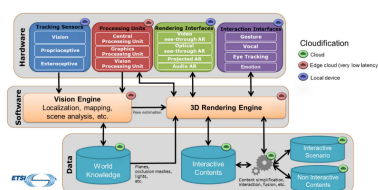


Figure 28. Global overview of the architecture of an AR system (from ETSI ARF)

Note: there is a need for a format to transmit real environment data between the World Knowledge storage server and the 3D Rendering Engine in complement to the transmission of virtual contents, which is already the scope of MPEG-I SD.

9.1.4. Examples of framework for real environment handling

Several frameworks are available to scan, compute, store and load real environment data for AR experiences. An overview of the following frameworks is provided in this section:

- Microsoft's Mixed Reality framework
- Apple's ARKit framework
- Meta/Oculus framework

9.1.4.1. Microsoft's Mixed Reality framework

The Microsoft Mixed Reality framework has been developed for the HoloLens 2 device. It is composed of

- a spatial computing module, generating a mesh representation of the real environment as shown in [Figure 29](#)
- a scene understanding module from Mixed Reality Toolkit (MRTK) version 2.7 based on OpenXR, detecting and labeling planar surfaces for the placement of virtual content as shown in [Figure 30](#)

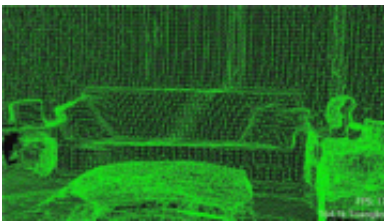


Figure 29. Mesh representation of the real environment after a spatial mapping computation



Figure 30. Semantic representation of the real environment after a scene understanding computation

A complete Microsoft's Scene Understanding SDK for Unity is available. An example of a C# code to scan, load and store real environment data based on the Scene Observer object is shown below

```

if (!SceneObserver.IsSupported())
{
    // Handle the error
}

// This call should grant the access we need.
await SceneObserver.RequestAccessAsync();

// Create Query settings for the scene update
SceneQuerySettings querySettings;

querySettings.EnableSceneObjectQuads = true;
// Requests that the scene updates quads.
querySettings.EnableSceneObjectMeshes = true;
// Requests that the scene updates watertight mesh data.
querySettings.EnableOnlyObservedSceneObjects = false;
// Do not explicitly turn off quad inference.
querySettings.EnableWorldMesh = true;
// Requests a static version of the spatial mapping mesh.
querySettings.RequestedMeshLevelOfDetail = SceneMeshLevelOfDetail.Fine; // Requests
the finest LOD of the static spatial mapping mesh

// Initialize a new Scene
Scene myScene = SceneObserver.ComputeAsync(querySettings, 10.0f).GetAwaiter()
    .GetResult();

// Create Query settings for the scene update
SceneQuerySettings querySettings;

// Compute a scene but serialized as a byte array
SceneBuffer newSceneBuffer = SceneObserver.ComputeSerializedAsync(querySettings, 10
    .0f).GetAwaiter().GetResult();

// If we want to use it immediately we can de-serialize the scene ourselves
byte[] newSceneData = new byte[newSceneBuffer.Size];
newSceneBuffer.GetData(newSceneData);
Scene mySceneDeSerialized = Scene.Deserialize(newSceneData);

// Save newSceneData for later

```

9.1.4.2. Apple's ARKit framework

On a fourth-generation iPad Pro running iPad OS 13.4 or later, Apple's ARKit uses the LiDAR Scanner to create a mesh representation of the user real environment. Then this mesh is further segmented and multiple anchors, called ARMeshAnchor, are assigned to the resulting set of segmented meshes. As shown in [Figure 31](#), a semantic labeling is performed for the real objects that ARKit can identify such as ceiling, door, floor, seat, table, wall and window labels.

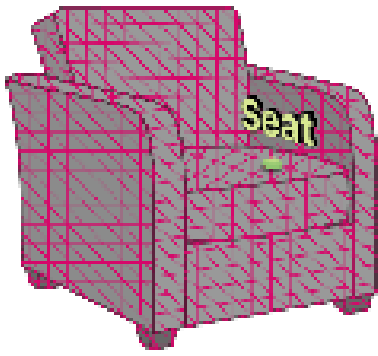


Figure 31. Semantic labeling of Apple's ARKit

These real environment data attached to the ARMeshAnchors can be saved and loaded by serializing/deserializing an ARWorldMap as shown in [Figure 32](#).

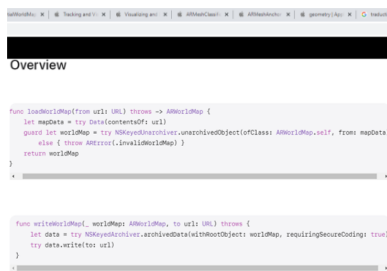


Figure 32. Saving and loading an Apple's ARKit ARWorldMap

9.1.4.3. Meta/Oculus framework

The Meta/Oculus framework has been developed for Meta Quest 2 and Meta Quest Pro devices. The scene understanding computation provides a scene model, which is a representation of the user real environment. The scene model contains Scene Anchors, with each anchor being attached to geometric components and semantic labels. The floor, ceiling, wall_face, desk, couch, door_frame and window_frame labels are currently supported as shown in [Figure 33](#).

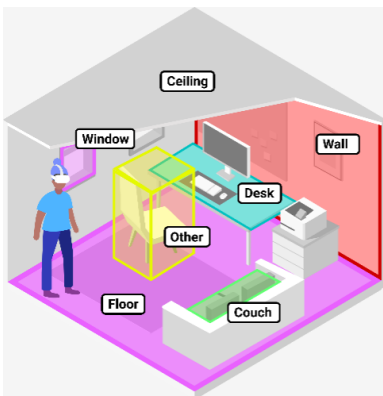


Figure 33. Semantic labeling of the Meta/Oculus Scene Understanding

The scene understanding computation is based on the Khronos OpenXR standard and relies on the Meta OpenXR XR_FB_scene extension. By using Unity as Presentation Engine, an OVRSceneManager allows access to the scene model. An OVRSceneAnchor component corresponds to a scene anchor. The semantic classification of a scene anchor is managed by the OVRSemanticClassification.

A Scene Model is generated by the Scene Capture system flow that lets users walk around and capture their scene. Users have complete control over the manual capture experience and decide

what they want to share about their environment.

As shown below, the `OVRSceneManager` provides functions

- to launch a scene capture to generate a Scene Model
- to load an existing Scene Model

```
OVRSceneManager.RequestSceneCapture()  
OVRSceneManager.LoadSceneModel()
```

9.2. Semantic representation

Source: [m64402](#)

9.2.1. Semantic Expression for 3D contents

We will divide the semantic expression for 3D contents into four criteria: the detailed attributes of objects, object (or scene)-level rendering priorities, semantic relationships between object by scene graph, and scene-level descriptions.

9.2.1.1. Detailed attributes of objects

The current glTF or other 3D format can include the color information (RGB values) or object name as attributes about objects. However, from the user's perspective, it needs to describe more detailed attributes for better understanding and interaction with a particular object (or mesh). For instance, a person object might need the emotion or situation currently experiencing, or an object like a product (e.g. wallet, chair) might need a color name, or a brand (include price).

9.2.1.2. Priority information according to object (definition of rendering order)

The current MPEG-I Scene Description (SD) does not take sufficient account of object priority within its information. Consequently, this can result in increased rendering complexity for individual objects. By incorporating rendering priority of objects into the SD object information, it would facilitate rendering based on the creator's intent. This means that even objects positioned at a greater distance within a 3D scene could be rendered first based on their importance. Furthermore, it would enable the application of rendering techniques such as super resolution and denoising to enhance the quality specifically for certain objects.

Additionally, it would provide the flexibility to selectively specify the rendering order for object classes.

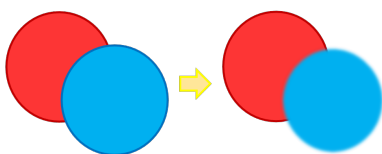


Figure 34. Example of rendering when distant objects have high priority

9.2.1.3. Semantic relationships between objects

An object is included as a lower node in MPEG-I Scene Description (SD), but there are cases where a semantic relationship is required.

For example, if there is a wallet on a desk, sub nodes of the desk might have a desk, desk legs, and a wallet. At this time, if there is no semantic relationship, the desk, desk legs, and wallet can all be separated when recreating or editing scenes. If the desk legs are separated, the meaning of the desk class becomes meaningless, so to prevent this phenomenon, the desk and the desk legs store semantic relationship information that is not separated, and the wallet has separate semantic relationship information for clear and efficient reproduction. Creation and scene editing are possible.

9.2.1.4. Scene-level descriptions

Scene-level descriptors are useful information for users who want to interact(user-experience) or edit contents. These scene-level descriptors can be defined through a descriptor neural network model. At this point, the scene graph described above may optionally be input to increase the performance of the neural network model.

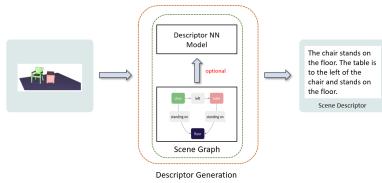


Figure 35. Example of scene-level description generation

Appendix A: Disclaimer



The formatting of the document is based on the Khronos glTF specification formatting under CC-BY 4.0.



The extensions information are automatically generated using [wetzel](#) tool under Apache License 2.0.