



Technology under Consideration for ISO/IEC 23090-14

WG3 Scene Description BoG

MDS23818_WG03_N01208

Table of Contents

1. Extensions	1
1.1. MPEG_material_acoustic	1
1.1.1. General	1
1.1.2. Semantics	1
1.1.3. Processing Model	4
1.2. The Physics glTF extension and interactivity	4
1.2.1. 1 Introduction	5
1.2.2. 2 Khronos Physics Extensions	5
1.2.3. 3 Relationship to the MPEG Interactivity Extension	6
2. ISOBMFF	7
2.1. Improvements for MPEG-I SD random access description	7
2.1.1. General	7
2.1.2. Characteristics of random access points of MPEG-I Scene Description	7
2.1.3. Description and processing of random access points	7
2.1.4. Proposed text improvements	8
2.2. On sample formats for lighting information	9
2.2.1. Introduction	9
2.2.2. Lighting information signalling	10
2.2.3. Proposals	11
3. Codec Support	20
3.1. Dynamic mesh support in scene description	20
3.2. Support for multiple atlases for MIV applications (MPEG142)	20
3.2.1. Multiple atlases	20
3.2.2. References	27
4. Interfaces	28
4.1. Supporting Multiple Viewers in the Media Access Function	28
4.1.1. General	28
4.1.2. Proposed Updates to MAF API	28
4.2. Generic API for Presentation Engine	29
4.2.1. Generic Render Control API	30
4.2.2. Extension for Audio Node Mapping	33
5. MPEG-I Audio in Scene Description	35
5.1. Immersive audio extension	35
5.1.1. Introduction	35
5.1.2. Background	35
5.1.3. MPEG-I immersive audio support	36
5.1.4. References	40
5.2. MPEG-I Audio in Scene Description	40

5.2.1. General	40
5.3. Establishing a Mapping between Audio and MPEG-I Scenes	42
5.3.1. General	42
5.3.2. Extension for Audio Node Mapping	42
5.4. On spatial synchronization between graphs	43
5.4.1. Attempt problem definition for the spatial synchronization	43
5.4.2. Approach proposal for the spatial synchronization	46
5.4.3. Conclusion	48
5.4.4. References	48
6. Reference Software	49
6.1. Thoughts on trimesh playback of AR scenes	49
6.1.1. General	49
6.1.2. AR Sessions recording and format	49
6.1.3. AR Session playback in trimesh	52
7. Interactivity framework	53
7.1. On event-based scene update	53
7.1.1. General	53
7.1.2. A use case for event based updates	54
7.1.3. JSON patch limitations	55
7.1.4. Semantics for event-based update	56
8. Collected problem statements and industry needs	58
8.1. On the support of real environment data	58
8.1.1. General	58
8.1.2. Representation of the real environment	58
8.1.3. Storing a representation of the real environment	59
8.1.4. Examples of framework for real environment handling	60
8.2. Semantic representation	63
8.2.1. Semantic Expression for 3D contents	63
8.3. The support of XR Spatial Computing of real environment	64
8.3.1. Introduction	64
8.3.2. Configuration examples of XR Spatial Computing	65
8.3.3. Approach proposal to support XR Spatial Computing	66
Appendix A: Disclaimer	68

Chapter 1. Extensions

1.1. MPEG_material_acoustic

Source: [m64377](#)

1.1.1. 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.

1.1.2. 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.
diffuseScattering	number	0.0	O	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	O	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.

Name	Type	Default	Usage	Description
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.
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.

Syntax	Length (bits)	Type	Semantics
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.

1.1.3. 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.

1.2. The Physics glTF extension and interactivity

Source: [m67814](#)

1.2.1. 1 Introduction

Khronos is currently working on an extension for rigid body physics that is expected to produce a set of KHR extensions. In this contribution, we introduce the current specification of this feature in Khronos and discuss how it can be integrated with the interactivity extensions from MPEG.

1.2.2. 2 Khronos Physics Extensions

The Khronos effort on adding support for Physics to glTF 2.0 to enable rigid body simulations has led to the development of 2 extensions: KHR_physics_rigid_bodies and KHR_collision_shapes.

The extensions to the glTF 2.0 document structure are depicted by the following figure:

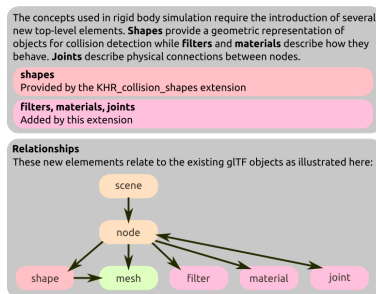


Figure 1. A black and white image of a camera Description automatically generated

The KHR_physics_rigid_bodies extensions defines properties of a rigid body, which are associated with a particular node in the graph. It may have one or more of the following properties:

- **Motion:** The motion property defines the type of motion a rigid body can have. It can be static, dynamic, or kinematic. Static bodies do not move and are not affected by forces. Dynamic bodies can move and are affected by forces. Kinematic bodies can move according to their velocity but are not affected by external forces. The motion object provides other information such as the mass, inertia vector, center of mass, linear and angular velocities of the node.
- **Collider:** The collider property describes the shape of the collider object for the purpose of collision detection. Each rigid body has exactly one collider shape, which is defined by the KHR_collision_shapes extension. In addition, the object provides pointers into physics materials, which are materials that contain physics properties that describe the collision response of that object. Finally, it also contains a set of filters, which describe if collision between two different colliders is allowed to be triggered or is ignored.
- **Trigger:** A trigger object is used to trigger application-specific behavior upon detection of a collision/overlap event. It is similar to a collider object but lacks a physics material.
- **Joint:** The joint property describes the physical connection between two rigid bodies. It defines the type of movement allowed between the bodies, such as hinge (rotation around one axis), slider (movement along one axis), or ball-and-socket (rotation around all axes). The joint property also includes descriptions of joint limits and joint drives. The restrictions for the movement of one rigid body with respect to the other are expressed by a limit object, which sets min and max values for the position/rotation along a specific axis. A joint_drive object further describes additional forces that are applied by a joint on a connected rigid body object.

As mentioned above, colliders are described through references to the KHR_collision_shapes, which is an independent extension. The extension currently supports the definition of the following

shapes: sphere, box, capsule, cylinder, convex, and trimesh.

1.2.3. 3 Relationship to the MPEG Interactivity Extension

The MPEG_scene_interactivity and MPEG_node_interactivity extensions define a collision trigger for interactivity, which relies on the implementation of a rigid body physics simulation. The use of the latter is activated through an explicit usePhysics flag. A mandatory collider mesh object is provided but it is allowed to use simplified primitives instead. It is suggested to make the collider object optional and mutually exclusive to the use of primitives.

We further suggest that the physics aspects be completely separated from the interactivity triggers as described by the interactivity extension. This will allow for the usage of the interactivity extension with any physics description model. A simple way to do that is by associating a collision or trigger event with the trigger for interactivity, e.g. through indexing. The physics properties should be completely extracted out of the interactivity extensions, which would allow for replacing the physics description in the scene graph.

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 2 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.

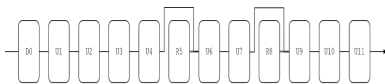


Figure 2. An example structure of scene description data with shadow sync samples

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 3. 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.

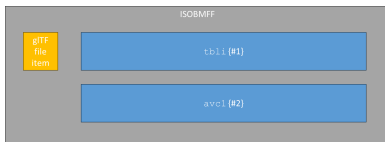


Figure 4. Carriage of texture-based light information in timed metadata track ('tbli')

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 ('tsd')

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 ('std')

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 timed accessor in the glTF to describe how the samples are formed.

```

aligned(8) class PunctionalLLightSampleEntry( )
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:
            PunctionalLLLightConfig config;
            return;
        case 1:
            TextureBasedLightingConfig config;
            return;
    }
}

```

Then the text would say that if metadata_type is equal to 0 then the sample is PunctionalLightsSample, 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

V-DMC is considered for future Amendment

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 1. MPEG_primitive_V3C Properties

	Type	Description	Required
atlases	MPEG_primitive_V3C.atlas [1-*]	An array of atlases	✓ Yes
_MPEG_V3C_CAD	MPEG_primitive_V3C._MPEG_V3C_CAD	This object lists different properties described for the Common Atlas Data in ISO/IEC 23090-5.	No
extensions	object	JSON object with extension-specific objects.	No
extras	any	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 2. MPEG_primitive_V3C._MPEG_V3C_CAD Properties

	Type	Description	Required
MIV_view_parameters	integer	indicates the accessor index which is used to refer to the list of MIV view parameters.	✓ Yes
extensions	object	JSON object with extension-specific objects.	No
extras	any	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 3. *MPEG_primitive_V3C.atlas Properties*

	Type	Description	Required
_MPEG_V3C_CONFIG	<i>integer</i>		✓ Yes
_MPEG_V3C_AD	<i>integer</i>		✓ Yes
_MPEG_V3C_GVD_MAPS	<i>integer [1-*</i>	an array of references to video texture maps.	✓ Yes
_MPEG_V3C_OVD_MAP	<i>integer [0-*</i>	a reference to a video texture that provides the occupancy map	No
_MPEG_V3C_AVD	<i>MPEG_primitive_V3C.attribute [0-*</i>		No
_MPEG_V3C_CAD	<i>object</i>	This object lists different properties described for the Common Atlas Data in ISO/IEC 23090-5.	No
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.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 4. MPEG_primitive_V3C.attribute Properties

	Type	Description	Required
type	integer	provides the type of the attribute.	No
maps	integer [1-*]		✓ Yes
extensions	object	JSON object with extension-specific objects.	No
extras	any	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-PCC), Online, <https://www.iso.org/standard/73025.html>

Chapter 4. Interfaces

4.1. Supporting Multiple Viewers in the Media Access Function

Source: [m58510](#)

4.1.1. 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.

4.1.2. 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**).

4.1.2.1. Methods

Table 5. n/a

Methods	State after success	Description
startFetching()	ACTIVE	<p>Once initialized and in READY state, the Presentation Engine may request the media pipeline to start fetching the requested data.</p> <p>The initialization may be performed using view information for one or more viewers.</p>
updateView()	ACTIVE	<p>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.</p> <p>A call to this function shall include the view information for only those views whose parameters have significantly changed.</p>

4.1.2.2. 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.      ();
};
```

4.2. Generic API for Presentation Engine


Source: [m66705](#)

4.2.1. Generic Render Control API

The Generic Render Control API is an abstract API that is offered by external renderers to enable applications, such as Presentation Engines, to control the rendering process by aligning and synchronizing their rendering state to that of 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()	<p>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.</p> <p>The inputs to this method call should be:</p> <ul style="list-style-type: none">• A media source object that contains a handler to the buffer(s), where the source media will be made available by the MAF. A description of the media source and the contents of each buffer shall also be provided.

Method	Description
configure()	<p data-bbox="427 165 1458 286">Configures the external renderer to establish an initial alignment and synchronization between the Presentation Engine and the external renderer.</p> <p data-bbox="427 324 1023 360">The parameters to this method may include:</p> <ul data-bbox="451 398 1458 1361" style="list-style-type: none"> <li data-bbox="451 398 1458 562">• A mapping between the initial timestamp of the common Presentation Engine timeline and that of the media associated with the external renderer. It also provides information about the clock rate of the Presentation Engine. <li data-bbox="451 584 1458 913">• A list of mapped nodes in the source media rendered by the external renderer. This list shall at least contain one object with a mapping to the main camera of the main scene description. For audio renderers, this may be the audio listener. The information is provided by the the MPEG_node_mapping extension in the scene description document. It should also provide the initial position and transformation of the mapped nodes after applying the transformations associated with these node mappings. . <li data-bbox="451 936 1458 1301">• A description of the scene bounding box using the glTF 2.0 spatial coordinate system. The external renderer uses this information to establish a spatial alignment between the scene coordinate system and the coordinate system that is used by the source media. The external renderer may align the bounding box of the scene to that of its media stream, which establishes the transformation that needs to be applied to all spatial coordinates exchanged over the API, in order to determine the corresponding coordinates in the coordinate system of the media stream. <li data-bbox="451 1323 1430 1361">• A list of tracked AR anchors that may be used by the external renderer. <p data-bbox="427 1400 1458 1520">The external renderer may then subscribe for updates to specific aligned nodes or it may specifically ask for current state for these nodes, using the referenceId.</p> <div data-bbox="477 1655 542 1720">  </div> <p data-bbox="620 1570 1426 1812">all exchanges over this API are based on the scene (glTF2.0) coordinate system. It is the responsibility of the external renderer to convert into their own coordinate system. The Presentation Engine does not consider any other coordinate systems other than the one established by the scene description.</p>

Method	Description
start() pause() resume() stop()	Allows the Presentation Engine to control the playback of selected media sources associated with the external renderer for interactivity purposes.
update()	<p>Used by the Presentation Engine to update node positions and orientations for which there is a mapping with the external renderer. Updates may result from received scene updates, user interactions, animations, physics simulations, or any other events.</p> <p>The parameters passed to this method are an array of objects consisting of:</p> <ul style="list-style-type: none"> • The referenceId of the node to which this update applies • The transform matrix that sets the current pose of the tracked object after applying the transform operation as described by the corresponding MPEG_node_mapping. Any further adjustments need to be applied by the external renderer to align with its internal coordinate system.
updateGraph()	<p>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. Only nodes that have a mapping with the external renderer can be passed through this method.</p> <p>The parameters to this method are an array of objects that include:</p> <ul style="list-style-type: none"> • The graph operation: ADD, REMOVE, UPDATE • For ADD: the referenceId and the initialization information for the associated media data to the object that is to be added. • For REMOVE: the referenceId of the object to be removed. • For UPDATE: the referenceId of the object to be updated, as well as a dictionary of attributes and their update values.
registerCallback()	<p>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.</p> <p>The Presentation Engine shall register a callback function whenever possible.</p>

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

```
interface GenericRenderControl {
    void init();
    void configure();
}
```

```

void start();
void pause();
void resume();
void stop();
update();
void updateGraph();
void registerCallback();
};

```

4.2.2. Extension for Audio Node Mapping

4.2.2.1. 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:

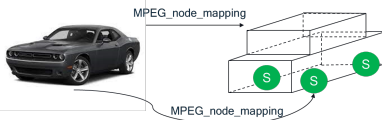


Figure 5. A black and white image of a camera Description automatically generated

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

4.2.2.2. Semantics

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

Name	Type	Default	Usage	Description
mappings	array(object)		M	An array of mappings associated with the containing node.
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 <code>MPEG_media</code> that provides the media resource that contains the mapped element.
referenceId	number	N/A	M	An identifier of the element in the referenced resource.

Name	Type	Default	Usage	Description
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.

4.2.2.3. 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 5. MPEG-I Audio in Scene Description

5.1. Immersive audio extension

Source: [m63549](#)

5.1.1. 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.

5.1.2. 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 6](#) 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

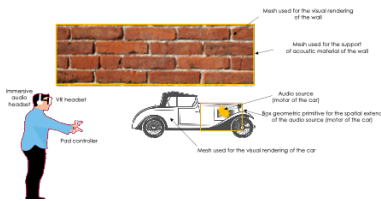


Figure 6. Virtual objects having dedicated representations for audio and visual renderers

Each object representation is dedicated to either the audio or visual renderer. For example, the geometry for the spatial extent 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.

5.1.3. 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 6](#), we describe and compare the different capabilities of MPEG_audio_spatial and the MPEG-I Audio solution.

Table 6. 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 7](#) . Note that alternatively, a single extension, as is the case with MPEG_audio_spatial, might be defined instead.

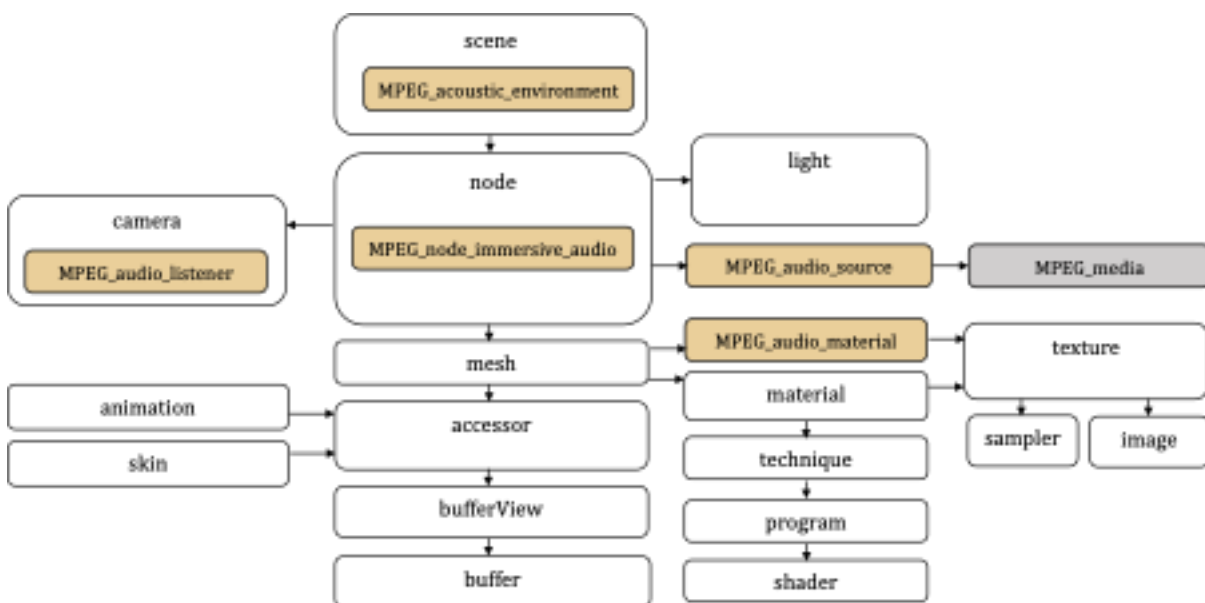


Figure 7. Proposed new MPEG glTF extensions to support MPEG-I immersive audio

5.1.3.1. 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.

5.1.3.2. 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.

5.1.3.3. 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])

The MPEG-I render pipeline is depicted by [Figure 9](#):

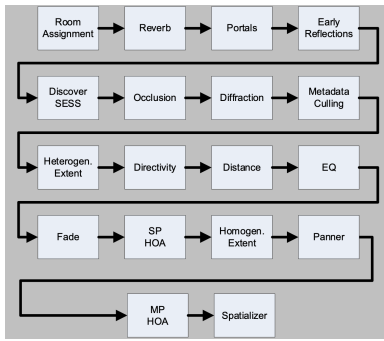


Figure 9. N/A

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 10](#):

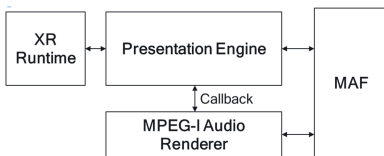


Figure 10. N/A

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 11](#):

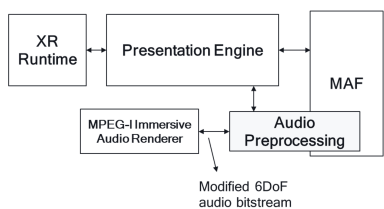


Figure 11. N/A

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.

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

Source: [m65378](#)

5.3.1. 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:

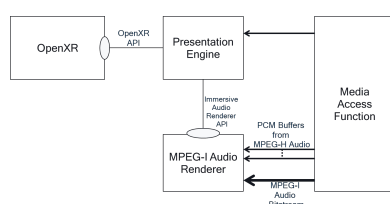


Figure 12. n/a

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.

5.3.2. Extension for Audio Node Mapping

5.3.2.1. 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:

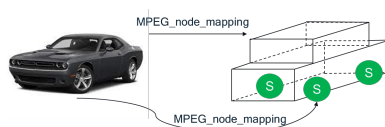


Figure 13. n/a

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

5.3.2.2. 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.

5.3.2.3. 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.

5.4. On spatial synchronization between graphs

Source: [m67011](#)

5.4.1. Attempt problem definition for the spatial synchronization

5.4.1.1. Virtual Reality (VR) use case

The VR use case corresponds to an animated virtual car. Each wheel can be animated individually. Spatial sounds are generated by the motor of the car, and by the contact of each wheel on the road.

[Figure 14](#) provides the SD and the immersive audio graph representations of the virtual car.

It can be noticed that these two graphs have not the same topology and not the same global XR Space (i.e., the global frame of reference in which 3D coordinates are expressed).

The following node mappings have been created:

- Between the root nodes of the car to ensure a consistent car animation
- Between each node related to a wheel to ensure a consistent wheel animation

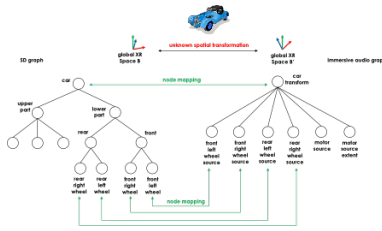


Figure 14. SD and immersive audio graph representations of a virtual car

Note 1: The node mapping needs to be investigated, when an extent is added to an audio source, to ensure the spatial synchronization of both the audio source and its extent. For example, the two following approaches may be envisaged if an extent is added to a wheel of the car:

- To allow nested spatial transformation nodes in the immersive audio graph [Figure 15](#)
 - The audio source and its extent would then be the children of a mapped spatial transformation node
- Or to allow the extent to be a child of the mapped audio source [Figure 16](#)

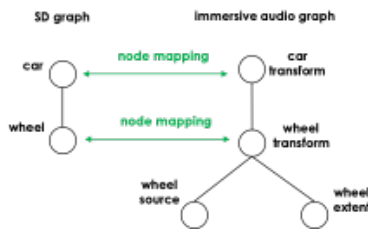


Figure 15. Possible approaches to ensure a spatial synchronization for both an audio source and its extent

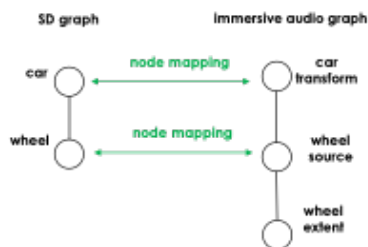


Figure 16. Possible approaches to ensure a spatial synchronization for both an audio source and its extent

The following issues need to be addressed to ensure a spatial synchronization between the two graphs:

- the knowledge of the transformation matrix between the global XR Space B and B',
- the identification of which initial parameters to be provided to the immersive audio renderer through the render control API at the configuration step,
- the identification of which parameters to be provided to the immersive audio renderer through the render control API to maintain the spatial synchronization during the VR experience.

5.4.1.2. Augmented Reality (AR) use case

In this use case, the virtual car of section 2 is inserted to the user's real environment using AR anchoring.

MPEG-I Scene Description has defined a dedicated MPEG_anchor glTF extension to support AR anchoring of virtual assets represented by a node graph.

The MPEG_anchor extension defines the Trackable and Anchor objects as follows (Figure 17):

Trackable: a real-world object that can be tracked by the XR runtime. Each trackable provides a local reference space, also known as a trackable space, in which an anchor can be expressed.

Anchor: a virtual element for which its position, orientation, scale and other properties are expressed in the trackable space defined by the trackable. A virtual asset's position, orientation, scale and other properties are expressed in relation to an anchor.

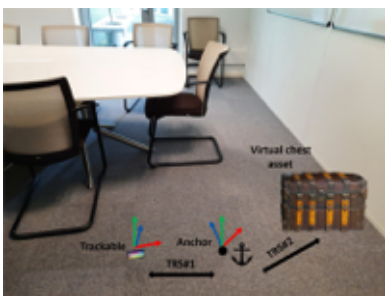


Figure 17. Trackable and Anchor for AR

In this AR use case, both the SD and the immersive audio graph may define a Trackable to insert the virtual car into the user's real environment.

Note 2: The immersive audio group uses a single Anchor object for the AR anchoring of the scene. This Anchor object corresponds to a Trackable object of an MPEG Scene Description. In other words, the transformation matrix between the Trackable and the Anchor objects (TRS#1 in Figure 17) is always the Identity matrix in the immersive audio graph.

Figure 18 illustrates the AR anchoring of the SD and immersive audio graphs representing the virtual car using a 2D marker by assuming that a common shared Trackable is defined in both the SD and immersive audio graphs.

Note 3: The root nodes of the car for the two graphs need to have identical initial transformation matrices to ensure a consistent spatial positioning with respect to the Trackable.

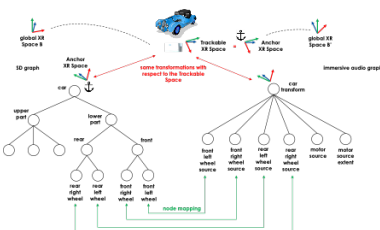


Figure 18. SD and immersive audio graph representations of a virtual car with AR anchoring using a 2D marker

The pose of the Trackable is retrieved from the XR Runtime API of the device (e.g. Khronos

OpenXR).

The XR Runtime needs to be configured through the XR Runtime API at the beginning of the AR session to be able to detect and track the Trackable at runtime.

It is assumed that the Presentation Engine related to the SD graph configures the XR Runtime. An approach would be that the poses of the Trackables are provided to the immersive audio renderer by the Presentation Engine through the render control API to ensure the spatial consistency between the two graphs.

5.4.2. Approach proposal for the spatial synchronization

This section proposes an approach to address the following issues for ensuring a spatial synchronization between the SD and the immersive audio graphs:

- the knowledge of the transformation matrix between the global XR Space B and B',
- the identification of which initial parameters to be provided to the immersive audio renderer through the render control API at the configuration step,
- the identification of which parameters to be provided to the immersive audio renderer through the render control API to maintain the spatial synchronization during the VR experience.

For the AR case, it is assumed that the Presentation Engine related to the SD graph configures the XR Runtime. Then, the poses of the Trackables are provided to the immersive audio renderer by the Presentation Engine through the render control API.

5.4.2.1. Determination the transformation matrix between the global XR Space of each graph

This spatial transformation corresponds to the matrix $P_{B'}^B$ which transforms the input 3D coordinates expressed in the global XR Space B of the SD graph to 3D coordinates expressed in the global Space B' of the immersive audio graph (1):

$$(x', y', z')_{B'} = P_{B'}^B (x, y, z)_B \quad (1)$$

The proposed approach uses the node mappings between the two graphs to obtain a common XR Space from which the calculation of this matrix $P_{B'}^B$ can be done.

Figure 19 illustrates this matrix calculation process with:

- The node *ref* of the SD graph used as the node mapping of reference, defining a local XR Space B_{ref} and a mapping transform matrix $P_{B_{ref}}^{B_{ref}}$ (i.e., the transform parameter of the node mapping glTF extension of [1])
- The node *ref'* of the immersive audio graph referenced by the *referenceId* parameter of the node mapping glTF extension of [1]

global XR Space B of the SD graph, the immersive audio renderer can convert these poses to the global XR Space B' of the immersive audio graph by using the formula (1).

5.4.2.3. Parameters to be provided to the immersive audio renderer to maintain the spatial synchronization

The spatial synchronization between the two graphs is maintained by providing the current poses of the mapped nodes and the Trackables expressed in the global XR Space B of the SD graph. Then, the immersive audio renderer can convert these poses to the global XR Space B' of the immersive audio graph by using the formula (1).

5.4.3. Conclusion

We propose to discuss on the content of the sections 2 and 3 with the immersive audio experts. If the proposed approach is agreeable, we propose to add the content of sections 3 to the TuC for further investigations.

5.4.4. References

[1] MPEG-I WG3 m66705, generic API for Presentation Engine, January 2024

Chapter 6. Reference Software

6.1. Thoughts on trimesh playback of AR scenes

Source: [m60282](#)

6.1.1. 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)

6.1.2. AR Sessions recording and format

6.1.2.1. 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 20](#) available in the [ARCore documentation](#).

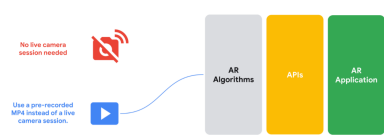


Figure 20. AR Session playback in ARCore

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

6.1.2.2. 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/>

6.1.3. AR Session playback in trimesh

As presented in clause [Section 6.1.2](#), 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 7. Interactivity framework

7.1. On event-based scene update

Source: [m61812](#)

7.1.1. 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.

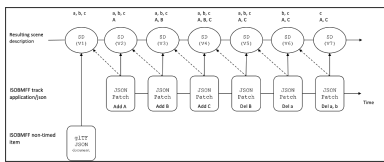


Figure 21. n/a

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.

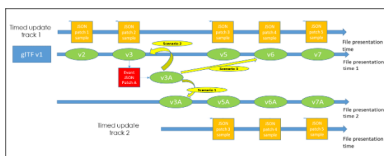


Figure 22. n/a

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.

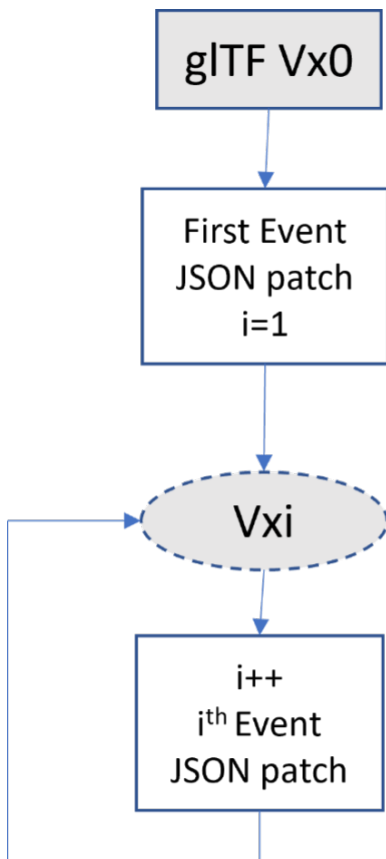


Figure 23. Event-based update diagram

7.1.2. A use case for event based updates

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

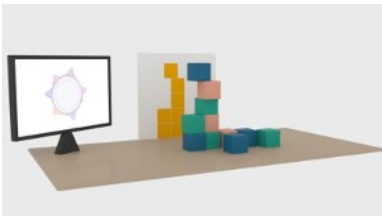


Figure 24. n/a



Figure 25. 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.

7.1.3. 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 7.1.2](#), 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...).

7.1.4. 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 7.1.2](#).

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 7. 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...), ...

Chapter 8. Collected problem statements and industry needs

8.1. On the support of real environment data

Source: [m61811](#)

8.1.1. 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.

8.1.2. Representation of the real environment

As shown in [Figure 26](#), 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

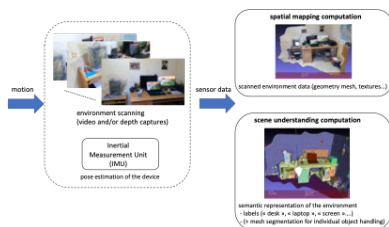


Figure 26. Computation of real environment data

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).”

8.1.3. 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 27, 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

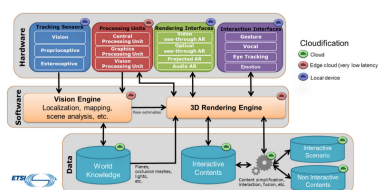


Figure 27. Global overview of the architecture of an AR system (from ETSI ARF)

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.

8.1.4. 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

8.1.4.1. 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 28](#)
- 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 29](#)

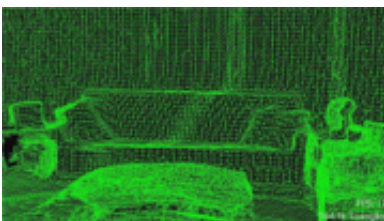


Figure 28. Mesh representation of the real environment after a spatial mapping computation



Figure 29. 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

```

8.1.4.2. 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 30](#), a semantic labeling is performed for the real objects that ARKit can identify such as ceiling, door, floor, seat, table, wall and window labels.

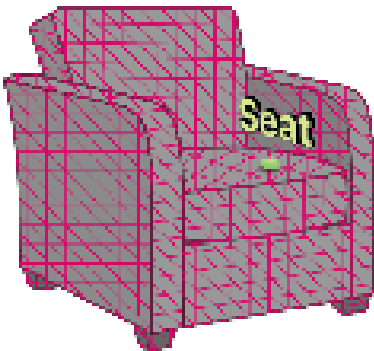


Figure 30. Semantic labeling of Apple's ARKit

These real environment data attached to the ARMeshAnchors can be saved and loaded by

serializing/deserializing an ARWorldMap as shown in [Figure 31](#).

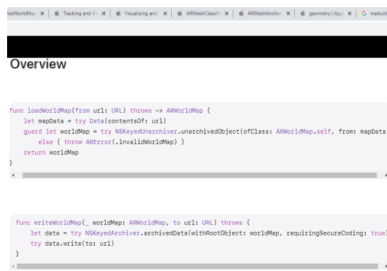


Figure 31. Saving and loading an Apple's ARKit ARWorldMap

8.1.4.3. 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 32](#).

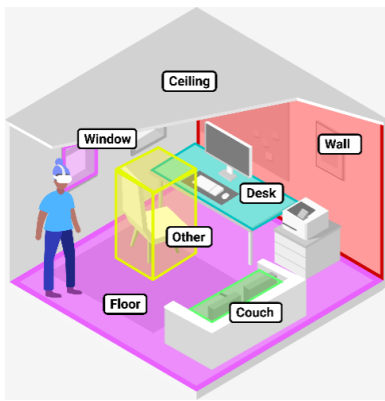


Figure 32. Semantic labeling of the Meta/Oculus Scene Understanding

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()
```

8.2. Semantic representation

Source: [m64402](#)

8.2.1. 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.

8.2.1.1. 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).

8.2.1.2. 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.

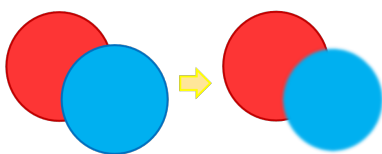


Figure 33. Example of rendering when distant objects have high priority

8.2.1.3. 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.

8.2.1.4. 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.

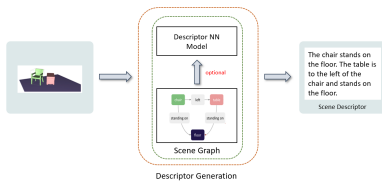


Figure 34. Example of scene-level description generation

8.3. The support of XR Spatial Computing of real environment

Source: [m67595](#)

8.3.1. Introduction

In Augmented Reality (AR) experiences, virtual content is inserted into the user real environment.

As described in MDS23494_WG03_N01127 Section 9.1 ([6]), the knowledge of the user real environment may be used for:

- the positioning of the virtual objects based on AR anchors,
- the consistent handling of collisions between virtual and real objects,
- the consistent rendering of virtual and real objects including occlusion and lighting/shadowing aspects.

An entity, commonly called *XR Spatial Computing*, computes an appropriate representation of the real environment (e.g., single mesh, segmented and labeled meshes) for the AR experience. This representation, commonly called *XR Spatial Description*, is then used by the Presentation Engine for the insertion and the rendering of virtual content into the user real environment.

In the context of MPEG-I Scene Description, a content creator may have the possibility to configure the XR Spatial Computing to get an XR Spatial Description that is best suited for that AR experience, i.e. it allows the best integration of the virtual scene provided in the scene description file.

The following aspects need to be addressed for the proposed study:

- the configuration of the XR Spatial Computing to generate the appropriate XR Spatial Description,
- the retrieval of the XR Spatial Description.

This contribution provides some configuration examples of the XR Spatial Computing (section 2) and proposes a tentative approach for the proposed study on the support of the XR Spatial Computing in Scene Description during the phase 3 (section 3).

8.3.2. Configuration examples of XR Spatial Computing

Recent AR devices (Apple Vision Pro, Meta Quest 3) compute a XR Spatial Description for AR experiences.

The configuration of the XR Spatial Computing and the retrieval of the generated XR Spatial Description are provided through XR Runtime APIs (e.g., Khronos OpenXR API with dedicated vendor extensions or proprietary API).

For example, Apple ARKit/RealityKit provides the following configuration options for its “scene reconstruction and understanding” API [2]:

ARView.Environment.SceneUnderstanding.Options

static let collision: ARView.Environment.SceneUnderstanding.Options The .collision option means that the reconstructed geometry can be used for collision queries (i.e. raycasting)

static let **default**: ARView.Environment.SceneUnderstanding.Options

The .default options is a sentinel value that indicates the user wants whatever scene understanding features work with the current device and are supported. It overrides other options in the options set. static let occlusion: ARView.Environment.SceneUnderstanding.Options The .occlusion option means that the reconstructed geometry will be used for rendering, but only to update the depth buffer. Parts of virtual objects which are behind the reconstructed geometry are not rendered. static let physics: ARView.Environment.SceneUnderstanding.Options

No abstract static let receivesLighting: ARView.Environment.SceneUnderstanding.Options The .receivesLighting option means that the virtual lights will interact with real world surfaces causing them to shine. The properties of the mesh will be set to a default material.

The .receivesLighting option means that the virtual lights will interact with real world surfaces causing them to shine. The properties of the mesh will be set to a default material.

In another example, Microsoft provides the following configuration options in its [XR_MSFT_scene_understanding](#) [3] Khronos OpenXR vendor extension:

```
XrSceneComputeFeatureMSFT:
typedef enum XrSceneComputeFeatureMSFT {
    XR_SCENE_COMPUTE_FEATURE_PLANE_MSFT = 1,
    XR_SCENE_COMPUTE_FEATURE_PLANE_MESH_MSFT = 2,
    XR_SCENE_COMPUTE_FEATURE_VISUAL_MESH_MSFT = 3,
    XR_SCENE_COMPUTE_FEATURE_COLLIDER_MESH_MSFT = 4,
    // Provided by XR_MSFT_scene_understanding_serialization
    XR_SCENE_COMPUTE_FEATURE_SERIALIZE_SCENE_MSFT = 1000098000,
    // Provided by XR_MSFT_scene_marker
    XR_SCENE_COMPUTE_FEATURE_MARKER_MSFT = 1000147000,
    XR_SCENE_COMPUTE_FEATURE_MAX_ENUM_MSFT = 0x7FFFFFFF
} XrSceneComputeFeatureMSFT;
```

In the case of the computation of segmented meshes, the XR Spatial Computing is capable of

providing a classification.

For example, the Apple ARKit ARMeshClassification API [5] provides the following classification:

```
case ceiling : The face is a part of a real-world ceiling.
case door : The face is a part of a real-world door.
case floor : The face is a part of a real-world floor.
case none : A face ARKit can't classify.
case seat : The face is a part of a real-world seat.
case table : The face is a part of a real-world table.
case wall : The face is a part of a real-world wall.
case window : The face is a part of a real-world window.
```

In another example, Microsoft the https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XR_MSFT_scene_understanding [XR_MSFT_scene_understanding] extension [4] provides the following classification:

```
XrSceneObjectTypeMSFT
* XR_SCENE_OBJECT_TYPE_UNCATEGORIZED_MSFT
* XR_SCENE_OBJECT_TYPE_BACKGROUND_MSFT
* XR_SCENE_OBJECT_TYPE_WALL_MSFT
* XR_SCENE_OBJECT_TYPE_FLOOR_MSFT
* XR_SCENE_OBJECT_TYPE_CEILING_MSFT
* XR_SCENE_OBJECT_TYPE_PLATFORM_MSFT
* XR_SCENE_OBJECT_TYPE_INFERRED_MSFT
```

In another example, Meta provides the following classification for its Quest 3 VR headset, mapped with the AR Foundation labels in Unity :

Meta Label	AR Foundation Label
DESK	Table
COUCH	Seat
FLOOR	Floor
CEILING	Ceiling
WALL_FACE	Wall
DOOR_FRAME	Door
WINDOW_FRAME	Window
SCREEN	Other
LAMP	Other
PLANT	Other
STORAGE	Other
BED	Other
OTHER	Other

8.3.3. Approach proposal to support XR Spatial Computing

The two following aspects need to be addressed to support XR Spatial Computing:

- the configuration of the XR Spatial Computing to generate the appropriate XR Spatial Description,
- the retrieval of the XR Spatial Description.

The parameters for the configuration of the XR Spatial Computing may be provided within dedicated MPEG glTF extension(s) in the Scene Description graph.

The XR Spatial Computing may handle the real objects within its own real scene representation (e.g., a real scene graph). This real scene representation may be time-evolving.

Therefore, a Spatial Computing (SC) API may be defined for the configuration and the retrieval of the XR Spatial description.

In that sense, this approach may be similar to the one addressing the need of time and spatial synchronizations between the Scene Description graph managed by a Presentation Engine and another graph managed by an external renderer (e.g. an immersive audio renderer)[1].

A high-level architecture corresponding to the proposed approach is provided below:

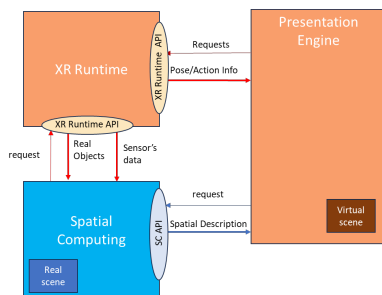


Figure 35. Proposed high-level architecture for the XR Spatial Computing support

[1] [MPEG-I WG3 m66705, generic API for Presentation Engine, January 2024](#)

[2] [Apple Scene Understanding API](#)

[3] [Microsoft scene understanding OpenXR extension: compute options](#)

[4] [Microsoft scene understanding OpenXR extension: object types](#)

[5] [Apple ARKit mesh classification](#)

[6] [MPEG-I Part 14 Scene Description Technology Under Considerations \(TuC\), MDS23494_WG03_N01127](#)

[7] [Unity Meta OpenXR platform: plane detection](#)

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.