



Technology under Consideration for ISO/IEC 23090-14

WG3 Scene Description BoG

MDS22652_WG03_N00909

Table of Contents

1. Extensions	1
1.1. MPEG_media	1
1.1.1. General	1
1.1.2. MPEG_media	1
1.1.3. MPEG_media.media	2
1.1.4. MPEG_media.media.controls	5
1.2. MPEG_audio_spatial	6
1.2.1. General	6
1.2.2. Semantics	6
1.3. MPEG_camera_control	11
1.3.1. General	11
1.3.2. Semantics	12
1.3.3. Processing Model	14
1.3.4. Example	14
1.4. MPEG_buffer_circular	15
1.4.1. General	15
1.4.2. MPEG_buffer_circular	15
1.5. MPEG_node_avatar	17
1.5.1. General	17
1.5.2. Semantics	17
1.5.3. Example	21
1.6. Shadow Scenes	22
2. ISOBMFF	23
2.1. Carriage Format for animation timing	23
2.1.1. Multiple animations	23
2.1.2. Interaction of animation and dynamic 3D object	24
2.2. Improvements for MPEG-I SD random access description	26
2.2.1. General	26
2.2.2. Characteristics of random access points of MPEG-I Scene Description	26
2.2.3. Description and processing of random access points	26
2.2.4. Proposed text improvements	28
3. Codec Support	29
3.1. Clarification of type of V-PCC track referenced from MPEG_media	29
3.1.1. Consideration	29
3.1.2. Proposal	29
3.2. Dynamic mesh support in scene description	30
3.2.1. Introduction	30
3.2.2. Design	30

3.2.3. Assets and Implementation	30
3.3. Support for multiple atlases for MIV applications (MPEG142)	31
3.3.1. Multiple atlases	31
3.3.2. References	38
3.4. Immersive audio extension	39
3.4.1. Introduction	39
3.4.2. Background	39
3.4.3. MPEG-I immersive audio support	40
3.4.4. References	44
3.5. On G-PCC support	44
3.5.1. Consideration on in-GPU processing	44
3.5.2. Proposal	45
3.5.3. Reference	46
3.5.4. Annex. Proposed MPEG extension	46
4. Data Formats	50
4.1. Support of glTF CBOR binary format	50
4.1.1. Problem Statement	50
4.1.2. Benefit of CBOR file/data format:	50
4.1.3. CBOR data size comparison example:	50
4.1.4. Use Cases	50
4.1.5. Potential Solutions	51
4.1.6. Open Issue Discussion	52
5. Interfaces	53
5.1. On DASH Dynamic Bitrate Adaption with Viewpoint Update	53
5.1.1. Problem Statement	53
5.1.2. Use Cases	53
5.1.3. Current Scene Description Support and Gasps	54
5.2. Supporting Multiple Viewers in the Media Access Function	55
5.2.1. General	55
5.2.2. Proposed Updates to MAF API	56
5.3. CoAP API support in MAF	57
5.3.1. General	57
5.3.2. MAF as CoAP Client	57
5.3.3. MAF as HTTP-CoAP Proxy	57
6. MPEG-I Audio in Scene Description	58
6.1. MPEG-I Audio in Scene Description	58
6.1.1. General	58
7. Reference Software	60
7.1. Thoughts on trimesh playback of AR scenes	60
7.1.1. General	60
7.1.2. AR Sessions recording and format	60

7.1.3. AR Session playback in trimesh.	64
8. Interactivity framework.	65
8.1. On event-based scene update.	65
8.1.1. General.	65
8.1.2. A use case for event based updates.	66
8.1.3. JSON patch limitations.	67
8.1.4. Semantics for event-based update.	68
9. Collected problem statements and industry needs.	70
9.1. On the support of real environment data.	70
9.1.1. General.	70
9.1.2. Representation of the real environment.	70
9.1.3. Storing a representation of the real environment.	71
9.1.4. Examples of framework for real environment handling.	72
Appendix A: JSON Schema for extensions.	76
A.1. JSON Schema for MPEG_buffer_circular extension.	76
A.2. JSON Schema for MPEG_media.	76
A.3. JSON Schema for MPEG_media.media.	78
A.4. JSON Schema for MPEG_media.media.controls.	81
A.5. JSON Schema for MPEG_node_transformation_external.	82
A.6. JSON Schema for MPEG_buffer_circular.	83
A.7. JSON Schema for MPEG_audio_spatial.source.	84
A.8. JSON Schema for MPEG_audio_spatial.source.cluster.	87
A.9. JSON Schema for MPEG_node_avatar.metadata object.	88
A.10. JSON Schema for MPEG_node_avatar_representation extension.	89
A.11. JSON Schema for MPEG_primitive_V3C.	91
A.12. JSON Schema for MPEG_primitive_V3C._MPEG_V3C_CAD.	93
A.13. JSON Schema for MPEG_primitive_V3C.atlas.	94
A.14. JSON Schema for MPEG_primitive_V3C.attribute.	96
Appendix B: Disclaimer.	97

Chapter 1. Extensions

1.1. MPEG_media

Source: [m56047](#)

1.1.1. General

It is proposed to support signaling more detailed playback control information about the MPEG media in MPEG_media extension.

Currently in MPEG_media extension a boolean “controls” is signalled which has the semantics that it “specifies that media controls should be displayed (such as a play/pause button etc)”. It is asserted that for MPEG-I scene description, a more detailed information should be allowed to be signaled in the MPEG-media extension to specify the supported playback control for the MPEG media.

For example, it is asserted that currently it is unspecified if the MPEG media referred by the MPEG_media extension is allowed to be fast forwarded or fast backwarded. It is asserted that the support for this should be allowed to be specified under content creator discretion (e.g. a game show broadcast may not allow fast backward). Similarly, certain content may be allowed to be paused, whereas other type of content may not be allowed to be paused.

1.1.2. MPEG_media

MPEG media used to create a texture, audio source or other objects in the scene.

Table 1. MPEG_media Properties

	Type	Description	Required
media	<code>MPEG_media.media [1-*</code>	An array of MPEG media. A MPEG media contains data referred by other object in a scene	✓ Yes
extensions	<code>object</code>	JSON object with extension-specific objects.	No
extras	<code>any</code>	Application-specific data.	No

Additional properties are allowed.

- JSON schema: `MPEG_media.schema.json`

1.1.2.1. MPEG_media.media

An array of MPEG media. A MPEG media contains data referred by other object in a scene

- **Type:** `MPEG_media.media` [1-*]
- **Required:** ✓ Yes

1.1.2.2. MPEG_media.extensions

JSON object with extension-specific objects.

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

1.1.2.3. MPEG_media.extras

Application-specific data.

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

1.1.3. MPEG_media.media

MPEG media used to create a texture, audio source, or any other media type defined by MPEG.

Table 2. `MPEG_media.media` Properties

	Type	Description	Required
name	<code>any</code>		No
startTime	<code>number</code>	The startTime gives the time at which the rendering of the timed texture will be in seconds.	No, default: <code>0</code>
startTimeOffset	<code>number</code>	The startTimeOffset indicates the time offset into the source, starting from which the timed texture is generated.	No, default: <code>0</code>

	Type	Description	Required
endTimeOffset	number	The endTimeOffset indicates the time offset into the source, up to which the timed texture is generated. The value is provided in seconds, where 0 corresponds to the start of the source.	No
autoplay	boolean	Specifies that the MPEG media start playing as soon as it is ready.	No
autoplayGroup	boolean	Specifies that playback starts simultaneously for all media sources with the autoplay flag set to true.	No
loop	boolean	Specifies that the MPEG media start over again, every time it is finished.	No, default: false
controls	MPEG_media.media.controls	Specifies that which MPEG media controls should be exposed to end user	No
alternatives	array[1-*]	An array of alternatives of the same media (e.g. different video code used)	No

Additional properties are allowed.

- **JSON schema:** MPEG_media.media.schema.json

1.1.3.1. MPEG_media.media.name

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

1.1.3.2. MPEG_media.media.startTime

The startTime gives the time at which the rendering of the timed texture will be in seconds.

- **Type:** number
- **Required:** No, default: 0

- **Minimum:** `>= 0`

1.1.3.3. MPEG_media.media.startTimeOffset

The startTimeOffset indicates the time offset into the source, starting from which the timed texture is generated.

- **Type:** `number`
- **Required:** No, default: `0`
- **Minimum:** `>= 0`

1.1.3.4. MPEG_media.media.endTimeOffset

The endTimeOffset indicates the time offset into the source, up to which the timed texture is generated. The value is provided in seconds, where 0 corresponds to the start of the source.

- **Type:** `number`
- **Required:** No
- **Minimum:** `>= 0`

1.1.3.5. MPEG_media.media.autoplay

Specifies that the MPEG media start playing as soon as it is ready.

- **Type:** `boolean`
- **Required:** No

1.1.3.6. MPEG_media.media.autoplayGroup

Specifies that playback starts simultaneously for all media sources with the autoplay flag set to true.

- **Type:** `boolean`
- **Required:** No

1.1.3.7. MPEG_media.media.loop

Specifies that the MPEG media start over again, every time it is finished.

- **Type:** `boolean`
- **Required:** No, default: `false`

1.1.3.8. MPEG_media.media.controls

Specifies that which MPEG media controls should be exposed to end user

- **Type:** `MPEG_media.media.controls`
- **Required:** No

1.1.3.9. MPEG_media.media.alternatives

An array of alternatives of the same media (e.g. different video code used)

- **Type:** `array[1-*]`
- **Required:** No

1.1.4. MPEG_media.media.controls

Specifies that which MPEG media controls should be exposed to end user

Table 3. MPEG_media.media.controls Properties

	Type	Description	Required
pauseSupported	<code>boolean</code>	Pause control displayed for the MPEG media.	No, default: <code>true</code>
fastForwardSupported	<code>boolean</code>	Fast forward control displayed for the MPEG media.	No, default: <code>true</code>
fastBackwardSupported	<code>boolean</code>	Fast backward control displayed for the MPEG media.	No, default: <code>true</code>

Additional properties are allowed.

- **JSON schema:** `MPEG_media.media.controls.schema.json`

1.1.4.1. MPEG_media.media.controls.pauseSupported

Pause control displayed for the MPEG media.

- **Type:** `boolean`
- **Required:** No, default: `true`

1.1.4.2. MPEG_media.media.controls.fastForwardSupported

Fast forward control displayed for the MPEG media.

- **Type:** `boolean`
- **Required:** No, default: `true`

1.1.4.3. MPEG_media.media.controls.fastBackwardSupported

Fast backward control displayed for the MPEG media.

- **Type:** `boolean`
- **Required:** No, default: `true`

1.2. MPEG_audio_spatial

Source: [m61267](#)

1.2.1. General

The MPEG audio extension adds support for spatial audio. This extension is identified by MPEG_audio_spatial, which may be included at top level or attached to any node in the scene. When present, the MPEG_audio_spatial extension shall be included as extension of a camera object or a node object defined in ISO/IEC DIS 12113:2021.

The MPEG_audio_spatial extension supports three different node types:

- source: an audio source that provides input audio data into the scene. Mono objects and HOA sources (as defined in Annex F.1 of ISO/IEC 23008-3:2020) are supported in this version of the document.
- Type: 'Object' or 'HOA'
- HOA audio sources shall ignore the parent node's position and be rendered only in 3DoF.
- **Moreover, an audio source may be a cluster audio source, which denotes a pre-mixed representation of a selection of audio sources. To indicate a cluster audio source, the isCluster flag should be set equal to True.**
- Reverb: A reverb effect can be attached to the output of an audio source. Several reverb units can exist and sound sources can feed into one or more of these reverb units. An audio renderer that does not support reverb shall ignore it if the bypass attribute is set to true. If the bypass attribute is set to false, the audio renderer shall return an error message listener: An audio listener represents the output of audio in the scene. A listener should be attached to a camera node in the scene. By being a child node of the camera, additional transformations can be applied to the audio listener relative to the transformation applied to the parent camera.

Figure 1 depicts the processing chain for audio in a scene.

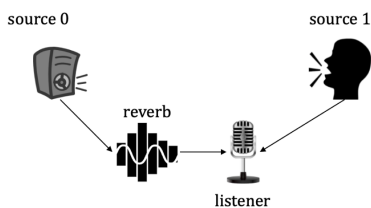


Figure 1. An example of the processing chain for audio in a scene

The specification of any audio effect processing is outside the scope of this document. The characteristics of a listener depend on the actual output devices available to the audio renderer.

1.2.2. Semantics

1.2.2.1. MPEG_audio_spatial.source

Table 4. MPEG_audio_spatial.source Properties

	Type	Description	Required
id	integer	A unique identifier	✓ Yes
type	string	A type of the audio source	✓ Yes
pregain	number	A level-adjustment of the audio source	No, default: 0
playbackSpeed	number	Playback speed of the audio source	No, default: 1
attenuation	any	A function used to calculate the attenuation of the audio source.	No, default: <code>linearDistance</code>
attenuationParameters	number [1-*	An array of attenuation parameters	No
referenceDistance	number	A distance in meters.	No, default: 1
accessors	integer [1-*	An array of accessors that describe the audio source	✓ Yes
reverbFeed	integer []	An array of pointers to reverb units	No
reverbFeedGain	number []	An array of gain values	No
isCluster	boolean	Indicates whether the audio source is a cluster.	✓ Yes
clusterProperties	MPEG_audio_spatial.source.cluster		No
extensions	object	JSON object with extension-specific objects.	No
extras	any	Application-specific data.	No

Additional properties are allowed.

- **JSON schema:** `MPEG_audio_spatial.source.schema.json`

1.2.2.1.1. MPEG_audio_spatial.source.id

A unique identifier of the audio source in the scene.

- **Type:** integer

- **Required:** ✓ Yes
- **Minimum:** ≥ 0

1.2.2.1.2. MPEG_audio_spatial.source.type

Indicates the type of the audio source. **type** value equal to **Object** indicates mono object. **type** value equal to **HOA** indicates HOA object

- **Type:** **string**
- **Required:** ✓ Yes
- **Allowed values:**
 - "Object"
 - "HOA"

1.2.2.1.3. MPEG_audio_spatial.source.pregain

Provides a level-adjustment in dB for the signal associated with the source.

- **Type:** **number**
- **Required:** No, default: 0
- **Minimum:** ≥ 0

1.2.2.1.4. MPEG_audio_spatial.source.playbackSpeed

Playback speed of the audio signal. A value of 1.0 corresponds to playback at normal speed. The value shall be between 0.5 and 2.0.

- **Type:** **number**
- **Required:** No, default: 1
- **Minimum:** ≥ 0.5
- **Maximum:** ≤ 2

1.2.2.1.5. MPEG_audio_spatial.source.attenuation

Indicates the function used to calculate the attenuation of the audio signal based on the distance to the source. attenuation value equal to **noAttenuation** indicates no attenuation function should be used. attenuation value equal to **inverseDistance** indicates inverse distance function should be used. attenuation value equal to **linearDistance** indicates linear distance function should be used. attenuation value equal to **exponentialDistance** indicates exponential distance function should be used. attenuation value equal to **custom** indicates custom function should be used. The definition of **custom** function is outside of the scope of ISO/IEC 23090-14. The attenuation functions and their parameters are defined in ISO/IEC 23090-14:Annex D.

- **Type:** **any**
- **Required:** No, default: **linearDistance**
- **Allowed values:**

- `noAttenuation`
- `inverseDistance`
- `linearDistance`
- `exponentialDistance`
- `custom`

1.2.2.1.6. MPEG_audio_spatial.source.attenuationParameters

An array of parameters that are input to the attenuation function. The semantics of these parameters depend on the attenuation function itself and are defined in ISO/IEC 23090-14

- **Type:** `number` [1-*]
- **Required:** No

1.2.2.1.7. MPEG_audio_spatial.source.referenceDistance

Provides the distance in meters for which the distance gain is implicitly included in the source signal after application of pregain. When type equals 'HOA' the element shall not be present.

- **Type:** `number`
- **Required:** No, default: `1`
- **Minimum:** `>= 1`

1.2.2.1.8. MPEG_audio_spatial.source.accessors

An array of accessor references, by specifying the accessors indices in accessors array, that describe the buffers where the decoded audio will be made available.

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

1.2.2.1.9. MPEG_audio_spatial.source.reverbFeed

Provides one or more pointers to reverb units, optionally extended by a floating point scaling factor. A reverb unit represents a reverberation audio processor that is configured by the metadata from a single reverb object. Typically, a reverb object represents reverberation properties of a single room.

- **Type:** `integer` []
- **Required:** No

1.2.2.1.10. MPEG_audio_spatial.source.reverbFeedGain

Provides an array of gain [dB] values to be applied to the source's signal(s) when feeding it to the corresponding reverbFeed. The array shall have the same number of elements as the reverbFeed array field.

- **Type:** `number []`
- **Required:** No

1.2.2.1.11. MPEG_audio_spatial.source.isCluster

Specifies if the audio source is a pre-mixed representation of a selection of audio sources.

- **Type:** `boolean`
- **Required:** ✓ Yes

1.2.2.1.12. MPEG_audio_spatial.source.clusterProperties

- **Type:** `MPEG_audio_spatial.source.cluster`
- **Required:** No

1.2.2.1.13. MPEG_audio_spatial.source.extensions

JSON object with extension-specific objects.

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

1.2.2.1.14. MPEG_audio_spatial.source.extras

Application-specific data.

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

1.2.2.2. MPEG_audio_spatial.source.cluster

Table 5. `MPEG_audio_spatial.source.cluster` Properties

	Type	Description	Required
sourceId	<code>integer [1-*</code>	An array of audio source ids.	✓ Yes
radius	<code>number</code>	A cluster radius.	✓ Yes
extensions	<code>object</code>	JSON object with extension-specific objects.	No
extras	<code>any</code>	Application-specific data.	No

Additional properties are allowed.

- **JSON schema:** `MPEG_audio_spatial.source.cluster.schema.json`

1.2.2.2.1. MPEG_audio_spatial.source.cluster.sourceId

An array of integers that contains the unique identifiers of the audio sources contained in this cluster.

- **Type:** `integer [1-*)`
- **Required:** ✓ Yes

1.2.2.2.2. MPEG_audio_spatial.source.cluster.radius

A distance in meters used to encompass the audio sources contained in this cluster.

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

1.2.2.2.3. MPEG_audio_spatial.source.cluster.extensions

JSON object with extension-specific objects.

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

1.2.2.2.4. MPEG_audio_spatial.source.cluster.extras

Application-specific data.

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

1.3. MPEG_camera_control

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

1.3.1. General

The scene description may describe a set of paths through which the camera is allowed to move. The paths may be described as a set of anchor points that are connected through path segments. For enhanced expressiveness of the camera control, each path segment may be enhanced with a bounding volume that allows some freedom in motion along the path. The [Figure 2](#) depicts this behavior.

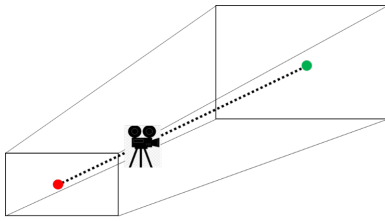


Figure 2. Example of Camera Path Segment with Bounding Volume

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

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

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

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

1.3.2. Semantics

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



*TODO : auto generate the semantics
schema is needed*

	Type	Description	Required
anchors	<code>number</code>	Number of anchor points in the camera paths.	No

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

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

The camera control information is structured as follows:

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

1.3.3. Processing Model

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

1.3.4. Example



TODO : add example

Input needed

1.4. MPEG_buffer_circular

Source: [m58186](#)

1.4.1. General

The definition of the MPEG_buffer_circular extension includes properties such as source and tracks to refer to an index in MPEG media entry and index to a track in the MPEG media entry respectively. In the definition of the MPEG_media extension, the tracks array is contained in an array of alternatives. The alternatives array is contained in media. Items in tracks[] may not necessarily follow the same indexing across different items in alternatives[]. Therefore, it is unclear from the MPEG_buffer_circular extension definition which item from the alternatives array is used.

1.4.2. MPEG_buffer_circular

glTF extension to specify circular buffer

Table 6. MPEG_buffer_circular Properties

	Type	Description	Required
count	integer	This provides the number of frames that are offered by this buffer.	No, default: 2
media	integer	The index of the MPEG media entry that provides the source.	✓ Yes
tracks	integer [1-*]	The array of indices of tracks the MPEG media entry that provides the source.	No
alternative	integer	The index of the alternative entry in MPEG media that provides the source.	No
extensions	object	JSON object with extension-specific objects.	No
extras	any	Application-specific data.	No

Additional properties are allowed.

- JSON schema: [MPEG_buffer_circular.schema.json](#)

1.4.2.1. MPEG_buffer_circular.count

This provides the number of frames that are offered by this buffer.

- **Type:** `integer`
- **Required:** No, default: `2`
- **Minimum:** `>= 2`

1.4.2.2. MPEG_buffer_circular.media

The index of the MPEG media entry that provides the source.

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

1.4.2.3. MPEG_buffer_circular.tracks

The array of indices of tracks the MPEG media entry that provides the source.

- **Type:** `integer [1-*)`
 - Each element in the array **MUST** be greater than or equal to `0`.
- **Required:** No

1.4.2.4. MPEG_buffer_circular.alternative

The index of the alternative entry in MPEG media that provides the source.

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

1.4.2.5. MPEG_buffer_circular.extensions

JSON object with extension-specific objects.

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

1.4.2.6. MPEG_buffer_circular.extras

Application-specific data.

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



Note

Items in alternatives[] should provide alternative to the equivalent content offered in the parent MPEG_media.media[] item.

1.5. MPEG_node_avatar

Source: [m61818](#)

1.5.1. General

The [Figure 3](#) illustrates the additional proposed attributes.

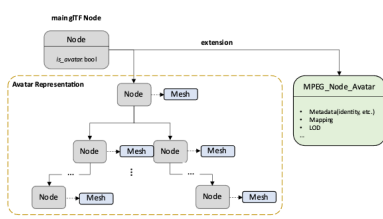


Figure 3. Proposed extension of the avatar representation in the scene description

1.5.2. Semantics

1.5.2.1. MPEG_node_avatar.metadata object

MPEG node avatar is used to represent and support avatars.

Table 7. MPEG_node_avatar.metadata object Properties

	Type	Description	Required
parameters	object		No
name	string	Name of the avatar.	✓ Yes
extensions	object	JSON object with extension-specific objects.	No
extras	any	Application-specific data.	No

Additional properties are allowed.

- **JSON schema:** [MPEG_node_avatar.metadata.schema.json](#)

1.5.2.1.1. MPEG_node_avatar.metadata.parameters

- **Type:** object
- **Required:** No
- **Allowed values:**

1.5.2.1.2. MPEG_node_avatar.metadata.name

Name of the avatar.

- **Type:** `string`
- **Required:** ✓ Yes

1.5.2.1.3. MPEG_node_avatar.metadata.extensions

JSON object with extension-specific objects.

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

1.5.2.1.4. MPEG_node_avatar.metadata.extras

Application-specific data.

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

1.5.2.2. MPEG_node_avatar_representation extension

The `MPEG_node_avatar_representation` is used to generally support avatars in the scene description. Providing additional information relative to level of appearance detail, vertex mapping between anatomical body parts, and common general information about the user.

Table 8. `MPEG_node_avatar_representation` extension Properties

	Type	Description	Required
metadata	<code>MPEG_node_avatar.metadata [1-*</code>	An array of trackables.	✓ Yes
lod	<code>object []</code>	Reference to the chosen level of detail to be used for the visual appearance.	No
mapping	<code>object []</code>	The mapping between child nodes and vertex semantics.	No
extensions	<code>object</code>	JSON object with extension-specific objects.	No
extras	<code>any</code>	Application-specific data.	No

Additional properties are allowed.

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

1.5.2.2.1. MPEG_node_avatar_representation extension.metadata

An array of trackables.

- **Type:** `MPEG_node_avatar.metadata [1-*]`
- **Required:** ✓ Yes

1.5.2.2.2. MPEG_node_avatar_representation extension.lod

Reference to the chosen level of detail to be used for the visual appearance.

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

1.5.2.2.3. MPEG_node_avatar_representation extension.mapping

The mapping between child nodes and vertex semantics.

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

1.5.2.2.4. MPEG_node_avatar_representation extension.extensions

JSON object with extension-specific objects.

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

1.5.2.2.5. MPEG_node_avatar_representation extension.extras

Application-specific data.

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

1.5.2.3. Texture Info

Reference to a texture.

Table 9. Texture Info Properties

	Type	Description	Required
index	integer	The index of the texture.	✓ Yes
texCoord	integer	The set index of texture's TEXCOORD attribute used for texture coordinate mapping.	No, default: 0
extensions	object	JSON object with extension-specific objects.	No
extras	any	Application-specific data.	No

Additional properties are allowed.

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

1.5.2.3.1. textureInfo.index

The index of the texture.

- **Type:** integer
- **Required:** ✓ Yes
- **Minimum:** ≥ 0

1.5.2.3.2. textureInfo.texCoord

This integer value is used to construct a string in the format `TEXCOORD_<set index>` which is a reference to a key in `mesh.primitives.attributes` (e.g. a value of 0 corresponds to `TEXCOORD_0`). A mesh primitive **MUST** have the corresponding texture coordinate attributes for the material to be applicable to it.

- **Type:** integer
- **Required:** No, default: 0
- **Minimum:** ≥ 0

1.5.2.3.3. textureInfo.extensions

JSON object with extension-specific objects.

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

1.5.2.3.4. textureInfo.extras

Application-specific data.

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

1.5.3. Example

Example of the MPEG_node_avatar extension

```
{
  "extensionsUsed": [
    "MPEG_node_avatar"
  ],
  "scene": 0,
  "scenes": [
    {
      "name": "Scene",
      "nodes": [
        1
      ]
    }
  ],
  "nodes": [
    {
      "extensions": {
        "MPEG_node_avatar": {
          "is_avatar": true,
          "extensions": {
            "MPEG_node_avatar_representation": {
              "metadata": [
                {
                  "parameters": {
                    "age": 37,
                    "gender": "None"
                  },
                  "name": "MyAvatar",
                }
              ],
              "lod": [
                {
                  "textures": [
                    {
                      "index": 0,
                      "texCoord": 0
                    }
                  ],
                  "resolution": "High_Resolution"
                }
              ]
            }
          }
        }
      }
    }
  ]
}
```

```

    ],
    "mapping": [
      {
        "vertex_id": {
          "0": 36583
        },
        "label": "head"
      }
    ]
  }
}
],
"meshes": [
  {}
],
"materials": [
  {}
],
"textures": [
  {}
]
}

```

1.6. Shadow Scenes

Source: [m62227](#)

A scene may oftentimes contain content that is not meant to be visible but provides valuable information to the Presentation Engine for processing the scene. An example is given by the simplified representation of the scene that is used for providing Acoustic properties of the scene. A simplified geometry description of the scene or some nodes in the scene, to which additional acoustic properties may be attached, would be needed. For example, a wall may visually have lots of details on the surface but can simply be presented by a plane for the purpose of describing the acoustic reflections on the wall.

MPEG-I SD needs to provide for means to describe these invisible simplified geometries in the scene.

We propose to identify the need and if necessary, work on a solution for describing invisible simplified geometries that can be used for defined additional properties, such as acoustic properties.

Chapter 2. ISOBMFF

2.1. Carriage Format for animation timing

Source: [m56039](#)

2.1.1. Multiple animations

2.1.1.1. Problem description

The current syntax for `glTFAnimationSample` allows multiple animations to be triggered at a certain point in time that applies to several objects. Also, while playing an animation, a further animation could be triggered simultaneously affecting the same object that is started on top on the already running one.

There are different examples for which multiple animations running in parallel might be useful. One could be two sequential animation having a short overlapping interval so that the transition phase from one animation to another does not look abrupt. For instance, if the first animation is walking slowly and the second is running one could have a transition phase where the two animations (walking and running) are actuating onto the 3D object and being each of it balanced properly so that the overall timeline looks good and there is not an abrupt change.

Other might be simply a complete overlap of multiple animations that might actuate at the same time. E.g., a person walking and at a certain point on top of walking a head turning animation is triggered for a while. In such a case, the walking animation will also have an impact on joints involving head movement, e.g., some tilting of the head. As for the current solution, there is no clue on how such animations are to be played at the same time.

- Are both of the animations being applied simultaneously?
- In which order are the animations are applied if both applied at the same time? The result might not be the same if they are applied in different order.
- Is there some kind of average contribution of each animation computed to the final render? For a realistic combination of multiple animations, it is necessary to allow controlling how multiple animations affect the target nodes and its property (i.e. `animation.channel.target.path`). For instance, in case of multiple animation, if only one animation is allowed to affect a node. Then any effect from any other animations on that particular node must be zeroed. Also, in the example mentioned above, only the tilting of the head from the walking animation could be kept and any other channel acting on a node affecting the head movement would be zeroed. Therefore, weight for each channel of the animations is provided in the proposed solution.

In order to address the questions: * With simultaneous playback of multiple animations, a subset of channels can be allowed to influence the node transformations partially or fully (depending on the weights)

- With an explicit order index assigned to an animation, simultaneous animations can be applied in an orderly manner
- The associated weight factor for an animation, influence its contribution to the final render

2.1.1.2. Syntax

```
aligned(8) class glTFAnimationSample
{
    unsigned int(1) apply_to_all;
    unsigned int(7) reserved;
    unsigned int(16) num_events;
    for( i=0; i < num_events; i++ ){
        unsigned int(32) index;
        int(32) speed;
        unsigned int(8) state;
        unsigned int (8) order_id;
        unsigned int(32) num_channels;
        for (int j = 0; j < num_channels; j++) {
            int (8) weight[j];
            unsigned int (32) channel_index[j];
        }
    }
}
```

2.1.1.3. Semantics

order_id – specifying a value to indicate the order in which animations are applied. Animations with lower values are applied before animation with higher values.

num_channels – specifying the number of channels of an animation for which a weight is provided.

weight[j] – specifying the weight to be applied to the j-th channel of the animation in units of 1/255.

channel_index[j] – specifying the index of the j-th channel of the animation.

2.1.2. Interaction of animation and dynamic 3D object

2.1.2.1. Problem description

Similar to the discussion above, it is currently not clear on how an animation or multiple animations and a dynamic 3D object can be combined together. The problem statement is:

- What is the result of a video of a dynamic 3D object when it is still being actively played (i.e., a dynamic object that changes over time) and an animation is triggered on top of it?

The proposed solution is similar to what it is proposed for multiple animations above. In this solution, the dynamic behavior of the 3D object can be expressed by channels specific to the objects. Thereby each channel transformations can be controlled and merged with transformations introduced by any externally triggered animations.

Taking the same example as above, a dynamic 3D object could be walking and a head turning animation is trigger to be played on top.

2.1.2.2. Syntax

```
aligned(8) class glTFAnimationSample
{
    unsigned int(1) apply_to_all;
    unsigned int(7) reserved;
    unsigned int(16) num_events;
    unsigned int(16) num_objects;
    for( i=0; i < num_objects; i++ ){
        unsigned (8) obj_order_id;
        unsigned int(32) obj_num_channels;
        unsigned int (32) object_index;
        for (int j = 0; j < obj_num_channels; j++) {
            unsigned (8) obj_weight[j];
            unsigned int (32) obj_channel_index[j];
        }
    }
    for( i=0; i < num_events; i++ ){
        unsigned int(32) index;
        int(32) speed;
        unsigned int (8) order_id;
        unsigned int(32) num_channels;
        for (int j = 0; j < num_channels; j++) {
            int (8) weight[j];
            unsigned int (32) channel_index[j];
        }
    }
}
```

2.1.2.3. Semantics

num_objects – specifying the number of dynamic 3D objects

object_index - specifying the node index of a dynamic 3D object

obj_order_id – specifying a value to indicate the order in which transformations are applied. Dynamic transformation of 3D objects with lower values are applied before the transformation introduced with higher values.

obj_num_channels – specifying the number of channels which are dynamically changing the 3D object

obj_weight[j] – specifying the influence for each channel which affects the dynamicity of the 3D object

obj_channel_index[j] - specifying the index for the channel which affects the dynamicity of the 3D object.

order_id – specifying a value to indicate the order in which transformations are applied. Transformations with lower values are applied before transformations with higher values.

`num_channels` – specifying the number of channels of an animation for which a weight is provided.

`weight[j]` – specifying the weight to be applied to the j-th channel of the animation in units of 1/255.

`channel_index[j]` – specifying the index of the j-th channel of the animation.

2.2. Improvements for MPEG-I SD random access description

Source: [m58853](#)

2.2.1. 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.

2.2.2. 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.

2.2.3. Description and processing of random access points

2.2.3.1. 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 processed 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.

2.2.3.2. 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 4 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 4. 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.

2.2.4. Proposed text improvements

2.2.4.1. 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.

2.2.4.2. 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_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 succeeding sample.

Chapter 3. Codec Support

3.1. Clarification of type of V-PCC track referenced from MPEG_media

Source: [m57336](#)

3.1.1. Consideration

Though it has been proposed and discussed how to indicate V-PCC-specific attributes and how to associate those with accessors, it is still unclear how the referenced track is indicated in MPEG_media.

There are two alternatives in how to encapsulate V-PCC data into ISOBMFF; single track encapsulation and multi-track encapsulation.

Thus, the referenced track indication in MPEG_media is considered for all combinations of the pipeline options and the V-PCC encapsulation options.

For pipeline option#1

A MPEG_media is associated with 1 buffer.

- If V-PCC data is encapsulated as single track, there is one V3C bitstream track in ISOBMFF. Hence, it is obvious that the referenced track in MPEG_media is V3C bitstream track.
- Otherwise (V-PCC data is encapsulated as multi track), there are multiple tracks such as V3C atlas track and V3C video component tracks. As V3C atlas track is the entry point and has track references to the V3C video component tracks, it is straight forward to indicate V3C atlas track as the referenced track in MPEG_media.

For pipeline option#2

There is one MPEG_media associated with individual buffer for position and V-PCC-specific attributes.

- If V-PCC data is encapsulated as single track, there is one V3C bitstream track in ISOBMFF. Hence, the referenced track of MPEG_media needs to be the identical V3C bitstream track.
- Otherwise (V-PCC data is encapsulated as multi track), there are multiple tracks such as V3C atlas track and V3C video component tracks. As V3C atlas track is the entry point and has track references to the V3C video component tracks, it is straight forward to indicate V3C atlas track as the referenced track in MPEG_media.

3.1.2. Proposal

Based on the consideration above, it is proposed to add the following text in the MPEG-I Part 14 specification text.

For both cases that point cloud reconstruction is performed by the MAF and PE,

MPEG_buffer_circular associated with each attribute shall refer the same MPEG_media. The referenced track in MPEG_media shall be specified as follows.

- For single-track encapsulated V3C data, the referenced track in MPEG_media shall be the V3C bitstream track.
- For multi-track encapsulated V3C data, the referenced track in MPEG_media shall be the V3C atlas track.

3.2. Dynamic mesh support in scene description

Source: [m57410](#)

3.2.1. Introduction

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

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

3.2.2. Design

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

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

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

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

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

3.2.3. Assets and Implementation

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

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

The only deviation is the compression of the texture image sequence into an HEVC bitstream that can be used with the already supported video texture extension.

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

Figure 5 depicts the setup:

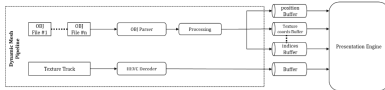


Figure 5. n/a

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

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

Source: [m62515](#)

3.3.1. Multiple atlases

3.3.1.1. 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.

3.3.1.2. 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.

3.3.1.3. 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.

3.3.1.3.1. MPEG_primitive_V3C

glTF extension to specify support for V3C compressed primitives.

Table 10. `MPEG_primitive_V3C` Properties

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

Additional properties are allowed.

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

3.3.1.3.1.1. MPEG_primitive_V3C.atlases

An array of atlases

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

3.3.1.3.1.2. 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

3.3.1.3.1.3. MPEG_primitive_V3C.extensions

JSON object with extension-specific objects.

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

3.3.1.3.1.4. MPEG_primitive_V3C.extras

Application-specific data.

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

3.3.1.3.2. MPEG_primitive_V3C._MPEG_V3C_CAD

defines the common atlas data for a v3c object

Table 11. 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

3.3.1.3.2.1. 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

3.3.1.3.2.2. MPEG_primitive_V3C._MPEG_V3C_CAD.extensions

JSON object with extension-specific objects.

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

3.3.1.3.2.3. MPEG_primitive_V3C._MPEG_V3C_CAD.extras

Application-specific data.

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

3.3.1.3.3. MPEG_primitive_V3C.atlas

glTF extension to specify support for V3C compressed primitives.

Table 12. *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

	Type	Description	Required
extras	any	Application-specific data.	No

Additional properties are allowed.

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

3.3.1.3.3.1. MPEG_primitive_V3C.atlas._MPEG_V3C_CONFIG

- **Type:** integer
- **Required:** ✓ Yes
- **Minimum:** ≥ 0

3.3.1.3.3.2. MPEG_primitive_V3C.atlas._MPEG_V3C_AD

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

- **Type:** integer
- **Required:** ✓ Yes
- **Minimum:** ≥ 0

3.3.1.3.3.3. 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

3.3.1.3.3.4. 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

3.3.1.3.3.5. 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

3.3.1.3.3.6. 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

3.3.1.3.3.7. MPEG_primitive_V3C.atlas.extensions

JSON object with extension-specific objects.

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

3.3.1.3.3.8. MPEG_primitive_V3C.atlas.extras

Application-specific data.

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

3.3.1.3.4. MPEG_primitive_V3C.attribute

defines the attribute of a V3C object.

Table 13. `MPEG_primitive_V3C.attribute` Properties

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

Additional properties are allowed.

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

3.3.1.3.4.1. MPEG_primitive_V3C.attribute.type

provides the type of the attribute.

- **Type:** `integer`

- **Required:** No
- **Minimum:** ≥ 0
- **Maximum:** ≤ 255

3.3.1.3.4.2. 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

3.3.1.3.4.3. MPEG_primitive_V3C.attribute.extensions

JSON object with extension-specific objects.

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

3.3.1.3.4.4. 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 3.3.1.3.3](#)

```

{
  "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
          }
        }
      }
    }
  ]
}

```

3.3.2. References

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

3.4. Immersive audio extension

Source: [m63549](#)

3.4.1. 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.

3.4.2. 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

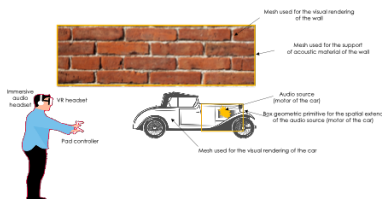


Figure 6. Virtual objects having dedicated representations for audio and visual renderers

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.

3.4.3. 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 14](#), we describe and compare the different capabilities of MPEG_audio_spatial and the MPEG-I Audio solution.

Table 14. 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.

	MPEG_audio_spatial	MPEG-I Audio	New Extension
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.
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.

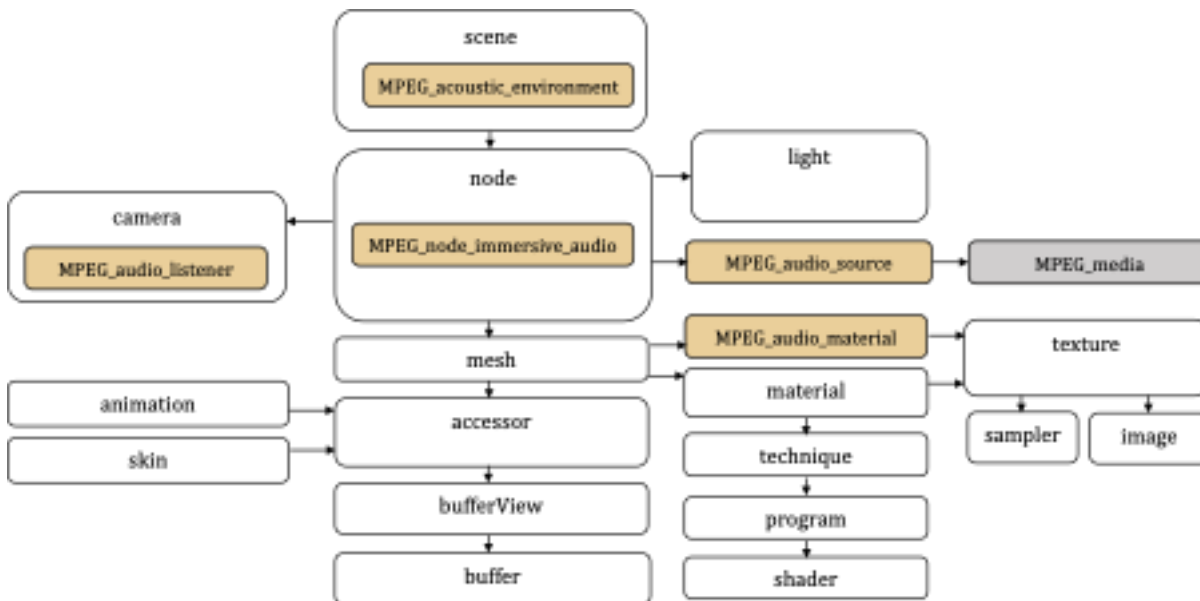


Figure 7. Proposed new MPEG glTF extensions to support MPEG-I immersive audio

3.4.3.1. 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.

3.4.3.2. 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.

3.4.3.3. 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])

3.4.3.4. Audio/acoustic material data at mesh primitive level

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

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

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

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

3.4.4. References

[1] ISO/IEC 23090-14

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

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

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

3.5. On G-PCC support

Source: [m63070](#)

3.5.1. Consideration on in-GPU processing

3.5.1.1. Geometry processing

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

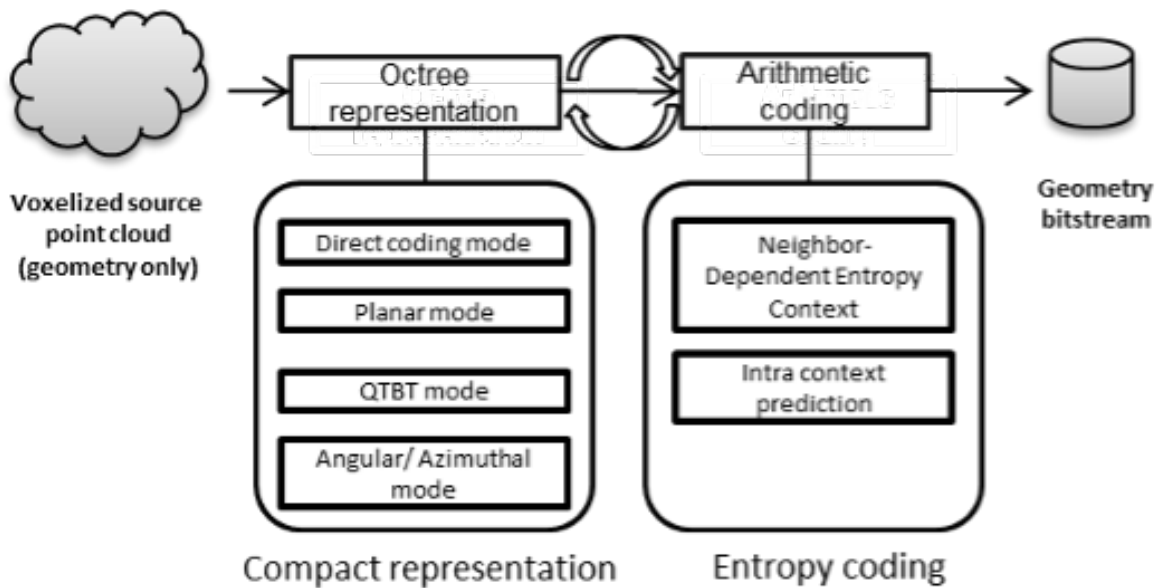


Figure 8. Encoding process for geometry data

Hence, the decoding process flow is as follows:

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

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

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

separately distributed to MAF and PE.

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

3.5.1.2. Attribute processing

The encoding process for attribute data of G-PCC bitstream is shown in Figure 9.

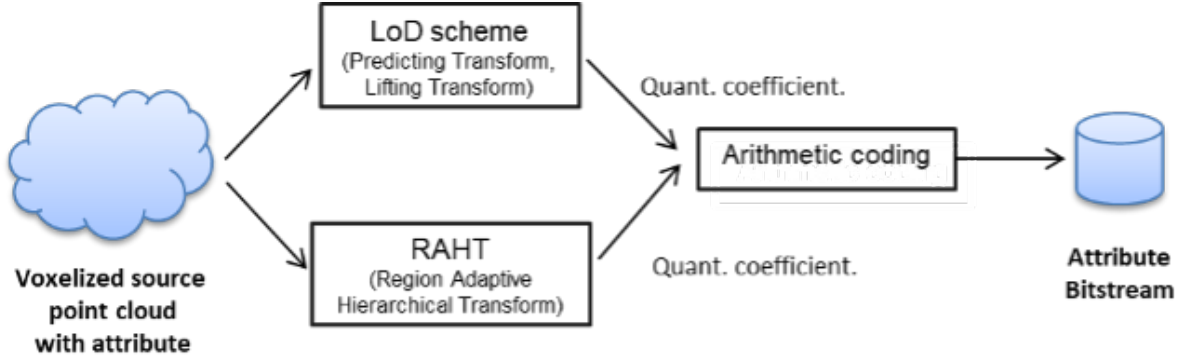


Figure 9. Encoding process for attribute data

Hence, the decoding process flow is as follows:

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

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

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

3.5.1.3. Summary

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

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

3.5.2. Proposal

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

reconstruct point cloud data by utilizing GPU.

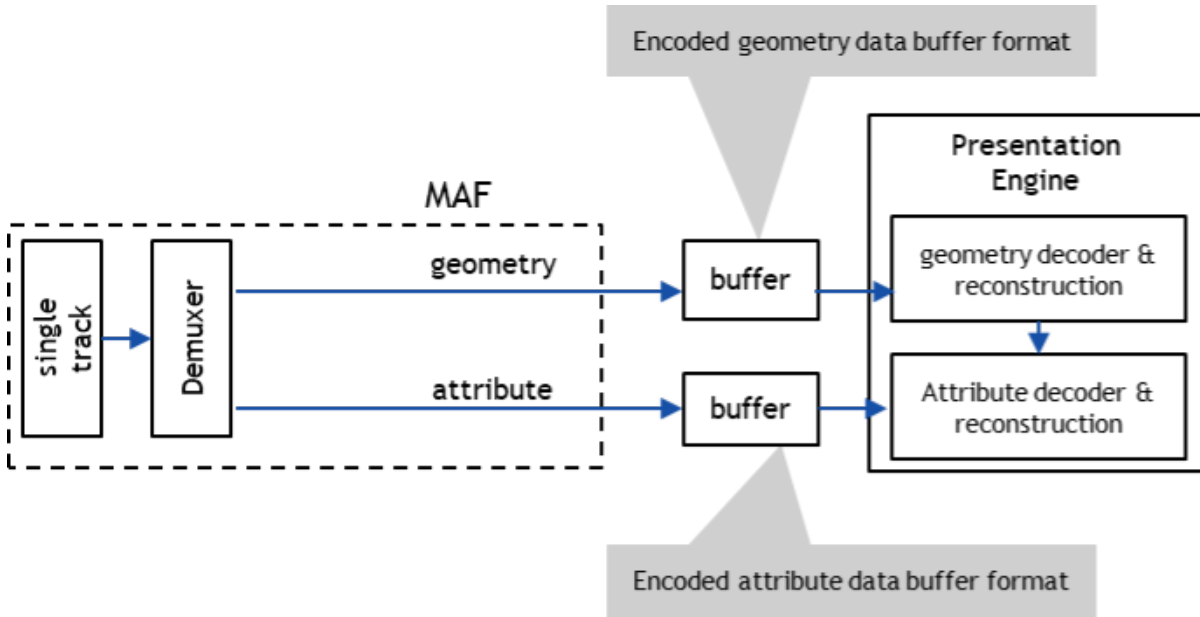


Figure 10. The proposed pipeline

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

3.5.3. Reference

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

3.5.4. Annex. Proposed MPEG extension

It is proposed new MPEG extension, MPEG_primitive_GPCC.

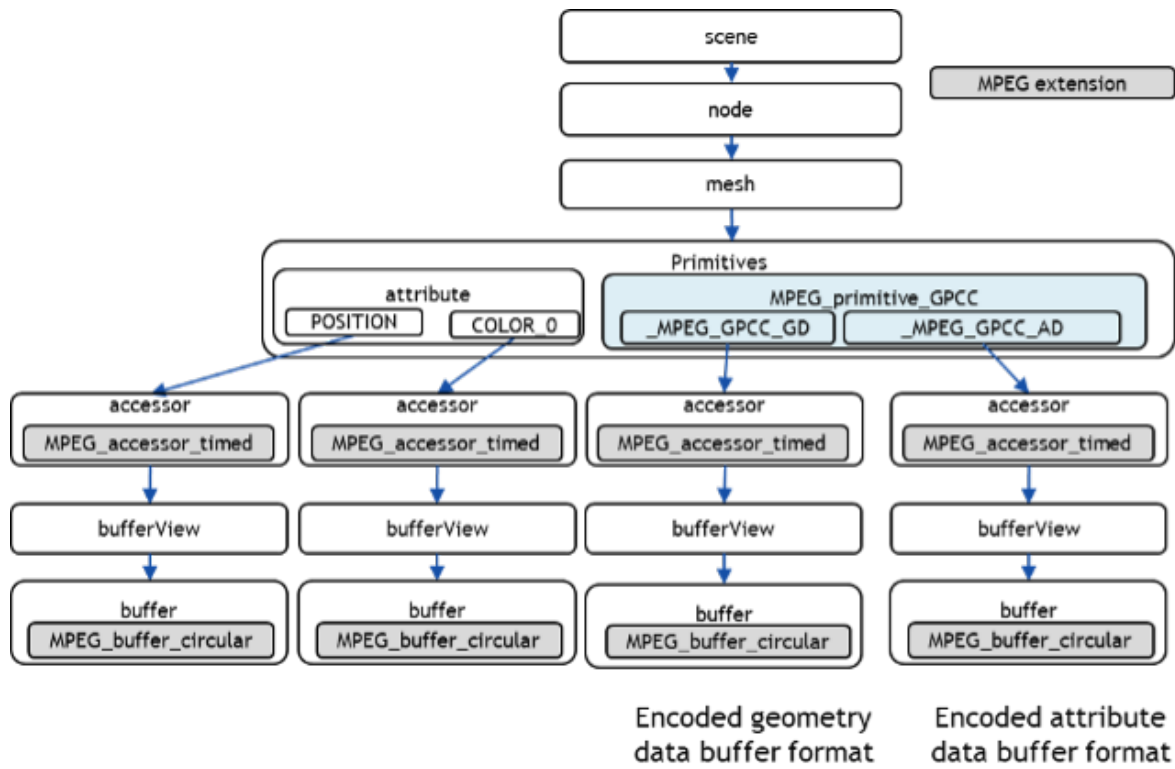


Figure 11. Overview of the *MPEG_primitive_GPCC*

Properties of *MPEG_primitive_GPCC*:

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

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

Properties of _MPEG_GPCC_AD object:

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

Encoded geometry data buffer format:

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

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

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

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

Chapter 4. Data Formats

4.1. Support of glTF CBOR binary format

Source: [m56102](#)

4.1.1. Problem Statement

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

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

4.1.2. Benefit of CBOR file/data format:

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

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

4.1.3. CBOR data size comparison example:

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

Table 15. n/a

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

4.1.4. Use Cases

4.1.4.1. CBOR binary data associated with “url”

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

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

4.1.4.2. Using CBOR file instead of JSON

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

4.1.4.3. Using CBOR as local data storage

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

4.1.5. Potential Solutions

4.1.5.1. Proposed CBOR Parser API

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

The CBOR parser API offers the following methods:

Table 16. Description of CBOR Parser API

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

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

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

4.1.5.2. Proposed Test Cases

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

be the followings:

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

4.1.6. Open Issue Discussion

4.1.6.1. CBOR IPR

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

4.1.6.2. CBOR data security

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

4.1.6.3. Implementation

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

4.1.6.4. Potential Data format issue

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

Chapter 5. Interfaces

5.1. On DASH Dynamic Bitrate Adaption with Viewpoint Update

Source: [m56094](#)

5.1.1. Problem Statement

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

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

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

5.1.2. Use Cases

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

Table 17. n/a

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

5.1.2.1. One timed media playback

A simple use case is there is only one DASH-based timed media is played in a scene as shown in [Figure 12](#). Currently, the media is rendered based on the MPEG_media extension with configurable parameters such as autoplay, loop, etc. DASH adaptative streaming in this case is used within its native mechanism by switching bitrate based on either network condition or buffer status. The key observation in this case is that the video keeps playing even when the viewport is not in focus. In an adequate network environment, DASH switches to the highest bitrate possible without considering

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



Figure 12. One DASH-based Timed Media Playback

5.1.2.2. More than one timed media playback

When there is more than one timed media is played at the same time, as shown in Figure 13, network bandwidth usage is similar to the use case in Section 5.1.2.1. However, the situation may get worse when all of the timed media are in a high-resolution setup. The lack of balancing network resources for each of the media play will worsen the view quality.

There are couple of scenarios in this use case:

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



Figure 13. Two DASH-based Timed Media Playback

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

5.1.3. Current Scene Description Support and Gaps

5.1.3.1. Support of viewpoint data fetching

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

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

The “ViewInfo” data structure is as follows:

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

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

- Pose
- Transform

5.1.3.2. Gaps Analysis

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

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

5.2. Supporting Multiple Viewers in the Media Access Function

Source: [m58510](#)

5.2.1. General

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

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

media parts for each viewer.

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

5.2.2. Proposed Updates to MAF API

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

5.2.2.1. Methods

Table 18. n/a

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

5.2.2.2. IDL for media pipeline

```

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

```

5.3. CoAP API support in MAF

Source: [m56739](#)

5.3.1. General

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

5.3.2. MAF as CoAP Client

In this clause, the proposed MAF API in [Table 19](#) applies to the case where the MAF acts as a CoAP client to fetch timed media from the CoAP media server. The CoAP API offers the following methods:

Table 19. Description of CoAP Client API

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

5.3.3. MAF as HTTP-CoAP Proxy

In this clause, the proposed MAF API in [Table 20](#) applies to the case where the MAF acts as an HTTP-CoAP proxy.

Table 20. Description of HTTP-CoAP proxy API

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

Chapter 6. MPEG-I Audio in Scene Description

6.1. MPEG-I Audio in Scene Description

Source: [m61180](#)

6.1.1. General

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

Figure 14 depicts the MPEG-I audio architecture:

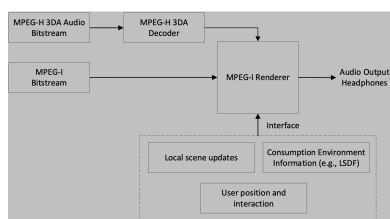


Figure 14. N/A

The MPEG-I render pipeline is depicted by [m61180_fig2]:

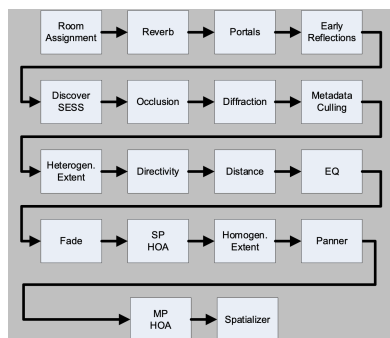


Figure 15. N/A

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 16](#):

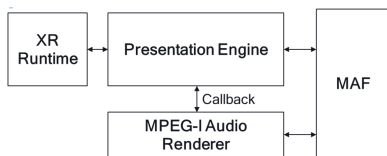


Figure 16. N/A

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 17](#):

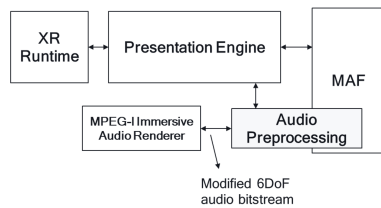


Figure 17. N/A

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.

Chapter 7. Reference Software

7.1. Thoughts on trimesh playback of AR scenes

Source: [m60282](#)

7.1.1. General

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

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

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

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

7.1.2. AR Sessions recording and format

7.1.2.1. AR Session in Google ARCore

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

This is depicted in [Figure 18](#) available in the [ARCore documentation](#).

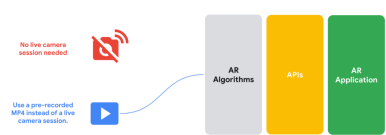


Figure 18. AR Session playback in ARCore

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

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

- Custom/user event

7.1.2.2. AR Session file format

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

Here are some dump information from the recorded files.

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

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

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

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

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

```

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

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

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

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

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

```

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

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

hardware depth sensor”.

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

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

7.1.3. AR Session playback in trimesh

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

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

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

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

Chapter 8. Interactivity framework

8.1. On event-based scene update

Source: [m61812](#)

8.1.1. General

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

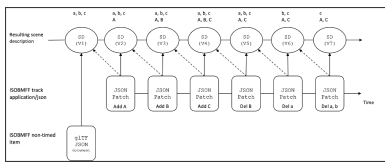


Figure 19. n/a

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.

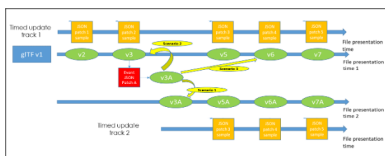


Figure 20. n/a

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.

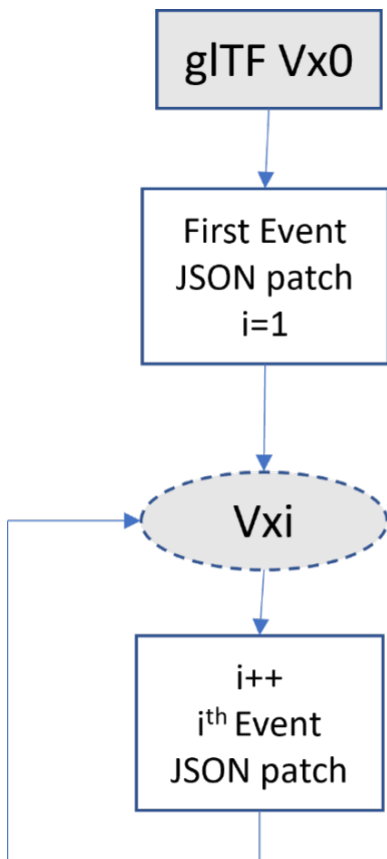


Figure 21. Event-based update diagram

8.1.2. A use case for event based updates

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

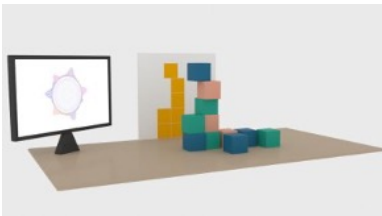


Figure 22. n/a



Figure 23. n/a

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

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

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

Two kinds of updates are triggered during the game:

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

8.1.3. JSON patch limitations

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

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

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

8.1.4. Semantics for event-based update

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

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

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

Table 21. 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 9. Collected problem statements and industry needs

9.1. On the support of real environment data

Source: [m61811](#)

9.1.1. General

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

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

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

9.1.2. Representation of the real environment

As shown in [Figure 24](#), 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

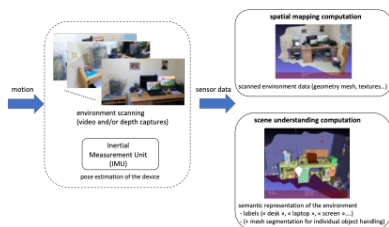


Figure 24. Computation of real environment data

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

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

9.1.3. Storing a representation of the real environment

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

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

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

Generating real environment data before runtime requires efficient storage. Storing real environment data in the Cloud has been investigated by ETSI Augmented Reality Framework (ARF). As shown in Figure 25, 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

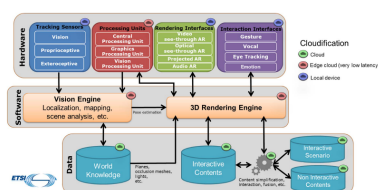


Figure 25. Global overview of the architecture of an AR system (from ETSI ARF)

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

9.1.4. Examples of framework for real environment handling

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

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

9.1.4.1. Microsoft's Mixed Reality framework

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

- a spatial computing module, generating a mesh representation of the real environment as shown in [Figure 26](#)
- 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 27](#)

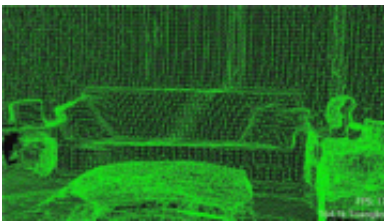


Figure 26. Mesh representation of the real environment after a spatial mapping computation



Figure 27. Semantic representation of the real environment after a scene understanding computation

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

```

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

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

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

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

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

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

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

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

// Save newSceneData for later

```

9.1.4.2. Apple's ARKit framework

On a fourth-generation iPad Pro running iPad OS 13.4 or later, Apple's ARKit uses the LiDAR Scanner to create a mesh representation of the user real environment. Then this mesh is further segmented and multiple anchors, called ARMeshAnchor, are assigned to the resulting set of segmented meshes. As shown in [Figure 28](#), a semantic labeling is performed for the real objects that ARKit can identify such as ceiling, door, floor, seat, table, wall and window labels.

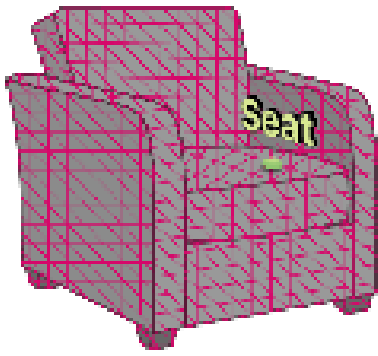


Figure 28. Semantic labeling of Apple's ARKit

These real environment data attached to the ARMeshAnchors can be saved and loaded by serializing/deserializing an ARWorldMap as shown in [Figure 29](#).

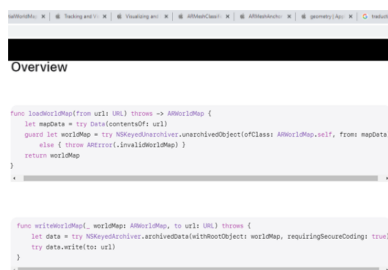


Figure 29. Saving and loading an Apple's ARKit ARWorldMap

9.1.4.3. Meta/Oculus framework

The Meta/Oculus framework has been developed for Meta Quest 2 and Meta Quest Pro devices. The scene understanding computation provides a scene model, which is a representation of the user real environment. The scene model contains Scene Anchors, with each anchor being attached to geometric components and semantic labels. The floor, ceiling, wall_face, desk, couch, door_frame and window_frame labels are currently supported as shown in [Figure 30](#).

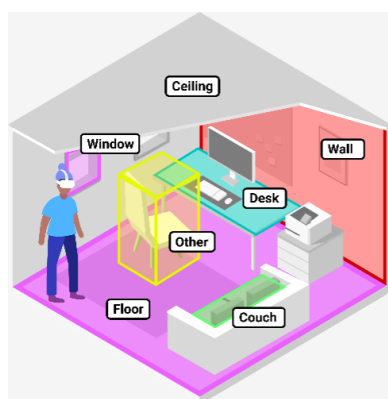


Figure 30. Semantic labeling of the Meta/Oculus Scene Understanding

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()
```

Appendix A: JSON Schema for extensions

A.1. JSON Schema for MPEG_buffer_circular extension

```
{
  "$schema" : "http://json-schema.org/draft-07/schema",
  "title" : "MPEG_buffer_circular",
  "type" : "object",
  "description": "glTF extension to specify circular buffer",
  "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
  "properties" : {
    "count": {
      "type": "integer",
      "default": 2,
      "minimum": 2,
      "description": "This provides the number of frames that are offered by
this buffer."
    },
    "media": {
      "allOf": [ { "$ref": "glTFid.schema.json" } ],
      "description": "The index of the MPEG media entry that provides the
source."
    },
    "tracks": {
      "type": "array",
      "items": {
        "allOf": [ { "$ref": "glTFid.schema.json" } ]
      },
      "minItems": 1,
      "description": "The array of indices of tracks the MPEG media entry that
provides the source."
    },
    "alternative": {
      "type": "integer",
      "allOf": [ { "$ref": "glTFid.schema.json" } ],
      "description": "The index of the alternative entry in MPEG media that
provides the source."
    },
    "extensions": {},
    "extras": {}
  },
  "required": [ "media" ]
}
```

A.2. JSON Schema for MPEG_media

```

{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "MPEG_media",
  "type": "object",
  "description": "MPEG media used to create a texture, audio source or other objects
in the scene.",
  "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
  "properties": {
    "media": {
      "type": "array",
      "description": "An array of MPEG media. A MPEG media contains data
referred by other object in a scene",
      "items": {
        "$ref": "MPEG_media.media.schema.json"
      },
      "minItems": 1
    },
    "extensions": {},
    "extras": {}
  },
  "required": [ "media" ]
}

```

A.3. JSON Schema for MPEG_media.media

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "MPEG_media.media",
  "type": "object",
  "description": "MPEG media used to create a texture, audio source, or any other
media type defined by MPEG.",
  "properties": {
    "name": { },
    "startTime": {
      "type": "number",
      "minimum": 0.0,
      "default": 0.0,
      "exclusiveMinimum": false,
      "description": "The startTime gives the time at which the rendering of the
timed texture will be in seconds. "
    },
    "startTimeOffset": {
      "type": "number",
      "minimum": 0.0,
      "default": 0.0,
      "exclusiveMinimum": false,
      "description": "The startTimeOffset indicates the time offset into the
source, starting from which the timed texture is generated."
    },
    "endTimeOffset": {
      "type": "number",
      "minimum": 0.0,
      "description": "The endTimeOffset indicates the time offset into the
source, up to which the timed texture is generated. The value is provided in seconds,
where 0 corresponds to the start of the source."
    },
    "autoplay": {
      "type": "boolean",
      "description": "Specifies that the MPEG media start playing as soon as it
is ready."
    },
    "autoplayGroup": {
      "type": "boolean",
      "description": "Specifies that playback starts simultaneously for all
media sources with the autoplay flag set to true."
    },
    "loop": {
      "type": "boolean",
      "default": false,
      "description": "Specifies that the MPEG media start over again, every time
it is finished."
    },
    "controls": {
```

```

        "type": "object",
        "$ref": "MPEG_media.media.controls.schema.json"
    },
    "alternatives": {
        "type": "array",
        "description": "An array of alternatives of the same media (e.g. different
video code used)",
        "items": {
            "uri": {
                "type": "string",
                "description": "The uri of the media.",
                "format": "uriref",
                "glTF_detailedDescription": "The uri of the media. Relative paths
are relative to the .glTF file.",
                "glTF_uriType": "media"
            },
            "mimeType": {
                "anyOf": [
                    {
                        "type": "string",
                        "enum": [ "video/mp4", "application/dash+xml" ]
                    },
                    {
                        "type": "string"
                    }
                ],
                "description": "The MPEG media's MIME type."
            },
            "tracks": {
                "type": "array",
                "description": "List of all tracks in MPEG media container (e.g.
mp4 file or DASH manifest",
                "items": {
                    "track": {
                        "type": "string",
                        "description": "URL fragments e.g, DASH : Using MPD
Anchors (URL fragments) as defined in Annex C of ISO/IEC 23009-1 (Table C.1). ISOBMFF:
URL fragments as specified in Annex L of ISO/IEC 14496."
                    },
                    "codec": {
                        "type": "string",
                        "description": "The codecs parameter, as defined in IETF
RFC 6381, of the media included in the track."
                    }
                }
            },
            "extraparams": {
                "type": "object",
                "additionalProperties": true
            },
            "required": [ "uri", "mimeType" ]
        }
    }

```

```
    },  
    "minItems": 1  
  }  
}
```

A.4. JSON Schema for MPEG_media.media.controls

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "MPEG_media.media.controls",
  "type": "object",
  "description": "Specifies that which MPEG media controls should be exposed to end user",
  "properties": {
    "pauseSupported": {
      "type": "boolean",
      "description": "Pause control displayed for the MPEG media.",
      "default": true
    },
    "fastForwardSupported": {
      "type": "boolean",
      "description": "Fast forward control displayed for the MPEG media.",
      "default": true
    },
    "fastBackwardSupported": {
      "type": "boolean",
      "description": "Fast backward control displayed for the MPEG media.",
      "default": true
    }
  }
}
```

A.5. JSON Schema for MPEG_node_transformation_external

```
{
  "$schema" : "http://json-schema.org/draft-04/schema",
  "title" : "MPEG_node_transformation_external",
  "type" : "object",
  "description": "glTF extension to specify pose in scene is dependent on external
information",
  "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],
  "properties" : {
    "matrix": {
      "uri": {
        "type": "string",
        "description": "The uri provides node's source of the transformation
matrix",
        "gltf_detailedDescription": "A floating-point 4x4 transformation
matrix stored in column-major order.",
        "gltf_webgl": "`uniformMatrix4fv()` with the transpose parameter equal
to false"
      }
    },
    "rotation": {
      "uri": {
        "type": "string",
        "description": "The uri provides node's source of unit quaternion
rotation in the order (x, y, z, w), where w is the scalar."
      },
      "description": "The node's unit quaternion rotation in the order (x, y, z,
w), where w is the scalar."
    },
    "scale": {
      "uri": {
        "type": "string",
        "description": "The uri provides node's source of non-uniform scale,
given as the scaling factors along the x, y, and z axes."
      }
    },
    "translation": {
      "uri": {
        "type": "string",
        "description": "The uri provides node's source of translation along
the x, y, and z axes."
      }
    }
  }
}
```

A.6. JSON Schema for MPEG_buffer_circular

```
{
  "$schema" : "http://json-schema.org/draft-07/schema",
  "title" : "MPEG_buffer_circular",
  "type" : "object",
  "description": "glTF extension to specify circular buffer",
  "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
  "properties" : {
    "count": {
      "type": "integer",
      "default": 2,
      "minimum": 2,
      "description": "This provides the number of frames that are offered by
this buffer."
    },
    "media": {
      "allOf": [ { "$ref": "glTFId.schema.json" } ],
      "description": "The index of the MPEG media entry that provides the
source."
    },
    "tracks": {
      "type": "array",
      "items": {
        "allOf": [ { "$ref": "glTFId.schema.json" } ]
      },
      "minItems": 1,
      "description": "The array of indices of tracks the MPEG media entry that
provides the source."
    },
    "alternative": {
      "type": "integer",
      "allOf": [ { "$ref": "glTFId.schema.json" } ],
      "description": "The index of the alternative entry in MPEG media that
provides the source."
    },
    "extensions": {},
    "extras": {}
  },
  "required": [ "media" ]
}
```

A.7. JSON Schema for MPEG_audio_spatial.source

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "title" : "MPEG_audio_spatial.source",
  "type" : "object",
  "description": "",
  "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
  "properties": {
    "id": {
      "description": "A unique identifier",
      "gltf_detailedDescription": "A unique identifier of the audio source in the scene.",
      "type": "integer",
      "minimum": 0
    },
    "type": {
      "description": "A type of the audio source",
      "gltf_detailedDescription": "Indicates the type of the audio source. `type` value equal to `Object` indicates mono object. `type` value equal to `HOA` indicates HOA object",
      "type": "string",
      "enum": ["Object", "HOA"]
    },
    "pregain": {
      "description": "A level-adjustment of the audio source",
      "gltf_detailedDescription": "Provides a level-adjustment in dB for the signal associated with the source.",
      "type": "number",
      "default": 0.0,
      "minimum": 0.0
    },
    "playbackSpeed": {
      "description": "Playback speed of the audio source",
      "gltf_detailedDescription": "Playback speed of the audio signal. A value of 1.0 corresponds to playback at normal speed. The value shall be between 0.5 and 2.0.",
      "type": "number",
      "minimum": 0.5,
      "maximum": 2.0,
      "default": 1.0
    },
    "attenuation": {
      "description": "A function used to calculate the attenuation of the audio source.",
      "gltf_detailedDescription": "Indicates the function used to calculate the attenuation of the audio signal based on the distance to the source. attenuation value equal to `noAttenuation` indicates no attenuation function should be used. attenuation value equal to `inverseDistance` indicates inverse distance function should be used. attenuation value equal to `linearDistance` indicates linear distance function should
```

be used. attenuation value equal to 'exponentialDistance' indicates exponential distance function should be used. attenuation value equal to 'custom' indicates custom function should be used. The definition of 'custom' function is outside of the scope of ISO/IEC 23090-14. The attenuation functions and their parameters are defined in ISO/IEC 23090-14:Annex D.",

```

    "enum": ["noAttenuation", "inverseDistance", "linearDistance",
"exponentialDistance", "custom"],
    "default": "linearDistance"
  },
  "attenuationParameters": {
    "description": "An array of attenuation parameters",
    "glTF_detailedDescription": "An array of parameters that are input to the
attenuation function. The semantics of these parameters depend on the attenuation
function itself and are defined in ISO/IEC 23090-14",
    "type": "array",
    "items": {
      "type": "number"
    },
    "minItems": 1
  },
  "referenceDistance": {
    "description": "A distance in meters.",
    "glTF_detailedDescription": "Provides the distance in meters for which the
distance gain is implicitly included in the source signal after application of
pregain. When type equals HOA the element shall not be present.",
    "type": "number",
    "default": 1.0,
    "minimum": 1.0
  },
  "accessors": {
    "description": "An array of accessors that describe the audio source",
    "glTF_detailedDescription": "An array of accessor references, by
specifying the accessors indices in accessors array, that describe the buffers where
the decoded audio will be made available.",
    "type": "array",
    "items": {
      "allOf": [ { "$ref": "glTFid.schema.json" } ]
    },
    "minItems": 1
  },
  "reverbFeed": {
    "description": "An array of pointers to reverb units",
    "glTF_detailedDescription": "Provides one or more pointers to reverb
units, optionally extended by a floating point scaling factor. A reverb unit
represents a reverberation audio processor that is configured by the metadata from a
single reverb object. Typically, a reverb object represents reverberation properties
of a single room.",
    "type": "array",
    "items": {
      "type": "integer"
    }
  }

```

```

    },
    "reverbFeedGain": {
      "description": "An array of gain values",
      "glTF_detailedDescription": "Provides an array of gain [dB] values to be
applied to the source's signal(s) when feeding it to the corresponding reverbFeed. The
array shall have the same number of elements as the reverbFeed array field.",
      "type": "array",
      "items": {
        "type": "number"
      }
    },
    "isCluster": {
      "description": "Indicates whether the audio source is a cluster.",
      "glTF_detailedDescription": "Specifies if the audio source is a pre-mixed
representation of a selection of audio sources.",
      "type": "boolean",
      "default": false
    },
    "clusterProperties": {
      "description": "An object describing cluster properties.",
      "glTF_detailedDescription": "A sourceClusterProperties object describing
cluster properties. This field must be defined, when the isCluster is set to True.",
      "$ref": "MPEG_audio_spatial.source.cluster.schema.json"
    },
    "extensions": {},
    "extras": {}
  },
  "required": [
    "id", "type", "accessors", "isCluster"
  ]
}

```

A.8. JSON Schema for MPEG_audio_spatial.source.cluster

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "title" : "MPEG_audio_spatial.source.cluster",
  "type" : "object",
  "description": "",
  "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
  "properties": {
    "sourceId": {
      "description": "An array of audio source ids.",
      "gltf_detailedDescription": "An array of integers that contains the unique identifiers of the audio sources contained in this cluster.",
      "type": "array",
      "items": {
        "type": "integer"
      },
      "minItems": 1
    },
    "radius": {
      "description": "A cluster radius.",
      "gltf_detailedDescription": "A distance in meters used to encompass the audio sources contained in this cluster.",
      "type": "number",
      "default": 0.0,
      "minimum": 0.0
    },
    "extensions": {},
    "extras": {}
  },
  "required": [
    "sourceId", "radius"
  ]
}
```

A.9. JSON Schema for MPEG_node_avatar.metadata object

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "title": "MPEG_node_avatar.metadata object",
  "type": "object",
  "description": "MPEG node avatar is used to represent and support avatars.",
  "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
  "properties": {
    "parameters": {
      "anyOf": [
        { "type": "object", "properties": { "age": { "type": "number",
"description": "Age of the avatar." } } },
        { "type": "object", "properties": { "gender": { "type": "string",
"description": "Gender of the avatar." } } }
      ]
    },
    "name": {
      "type": "string",
      "description": "Name of the avatar."
    },
    "extensions": {},
    "extras": {}
  },
  "required": [ "name" ]
}
```

A.10. JSON Schema for MPEG_node_avatar_representation extension

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "MPEG_node_avatar_representation extension",
  "type": "object",
  "description": "The MPEG_node_avatar_representation is used to generally support avatars in the scene description. Providing additional information relative to level of appearance detail, vertex mapping between anatomical body parts, and common general information about the user.",
  "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
  "properties": {
    "metadata": {
      "type": "array",
      "description": "An array of trackables.",
      "items": {
        "$ref": "MPEG_node_avatar.metadata.schema.json"
      },
      "minItems": 1
    },
    "lod": {
      "type": "array",
      "description": "Reference to the chosen level of detail to be used for the visual appearance.",
      "items": {
        "type": "object",
        "properties": {
          "texture": {
            "allOf": [ { "$ref": "textureInfo.schema.json" } ],
            "description": "LOD of the appearance.",
            "glTF_detailedDescription": "The level of details of the appearance of the avatar."
          },
          "type": {
            "type": "string",
            "enum": ["High_Resolution", "Low_Resolution", "Reference", "Other"],
            "description": "Indicates the resolution of the texture.",
            "default": "Reference"
          }
        }
      }
    },
    "mapping": {
      "type": "array",
      "description": "The mapping between child nodes and vertex semantics.",
      "items": {
        "type": "object",
```

```

"properties":{
  "vertex_id": {
    "allOf": [ { "$ref": "glTFid.schema.json" } ],
    "description": "The index of a semantical vertex.",
    "gltf_detailedDescription": "Indicates the index to the 3D
mesh of which semantically corresponds to this node_avatar."
  },
  "label": {
    "type": "string",
    "enum": ["Full",
      "Upper_Body",
      "Head",
      "Face",
      "Back/Neck/Ears",
      "Mouth_Bag",
      "Lower_Jaw",
      "Upper_Jaw",
      "Eye_Left",
      "Eye_Right",
      "Chest",
      "Chest_Front",
      "Shoulder_Front_Left",
      "Shoulder_Front_Right",
      "Chest_Back",
      "Shoulder_Back_Left",
      "Shoulder_Back_Right",
      "Arm_Left",
      "Upper_Arm_Left",
      "Lower_Arm_Left",
      "Hand_Left",
      "Hand_Left_Palm",
      "Hand_Left_Thumb",
      "Hand_Left_Index",
      "Hand_Left_Middle",
      "Hand_Left_Ring",
      "Hand_Left_Little",
      "Arm_Right",
      "Upper_Arm_Right",
      "Lower_Arm_Right",
      "Hand_Right",
      "Hand_Right_Palm",
      "Hand_Right_Thumb",
      "Hand_Right_Index",
      "Hand_Right_Middle",
      "Hand_Right_Ring",
      "Hand_Right_Little",
      "Lower_Body",
      "Abdomen",
      "Abdomen_Front",
      "Pelvis_Front",
      "Abdomen_Back",

```

```

        "Pelvis_Back",
        "Leg_Left",
        "Upper_Leg_Left",
        "Lower_Leg_Left",
        "Foot_Left",
        "Leg_Right",
        "Upper_Leg_Right",
        "Lower_Leg_Right",
        "Foot_Right"
    ],
    "description": "Indicates the semantical body region."
  },
  },
  "required": [ "properties", "type" ]
}
},
"extensions": {},
"extras": {}
},
"required": [ "metadata" ]
}

```

A.11. JSON Schema for MPEG_primitive_V3C

```

{
  "$schema" : "http://json-schema.org/draft-07/schema",
  "title" : "MPEG_primitive_V3C",
  "type" : "object",
  "description": "glTF extension to specify support for V3C compressed primitives.",
  "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
  "properties" : {
    "atlases": {
      "type": "array",
      "description": "An array of atlases",
      "gltf_detailedDescription": "An array of atlases",
      "items": {
        "$ref": "MPEG_primitive_V3C.atlas.schema.json"
      },
      "minItems" : 1
    },
    "_MPEG_V3C_CAD": {
      "type" : "object",
      "allOf": [{ "$ref": "MPEG_primitive_V3C._MPEG_V3C_CAD.schema.json"}],
      "description": "This object lists different properties described for the  
Common Atlas Data in ISO/IEC 23090-5." ,
      "gltf_detailedDescription" : "This object lists different properties  
described for the Common Atlas Data in ISO/IEC 23090-5.",
      "minItems":0
    },
    "extensions": {},
    "extras": {}
  },
  "required": ["atlases"]
}

```

A.12. JSON Schema for MPEG_primitive_V3C._MPEG_V3C_CAD

```
{
  "$schema" : "http://json-schema.org/draft-07/schema",
  "title" : "MPEG_primitive_V3C._MPEG_V3C_CAD",
  "type" : "object",
  "description": "defines the common atlas data for a v3c object",
  "allOf": [ { "$ref": "glTFProperty.schema.json"} ],
  "properties" : {
    "MIV_view_parameters": {
      "type": "integer",
      "description": "indicates the accessor index which is used to refer to the  
list of MIV view parameters.",
      "gltf_detailedDescription": "indicates the accessor index which is used to  
refer to the list of MIV view parameters.",
      "minimum": 1
    }
  },
  "required": [
    "MIV_view_parameters"
  ]
}
```

A.13. JSON Schema for MPEG_primitive_V3C.atlas

```
{
  "$schema" : "http://json-schema.org/draft-07/schema",
  "title" : "MPEG_primitive_V3C.atlas",
  "type" : "object",
  "description": "glTF extension to specify support for V3C compressed primitives.",
  "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
  "properties" : {
    "_MPEG_V3C_CONFIG": {
      "allOf": [ { "$ref": "glTFid.schema.json" } ],
      "description": "",
      "gltf_detailedDescription": ""
    },
    "_MPEG_V3C_AD": {
      "allOf": [ { "$ref": "glTFid.schema.json" } ],
      "description": "",
      "gltf_detailedDescription": "a reference to the accessor that points to
the atlas data."
    },
    "_MPEG_V3C_GVD_MAPS": {
      "type": "array",
      "description": "an array of references to video texture maps.",
      "gltf_detailedDescription": "an array of references to video textures that
provide the geometry maps.",
      "items": {
        "allOf": [ { "$ref": "glTFid.schema.json" } ]
      },
      "minItems": 1
    },
    "_MPEG_V3C_OVD_MAP": {
      "type": "array",
      "description": "a reference to a video texture that provides the occupancy
map",
      "gltf_detailedDescription": "a reference to a video texture that provides
the occupancy map",
      "items": {
        "allOf": [ { "$ref": "glTFid.schema.json" } ]
      },
      "minItems": 0
    },
    "_MPEG_V3C_AVD": {
      "type": "array",
      "description": "",
      "gltf_detailedDescription": "An array of references to the video textures
that provide the attribute data",
      "items": {
        "$ref": "MPEG_primitive_V3C.attribute.schema.json"
      },
      "minItems": 0
    }
  }
}
```

```

    },
    "_MPEG_V3C_CAD": {
      "type" : "object",
      "description": "This object lists different properties described for the  
Common Atlas Data in ISO/IEC 23090-5." ,
      "glTF_detailedDescription" : "This object lists different properties  
described for the Common Atlas Data in ISO/IEC 23090-5.",
      "minItems":0
    },
    "extensions": {},
    "extras": {}
  },
  "required": ["_MPEG_V3C_CONFIG", "_MPEG_V3C_AD", "_MPEG_V3C_GVD_MAPS"]
}

```

A.14. JSON Schema for MPEG_primitive_V3C.attribute

```
{
  "$schema" : "http://json-schema.org/draft-07/schema",
  "title" : "MPEG_primitive_V3C.attribute",
  "type" : "object",
  "description": "defines the attribute of a V3C object.",
  "allOf": [ { "$ref": "glTFProperty.schema.json" } ],
  "properties" : {
    "type": {
      "type": "integer",
      "description": "provides the type of the attribute.",
      "gltf_detailedDescription": "provides the type of the attribute.",
      "minimum": 0,
      "maximum": 255
    },
    "maps": {
      "type": "array",
      "description": "",
      "gltf_detailedDescription": "provides the references to the corresponding
video texture maps.",
      "items": {
        "allOf": [ { "$ref": "glTFid.schema.json" } ]
      },
      "minItems": 1
    }
  },
  "required": ["maps"]
}
```

Appendix B: 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.