Technology under Consideration for ISO/IEC 23090-14

WG3 Scene Description BoG

MDS21733_WG03_N00604

Table of Contents

Chapter 1. Extensions

1.1. MPEG_media

Source: m56047

1.1.1. General

PIt 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	`MPEG_media.media [1-*]`	An array of MPEG media. A MPEG media contains data referred by other object in a scene	✔ Yes
extensions	`object`	JSON object with extension-specific objects.	No
extras	`any`	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	`any`		No
startTime	`number`	The startTime gives the time at which the rendering of the timed texture will be in seconds.	No, default: `0`
startTimeOffset	`number`	The startTimeOffset indicates the time offset into the source, starting from which the timed texture is generated.	No, default: `0`

	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	`boolean`	Pause control displayed for the MPEG media.	No, default: `true`
fastForwardSupported	`boolean`	Fast forward control displayed for the MPEG media.	No, default: `true`
fastBackwardSupported	`boolean`	Fast backward control displayed for the MPEG media.	No, default: `true`

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: m55132

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' or **'Cluster'**
- HOA audio sources shall ignore the parent node's position and be rendered only in 3DoF.
- **Cluster audio source is a pre-mixed representation of a selection of audio sources as a single source.**
- 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.

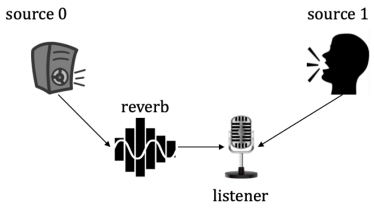

Figure 1. An example of the processing chain for audio in a scene

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. MPEG_audio_spatial.source

Table 4. `MPEG_audio_spatial.source` *Properties*

	Type	Description	Required
id	`integer`		✔ Yes
type	`string`		✔ Yes
pregain	`number`		No, default: `0`
playbackSpeed	`number`		No, default: `1`
attenuation	`any`		No, default: `linearDistance`
attenuationParameters	`number [1-*]`		No
referenceDistance	`number`		No, default: `1`
accessors	`integer [1-*]`		No
reverbFeed	`integer []`		No
reverbFeedGain	`number []`		No
clusters	`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. MPEG_audio_spatial.source.id

- **Type**: `integer`
- **Required**: ✔ Yes

1.2.2.2. MPEG_audio_spatial.source.type

- **Type**: `string`
- **Required**: ✔ Yes
- **Allowed values**:
  - `"Object"`
  - `"HOA"`
  - `"Cluster"`

1.2.2.3. MPEG_audio_spatial.source.pregain

- **Type**: `number`

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

1.2.2.4. MPEG_audio_spatial.source.playbackSpeed

- **Type**: `number`
- **Required**: No, default: `1`
- **Minimum**: `>= 0.5`
- **Maximum**: `<= 2`

1.2.2.5. MPEG_audio_spatial.source.attenuation

- **Type**: `any`
- **Required**: No, default: `linearDistance`
- **Allowed values**:
  - `noAttenuation`
  - `inverseDistance`
  - `linearDistance`
  - `exponentialDistance`
  - `custom`

1.2.2.6. MPEG_audio_spatial.source.attenuationParameters

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

1.2.2.7. MPEG_audio_spatial.source.referenceDistance

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

1.2.2.8. MPEG_audio_spatial.source.accessors

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

1.2.2.9. MPEG_audio_spatial.source.reverbFeed

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

1.2.2.10. MPEG_audio_spatial.source.reverbFeedGain

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

1.2.2.11. MPEG_audio_spatial.source.clusters

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

1.2.2.12. MPEG_audio_spatial.source.extensions

JSON object with extension-specific objects.

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

1.2.2.13. MPEG_audio_spatial.source.extras

Application-specific data.

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

1.2.3. MPEG_audio_spatial.source.cluster

Provides a list of cluster elements that are attached to this node.

Table 5. `MPEG_audio_spatial.source.cluster` *Properties*

	Type	Description	Required
id	`number`	unique identifier of the audio cluster in the scene.	No
sourceId	`number`	indicates the audio source id that represents the aggregated sources comprising this Cluster.	No
audioSources	`number []`	array of audio sources that are aggregated and represented by this cluster.	No

	Type	Description	Required
radius	`number`	indicates the distance in meters of the encompassed aggregated audio sources.	No

Additional properties are allowed.

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

1.2.3.1. MPEG_audio_spatial.source.cluster.id

unique identifier of the audio cluster in the scene.

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

1.2.3.2. MPEG_audio_spatial.source.cluster.sourceId

indicates the audio source id that represents the aggregated sources comprising this Cluster.

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

1.2.3.3. MPEG_audio_spatial.source.cluster.audioSources

array of audio sources that are aggregated and represented by this cluster.

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

1.2.3.4. MPEG_audio_spatial.source.cluster.radius

indicates the distance in meters of the encompassed aggregated audio sources.

- **Type**: `number`
- **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.

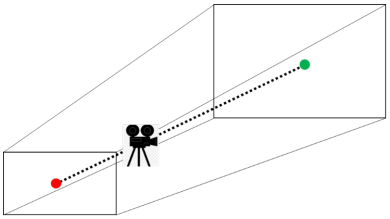

Figure 2. Example of Camera Path Segment with Bounding Volume

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	`number`	Number of anchor points in the camera paths.	No

	Type	Description	Required
segments	`number`	The type of the bounding volume for the path segments. Possible types are: * BV_NONE: no bounding volume * BV_CONE: capped cone bounding volume, defined by a circle at each anchor point. * 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. * 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. default: BV_NONE	No
boundingVolume	`number`	Quaternion describing the rotation of the scene in the anchor space. centerPosition and orientation are used as alternatives to transformation. default:false	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_node_transformation_external

Source: m56440

1.4.1. General

In order to enable node transformations based on external information sources, MPEG node transformation external is defined. The MPEG node transformation external extension is identified by MPEG_node_transformation_external, which shall be included in the extensionsUsed and extensionsRequired of the scene description document, whenever external node transformation is used in a scene. The MPEG node transformation external extension acts as a glue for linking nodes and node transformations with information sources that originate outside of the scene description document. For example, the extension enables adding a virtual object, which is positioned and oriented, based on viewer position and orientation. The transformation properties (matrix, rotation, translation, and scale) provided by the external entity are applied on top of the default properties defined by the concerned node.

1.4.2. MPEG_node_transformation_external

glTF extension to specify pose in scene is dependent on external information

Table 6. `MPEG_node_transformation_external` *Properties*

	Type	Description	Required
matrix	`any`		No
rotation	`any`	The node's unit quaternion rotation in the order (x, y, z, w), where w is the scalar.	No
scale	`any`		No
translation	`any`		No
name	`string`	The user-defined name of this object.	No
extensions	`object`	JSON object with extension-specific objects.	No
extras	`any`	Application-specific data.	No

Additional properties are allowed.

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

1.4.2.1. MPEG_node_transformation_external.matrix

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

1.4.2.2. MPEG_node_transformation_external.rotation

The node's unit quaternion rotation in the order (x, y, z, w), where w is the scalar.

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

1.4.2.3. MPEG_node_transformation_external.scale

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

1.4.2.4. MPEG_node_transformation_external.translation

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

1.4.2.5. MPEG_node_transformation_external.name

The user-defined name of this object. This is not necessarily unique, e.g., an accessor and a buffer could have the same name, or two accessors could even have the same name.

- **Type**: `string`
- **Required**: No

1.4.2.6. MPEG_node_transformation_external.extensions

JSON object with extension-specific objects.

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

1.4.2.7. MPEG_node_transformation_external.extras

Application-specific data.

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

1.4.3. Processing Model

The processing model could be the following.

- Application parses scene description document, which contains MPEG node transformation external extension.
- MPEG node transformation extension identifies that a node in glTF document may be linked with external information.
- The application decides, which type of external information suits the handle defined in MEPG node transformation external extension.
- The application applies transformation properties, as described by the extension, received from the external information source to the related node.
- The external transformation information is always applied on top of the node default properties.

It is always left for the application to decide, how to precisely map the URIs in the extension with external information sources visible to it. If no proper mapping can be performed, application may choose to ignore rendering of such nodes.

1.4.4. Example

```
{
    "nodes": [
        {
            "name": "Box 1",
            "translation": [
                0.0,
                0.0,
                0.2
            ]
        },
        {
            "name": "Box 2 with external transformation",
            "extensions": {
                "MPEG_node_transformation_external": {
                    "matrix": {
                          "uri": "mpeg:transformation:viewer:hand:left"
                      }
                }
             }
        }
    ]
}
```

In the example a virtual object is positioned in the scene in relation to information received with what application considers as proper input for: mpeg:transformation:viewer:hand:left. The virtual object is properly transformed according to left hand controller and always positioned with offset of 0.2m in z-axis. Other examples of URIs may include the following:

- mpeg:transformation:viewer
- mpeg:transformation:viewer:hand:right

- mpeg:transformation:viewer:hand:left
- mpeg:transformation:viewer:bounding_box
- mpeg:transformations:marker:plane
- mpeg:transformations:play_area:bounding_box The URI could be defined as well outside of the MPEG.
- geo location URI could have embedded base64-encoded data in the following format data:[<mediatype>][;base64],<data>. For example 'data:text/plain;charset=UTF-8; 35.1592;-98.4422;410'
- geo location URI according to RFC 5870 'geo:37.786971,-122.399677;u=35'

1.5. MPEG_buffer_circular

Source: m58186

1.5.1. 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.

1.5.2. MPEG_buffer_circular

glTF extension to specify circular buffer

Table 7. `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

	Type	Description	Required
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`

1.5.2.1. MPEG_buffer_circular.count

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

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

1.5.2.2. MPEG_buffer_circular.media

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

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

1.5.2.3. 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

1.5.2.4. MPEG_buffer_circular.alternative

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

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

1.5.2.5. MPEG_buffer_circular.extensions

JSON object with extension-specific objects.

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

1.5.2.6. 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.

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 process by a Presentation Engine already in the processing of that track without breaking continuity of the Presentation Engine. So, there must be description about how to process sync samples by a Presentation Engine already in the processing of a track. In this case, an ISOBMFF file track carrying scene description data can have more than one sync sample and all of each sync samples will contain a glTF JSON document which defines the status of the nodes at the presentation time of the sync sample. The Presentation Engine which has not processed any sample before the current sync sample can process a sync sample as normal scene description document. However, the Presentation Engine already processed any samples before the current sync sample in decoding order should process a sync sample as scene update even though document in the sample is not in the form of JSON patch. Therefore, the description about such processing model should be defined. Otherwise, there should be a restriction that only one sync sample is allowed in the track with MPEG-I Scene Description data.

2.2.3.2. 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 3 shows an example with redundant samples for random access. In this example, a track with scene description data has two redundant samples denoted as R. The redundant sample R8 whose composition time is between U7 and U9 contains a glTF JSON document contains description of the scene at the time of the composition time of R8. The The Presentation Engine starting from middle of the track starts play either R5 or R8, then play U6 or U9, respectively. The The Presentation Engine starting from the begining of the track starts play D0 and ignore R5 and R8. As the sample duration of U4 and U7 will be extended by sample duration of R5 and R8, respectively, the scene description information in U4 and U7 must consider that the Presentation Engine will play it longer than the duration of the sample containing it. For example, the animation of active scene of the Presentation Engine according to the animation samplers provided by the sample U4 and the samples before that sample may continue until it receives any updated animation samplers by the U6 sample or the samples after that sample.

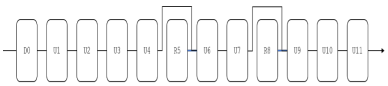

Figure 3. An example structure of scene description data with shadow sync samples

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

2.2.4. 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 compsotion time of that sample and would only be made available by the ISOBMFF parser to the Presentation Engine if the processing of the file starts with this sample. Otherwise, it is expected that the sample be ignored, and that processing of the current sample is continued beyond the duration of current sample for a duration equal to the duration of the ignored sample, as defined in ISO/IEC 14496-12.

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

Chapter 3. Codec Support

3.1. On V3C Support in Scene Description

Source: m56240

The MPEG-I Scene Description solution defines an MPEG_media extension that is used to reference external media. The media can be used for different purposes, e.g. to provide vertex buffers, vertex indices, vertex attributes, texture, audio samples, or metadata.

Scene Description has a requirement to support V-PCC compressed media and V3C by generalization. The MPEG_media element would point into a V3C compressed stream, which is identified through appropriate MIME type settings. The MAF will then set up the appropriate media pipeline to decode and process the V3C data to match the requested output buffer format(s). The buffer format(s) are described by the corresponding timed accessor and buffer views.

Figure 4 shows different ways of creating media pipelines for the processing and rendering of V3C compressed data.

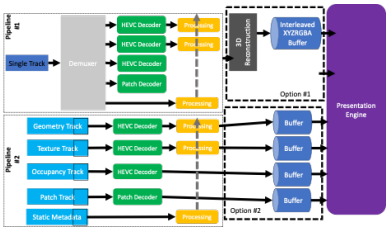

Figure 4. n/a

As shown in the figure, the V3C content may be coming from a single track or from multiple tracks, but has to ultimately be demultiplexed and each component decoded separately. Finally, the data may be passed in an interleaved manner or in separate buffers to the Presentation Engine. It is up to the Presentation Engine to decide how it wants to receive the data. Note that both Pipeline #1 and #2 can branch into Options #1 or #2 when handing over the data to the Presentation Engine.

In Option #1, which is the simplest, the full 3D reconstruction is performed by the media pipeline to reproduce a reconstructed 3D object. In this option, the 3D reconstruction usually takes place in the CPU, which might result in performance degradation.

An alternative to this approach is shown in Option #2 of the diagram. As stated earlier, the data can come from a single track or from multiple tracks. However, in this case, the Presentation Engine expects each component to be available in a separate buffer. It then uses custom shader programs to reconstruct and render the point cloud in the GPU. This has the advantage of improved rendering performance and lower CPU load.

To enable option 2, appropriate identification of the type and format of data in each buffer as well as of the role of that component is necessary. Currently, two solutions are proposed to address this issue:

- Define specific primitive attribute for each component (see contribution m54514)
- Use an association extension to associate components with attributes (see contribution m55929)

These 2 solutions should be evaluated in order to select the most appropriate solution.

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

Source: m57336

3.2.1. 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 considerd 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.

3.2.2. 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.

3.3. Dynamic mesh support in scene description

Source: m57410

3.3.1. 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.

3.3.2. 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.

3.3.3. 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:

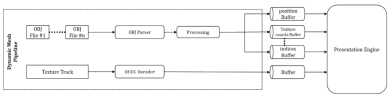

Figure 5. n/a

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

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

Table 8. 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:xxxxxxxx"
}
```

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 9. 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 10. 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 6. 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 6. 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 7, 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 7. Two DAH-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 Gasps

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-viwer 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 11. 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 12 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 12. Description of CoAP Client API

Method	Brief Description
`fetch ()`	The MAF sends media resource request to a CoAP server
`receive ()`	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 13 applies to the case where the MAF acts as an HTTP-CoAP proxy.

Table 13. Description of HTTP-CoAP proxy API

Method	Brief Description
`hc()`	The MAF maps the HTTP requests to CoAP and forward them to CoAP Server

Chapter 6. Reference Software

6.1. Thoughts on trimesh playback of AR scenes

Source: m60282

6.1.1. General

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

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

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

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

6.1.2. AR Sessions recording and format

6.1.2.1. AR Session in Google ARCore

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

This is depicted in Figure 8 available in the ARCore documentation.

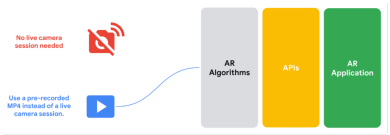

Figure 8. AR Session playback in ARCore

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

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

- Custom/user event

6.1.2.2. AR Session file format

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

Here are some dump information from the recorded files.

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

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

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

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

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

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

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

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

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

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

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

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

hardware depth sensor”.

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

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

6.1.3. AR Session playback in trimesh

As presented in clause Section 6.1.2, the ARCore API provides the ability to record all the information pertaining to an AR session in terms of sensor data and user events.

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

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

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

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_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": {
            "type": "integer",
            "minItems": 0
        },
        "type": {
            "type": "string",
            "enum": ["Object", "HOA", "Cluster"]
        },
        "pregain": {
            "type": "number",
            "default": 0.0,
            "minimum": 0.0
        },
        "playbackSpeed": {
            "type": "number",
            "minimum": 0.5,
            "maximum": 2.0,
            "default": 1.0
        },
        "attenuation": {
            "enum": ["noAttenuation", "inverseDistance", "linearDistance",
"exponentialDistance", "custom"],
            "default": "linearDistance"
        },
        "attenuationParameters": {
            "type": "array",
            "items": {
                "type": "number"
            },
            "minItems": 1
        },
        "referenceDistance": {
            "type": "number",
            "default": 1.0,
            "minimum": 1.0
        },
        "accessors": {
            "type": "array",
            "items": {
                "allOf": [ { "$ref": "glTFid.schema.json" } ]
            },
            "minItems": 1
        },
        "reverbFeed": {
            "type": "array",
```

```
            "items": {
                "type": "integer"
            }
        },
        "reverbFeedGain": {
            "type": "array",
            "items": {
                "type": "number"
            }
        },
        "clusters": {
            "type": "array",
            "items": {
                "allOf": [ { "$ref": "MPEG_audio_spatial.source.cluster.schema.json" }
]
            }
        }
    },
    "required": [
        "id", "type", "accessor"
    ]
}
```

A.3. 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": "Provides a list of cluster elements that are attached to this
node.",
    "properties": {
        "id": {
            "description": "unique identifier of the audio cluster in the scene.",
            "type": "number"
        },
        "sourceId": {
            "description": "indicates the audio source id that represents the
aggregated sources comprising this Cluster.",
            "type": "number"
        },
        "audioSources": {
            "description": "array of audio sources that are aggregated and represented
by this cluster.",
            "type": "array",
            "items": {
                "type": "number"
            }
        },
        "radius": {
            "description": "indicates the distance in meters of the encompassed
aggregated audio sources.",
            "type": "number"
        }
    }
}
```

A.4. 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.5. 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.6. 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.7. 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.8. 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" ]
}
```

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 automaticaly generated using wetzel tool under Apache License 2.0.