



Technology under Consideration for ISO/IEC 23090-14

WG3 Scene Description BoG

MDS24160_WG03_N01314

Table of Contents

1. Extensions	1
1.1. The Physics glTF extension and interactivity	1
1.1.1. 1 Introduction	1
1.1.2. 2 Khronos Physics Extensions	1
1.1.3. 3 Relationship to the MPEG Interactivity Extension	2
2. Codec Support	3
2.1. Dynamic mesh support in scene description	3
2.2. Support for multiple atlases for MIV applications (MPEG142)	3
2.2.1. Multiple atlases	3
2.2.2. References	10
3. Interfaces	11
3.1. Supporting Multiple Viewers in the Media Access Function	11
3.1.1. General	11
3.1.2. Proposed Updates to MAF API	11
3.2. Generic API for Presentation Engine	12
3.2.1. Generic Render Control API	13
3.2.2. Extension for Audio Node Mapping	16
4. MPEG-I Audio in Scene Description	18
4.1. Immersive audio extension	18
4.1.1. Introduction	18
4.1.2. Background	18
4.1.3. MPEG-I immersive audio support	19
4.1.4. References	23
4.2. MPEG-I Audio in Scene Description	23
4.2.1. General	23
4.3. Establishing a Mapping between Audio and MPEG-I Scenes	25
4.3.1. General	25
4.3.2. Extension for Audio Node Mapping	25
4.4. On spatial synchronization between graphs	26
4.4.1. Attempt problem definition for the spatial synchronization	26
4.4.2. Approach proposal for the spatial synchronization	29
4.4.3. Conclusion	31
4.4.4. References	31
5. Interactivity framework	32
5.1. On event-based scene update	32
5.1.1. General	32
5.1.2. A use case for event based updates	33
5.1.3. JSON patch limitations	34

5.1.4. Semantics for event-based update	35
6. Collected problem statements and industry needs	37
6.1. On the support of real environment data	37
6.1.1. General	37
6.1.2. Representation of the real environment	37
6.1.3. Storing a representation of the real environment	38
6.1.4. Examples of framework for real environment handling	39
6.2. The support of XR Spatial Computing of real environment	42
6.2.1. Introduction	42
6.2.2. Configuration examples of XR Spatial Computing	42
6.2.3. Approach proposal to support XR Spatial Computing	44
Appendix A: Disclaimer	46

Chapter 1. Extensions

1.1. The Physics glTF extension and interactivity

Source: [m67814](#)

1.1.1. 1 Introduction

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

1.1.2. 2 Khronos Physics Extensions

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

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

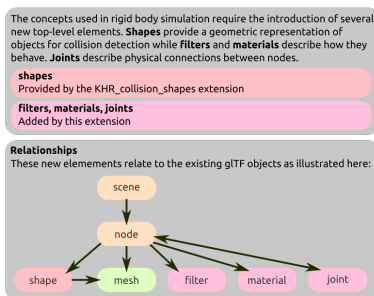


Figure 1. A black and white image of a camera Description automatically generated

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

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

slider (movement along one axis), or ball-and-socket (rotation around all axes). The joint property also includes descriptions of joint limits and joint drives. The restrictions for the movement of one rigid body with respect to the other are expressed by a limit object, which sets min and max values for the position/rotation along a specific axis. A joint_drive object further describes additional forces that are applied by a joint on a connected rigid body object.

As mentioned above, colliders are described through references to the KHR_collision_shapes, which is an independent extension. The extension currently supports the definition of the following shapes: sphere, box, capsule, cylinder, convex, and trimesh.

1.1.3. 3 Relationship to the MPEG Interactivity Extension

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

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

Chapter 2. Codec Support

2.1. Dynamic mesh support in scene description

V-DMC is considered for future Amendment

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

Source: [m62515](#)

2.2.1. Multiple atlases

2.2.1.1. Motivation

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

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

2.2.1.2. Overview

The proposals take the following aspects into consideration:

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

2.2.1.3. Array of atlases

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

The `atlasID` property is an integer values, where each integer value refers to the `vps_atlas_id`

specified in sub-clause 8.4.4 in [3] for each atlas in the V3C bitstream.

2.2.1.3.1. MPEG_primitive_V3C

glTF extension to specify support for V3C compressed primitives.

Table 1. MPEG_primitive_V3C Properties

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

Additional properties are allowed.

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

2.2.1.3.1.1. MPEG_primitive_V3C.atlases

An array of atlases

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

2.2.1.3.1.2. MPEG_primitive_V3C._MPEG_V3C_CAD

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

- **Type:** MPEG_primitive_V3C._MPEG_V3C_CAD
- **Required:** No

2.2.1.3.1.3. MPEG_primitive_V3C.extensions

JSON object with extension-specific objects.

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

2.2.1.3.1.4. MPEG_primitive_V3C.extras

Application-specific data.

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

2.2.1.3.2. MPEG_primitive_V3C._MPEG_V3C_CAD

defines the common atlas data for a v3c object

Table 2. MPEG_primitive_V3C._MPEG_V3C_CAD Properties

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

Additional properties are allowed.

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

2.2.1.3.2.1. MPEG_primitive_V3C._MPEG_V3C_CAD.MIV_view_parameters

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

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

2.2.1.3.2.2. MPEG_primitive_V3C._MPEG_V3C_CAD.extensions

JSON object with extension-specific objects.

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

2.2.1.3.2.3. MPEG_primitive_V3C._MPEG_V3C_CAD.extras

Application-specific data.

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

2.2.1.3.3. MPEG_primitive_V3C.atlas

glTF extension to specify support for V3C compressed primitives.

Table 3. *MPEG_primitive_V3C.atlas Properties*

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

Additional properties are allowed.

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

2.2.1.3.3.1. MPEG_primitive_V3C.atlas._MPEG_V3C_CONFIG

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

2.2.1.3.3.2. MPEG_primitive_V3C.atlas._MPEG_V3C_AD

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

- **Type:** *integer*

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

2.2.1.3.3.3. MPEG_primitive_V3C.atlas._MPEG_V3C_GVD_MAPS

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

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

2.2.1.3.3.4. MPEG_primitive_V3C.atlas._MPEG_V3C_OVD_MAP

a reference to a video texture that provides the occupancy map

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

2.2.1.3.3.5. MPEG_primitive_V3C.atlas._MPEG_V3C_AVD

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

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

2.2.1.3.3.6. MPEG_primitive_V3C.atlas._MPEG_V3C_CAD

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

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

2.2.1.3.3.7. MPEG_primitive_V3C.atlas.extensions

JSON object with extension-specific objects.

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

2.2.1.3.3.8. MPEG_primitive_V3C.atlas.extras

Application-specific data.

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

2.2.1.3.4. MPEG_primitive_V3C.attribute

defines the attribute of a V3C object.

Table 4. MPEG_primitive_V3C.attribute Properties

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

Additional properties are allowed.

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

2.2.1.3.4.1. MPEG_primitive_V3C.attribute.type

provides the type of the attribute.

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

2.2.1.3.4.2. MPEG_primitive_V3C.attribute.maps

provides the references to the corresponding video texture maps.

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

2.2.1.3.4.3. MPEG_primitive_V3C.attribute.extensions

JSON object with extension-specific objects.

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

2.2.1.3.4.4. MPEG_primitive_V3C.attribute.extras

Application-specific data.

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

Following is an example illustrating the use of the syntax described in [Section 2.2.1.3.3](#)

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

2.2.2. References

- [1] m61138, "Support for multiple atlases for MIV application", MPEG 140, Mainz Meeting, October 2022.
- [2] WG7N00553, "Technologies under Consideration on Scene description", MPEG 141, Online, January 2023.
- [3] ISO/IEC 23090-5:2021 Information technology — Coded representation of immersive media — Part 5: Visual volumetric video-based coding (V3C) and video-based point cloud compression (V-PCC), Online, <https://www.iso.org/standard/73025.html>

Chapter 3. Interfaces

3.1. Supporting Multiple Viewers in the Media Access Function

Source: [m58510](#)

3.1.1. General

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

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

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

3.1.2. Proposed Updates to MAF API

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

3.1.2.1. Methods

Table 5. n/a

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

3.1.2.2. IDL for media pipeline

```

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

```

3.2. Generic API for Presentation Engine


Source: [m66705](#)

3.2.1. Generic Render Control API

The Generic Render Control API is an abstract API that is offered by external renderers to enable applications, such as Presentation Engines, to control the rendering process by aligning and synchronizing their rendering state to that of the Presentation Engine. This API is used by the Presentation Engine to configure and update the status of the external renderer.

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

Method	Description
init()	<p>Initializes the external renderer by providing the related media source information and their corresponding buffers. It also establishes a session between the Presentation Engine and the external renderer.</p> <p>The inputs to this method call should be:</p> <ul style="list-style-type: none">• A media source object that contains a handler to the buffer(s), where the source media will be made available by the MAF. A description of the media source and the contents of each buffer shall also be provided.

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

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

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

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

```

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

```

3.2.2. Extension for Audio Node Mapping

3.2.2.1. General

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

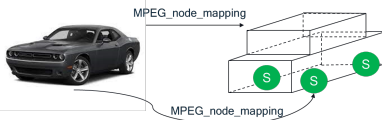


Figure 2. A black and white image of a camera Description automatically generated

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

3.2.2.2. Semantics

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

Name	Type	Default	Usage	Description
mappings	array(object)		M	An array of mappings associated with the containing node.
Role	string	“urn:mpeg:sd:role:default”	O	An identifier of the role associated with this mapping. The role may for instance be “urn:mpeg:sd:role:audio-renderer” to indicate that the component is an audio renderer.
source	number	N/A	M	The index in the <code>MPEG_media</code> that provides the media resource that contains the mapped element.
referenceId	number	N/A	M	An identifier of the element in the referenced resource.

Name	Type	Default	Usage	Description
transform	array(number)	Identity	O	A 4x4 matrix that supplies the transform used to align the referenced element to the current node.
supportsInteractivity	boolean	false	O	Indicates if interactivity actions applied to the node should be exposed if an API is made available to the Presentation Engine by the renderer of the resource.

3.2.2.3. Processing Model

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

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

Chapter 4. MPEG-I Audio in Scene Description

4.1. Immersive audio extension

Source: [m63549](#)

4.1.1. Introduction

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

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

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

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

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

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

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

4.1.2. Background

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

For a sake of illustration, a full VR experience is shown in [Figure 3](#) 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

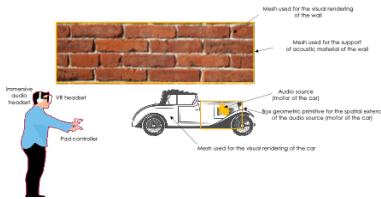


Figure 3. Virtual objects having dedicated representations for audio and visual renderers

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

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

4.1.3. MPEG-I immersive audio support

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

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

Table 6. Comparison between the different capabilities of MPEG_audio_spatial and the MPEG-I Audio solution

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

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

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

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

The following sections identifies new potential MPEG extensions at several levels of a glTF scene graph to support MPEG-I immersive audio as shown in [Figure 4](#) . Note that alternatively, a single extension, as is the case with MPEG_audio_spatial, might be defined instead.

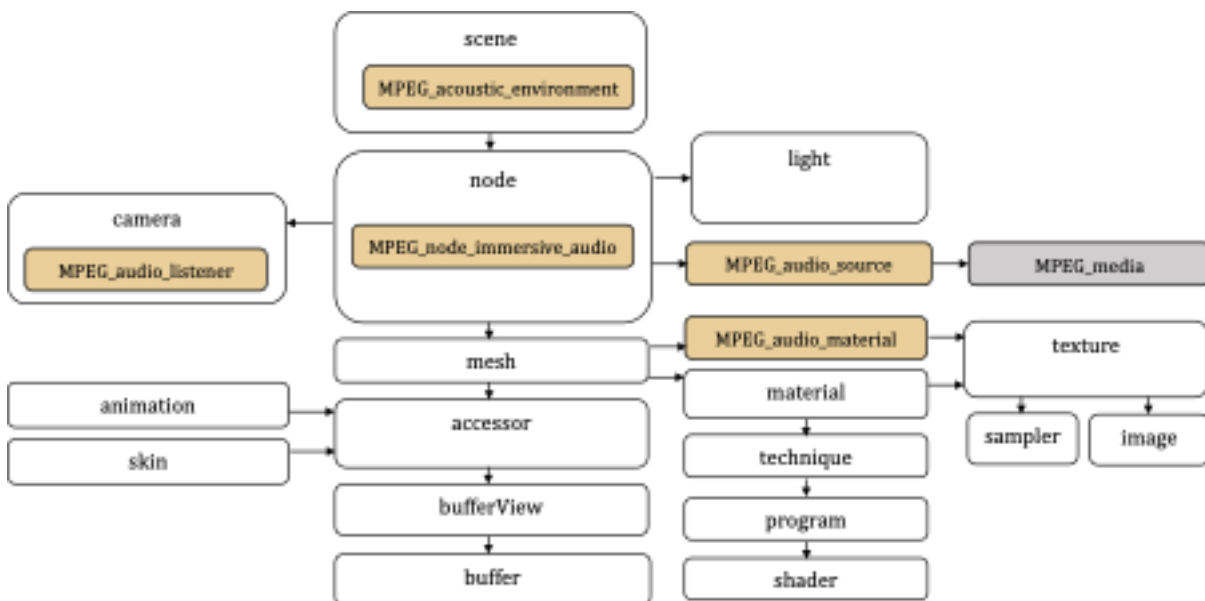


Figure 4. Proposed new MPEG glTF extensions to support MPEG-I immersive audio

4.1.3.1. Audio/acoustic data at global/scene level

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

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

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

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

4.1.3.2. Audio/acoustic data at node level

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

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

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

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

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

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

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

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

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

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

4.1.3.4. Audio/acoustic material data at mesh primitive level

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

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

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

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

4.1.4. References

[1] ISO/IEC 23090-14

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

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

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

4.2. MPEG-I Audio in Scene Description

Source: [m61180](#)

4.2.1. General

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

Figure 5 depicts the MPEG-I audio architecture:

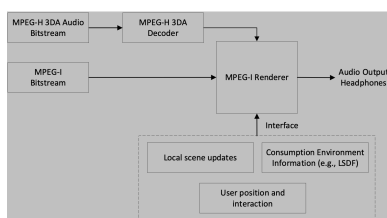


Figure 5. N/A

The MPEG-I render pipeline is depicted by [Figure 6](#):

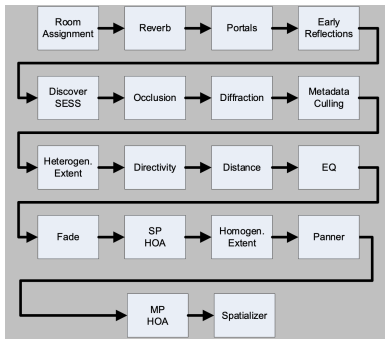


Figure 6. N/A

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 7](#):

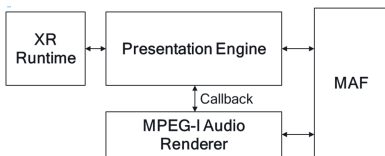


Figure 7. N/A

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 8](#):

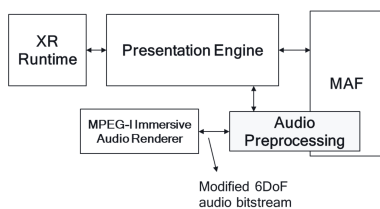


Figure 8. N/A

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.

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

Source: [m65378](#)

4.3.1. General

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

This approach is depicted by the following figure:

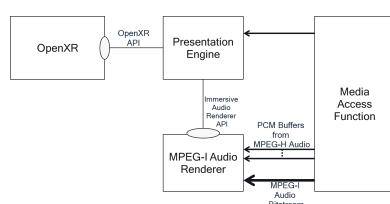


Figure 9. n/a

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

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

4.3.2. Extension for Audio Node Mapping

4.3.2.1. General

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

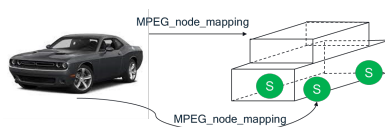


Figure 10. n/a

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

4.3.2.2. Semantics

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

table:

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

4.3.2.3. Processing Model

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

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

4.4. On spatial synchronization between graphs

Source: [m67011](#)

4.4.1. Attempt problem definition for the spatial synchronization

4.4.1.1. Virtual Reality (VR) use case

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

[Figure 11](#) provides the SD and the immersive audio graph representations of the virtual car.

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

The following node mappings have been created:

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

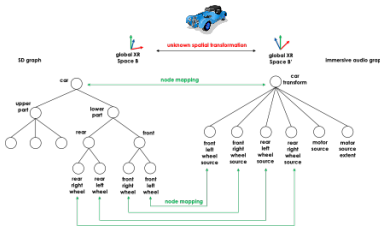


Figure 11. SD and immersive audio graph representations of a virtual car

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

- To allow nested spatial transformation nodes in the immersive audio graph [Figure 12](#)
 - The audio source and its extent would then be the children of a mapped spatial transformation node
- Or to allow the extent to be a child of the mapped audio source [Figure 13](#)

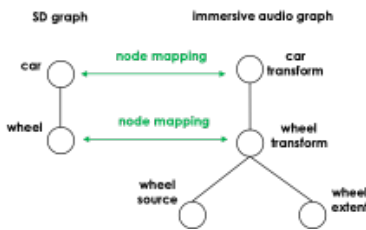


Figure 12. Possible approaches to ensure a spatial synchronization for both an audio source and its extent

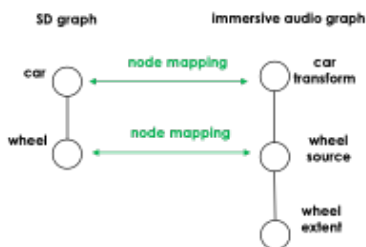


Figure 13. Possible approaches to ensure a spatial synchronization for both an audio source and its extent

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

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

4.4.1.2. Augmented Reality (AR) use case

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

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

The MPEG_anchor extension defines the Trackable and Anchor objects as follows (Figure 14):

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

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

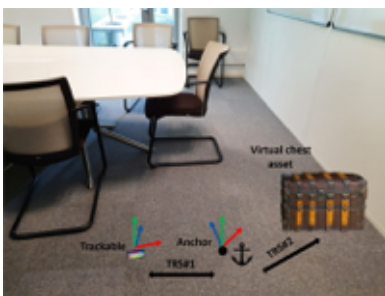


Figure 14. Trackable and Anchor for AR

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

Note 2: The immersive audio group uses a single Anchor object for the AR anchoring of the scene. This Anchor object corresponds to a Trackable object of an MPEG Scene Description. In other words, the transformation matrix between the Trackable and the Anchor objects (TRS#1 in Figure 14) is always the Identity matrix in the immersive audio graph.

Figure 15 illustrates the AR anchoring of the SD and immersive audio graphs representing the virtual car using a 2D marker by assuming that a common shared Trackable is defined in both the SD and immersive audio graphs.

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

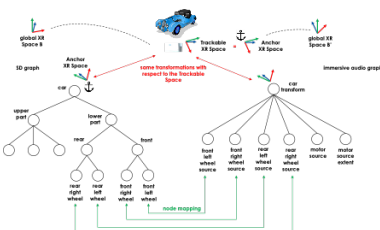


Figure 15. SD and immersive audio graph representations of a virtual car with AR anchoring using a 2D marker

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

OpenXR).

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

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

4.4.2. Approach proposal for the spatial synchronization

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

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

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

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

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

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

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

Figure 16 illustrates this matrix calculation process with:

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

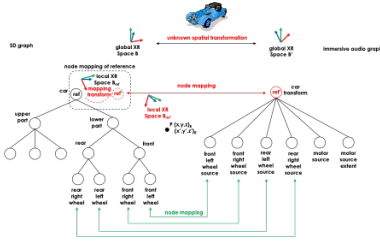


Figure 16. Transformation matrix determination using a node mapping

For any point P:

$$(x, y, z)_B = P_B^{Bref'} (x_{ref'}, y_{ref'}, z_{ref'})_{Bref'} = P_B^{Bref'} P_{Bref}^{Bref'} (x_{ref'}, y_{ref'}, z_{ref'})_{Bref'} \quad (2)$$

$$(x', y', z')_{B'} = P_{B'}^{Bref'} (x_{ref'}, y_{ref'}, z_{ref'})_{Bref'} \quad (3)$$

Then, with (2) and (3):

$$(x', y', z')_{B'} = P_{B'}^{Bref'} [P_{Bref}^{Bref'}]^{-1} [P_B^{Bref'}]^{-1} (x, y, z)_B \quad (4)$$

With (1) and (4):

$$P_{B'}^B = P_{B'}^{Bref'} [P_{Bref}^{Bref'}]^{-1} [P_B^{Bref'}]^{-1} \quad (5)$$

For a sake of clarity, the matrix product $[P_{Bref}^{Bref'}]^{-1} [P_B^{Bref'}]^{-1}$ may be called alignment matrix P_{align}

$$P_{align} = [P_{Bref}^{Bref'}]^{-1} [P_B^{Bref'}]^{-1} \quad (6)$$

And finally, with (5) and (6):

$$P_{B'}^B = P_{B'}^{Bref'} P_{align} \quad (7)$$

In formula (7), it has to be noted that:

- The Presentation Engine does not know the matrix $P_{B'}^{Bref'}$
- The immersive audio renderer does not know the alignment matrix P_{align} and which node ref' of the immersive audio graph has been used for the calculation of the transformation matrix $P_{B'}^B$

4.4.2.2. Parameters to be provided to the immersive audio renderer during the configuration step

The following parameters need to be provided to the immersive audio renderer during the configuration step:

- The alignment matrix P_{align} ,
- The unique identifier (i.e., the referenceId of the node mapping glTF extension of [1]) of the node of the immersive audio graph used for the calculation of the transformation matrix $P_{B'}^B$

By receiving the alignment matrix P_{align} and the referenceId of the node ref' , the immersive audio renderer can calculate and store the transformation matrix $P_{B'}^B$ using the formula (7).

Then, when receiving the initial poses of the mapped nodes and the Trackables expressed in the

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

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

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

4.4.3. Conclusion

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

4.4.4. References

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

Chapter 5. Interactivity framework

5.1. On event-based scene update

Source: [m61812](#)

5.1.1. 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.

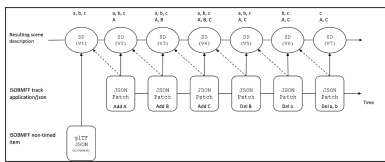


Figure 17. n/a

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.

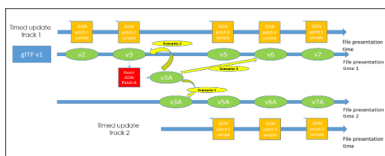


Figure 18. n/a

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.

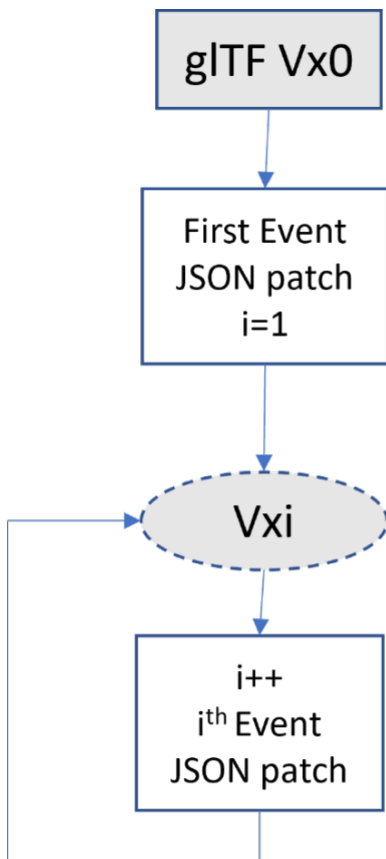


Figure 19. Event-based update diagram

5.1.2. A use case for event based updates

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

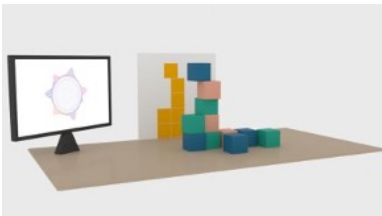


Figure 20. n/a



Figure 21. 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.

5.1.3. 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 5.1.2](#), 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...).

5.1.4. 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 5.1.2](#).

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

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

Table 7. Type of action

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

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

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

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

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

Chapter 6. Collected problem statements and industry needs

6.1. On the support of real environment data

Source: [m61811](#)

6.1.1. 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.

6.1.2. Representation of the real environment

As shown in [Figure 22](#), 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

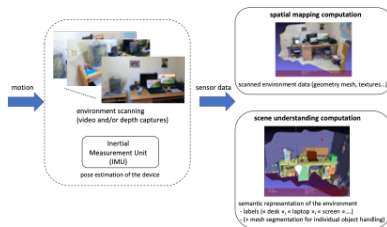


Figure 22. Computation of real environment data

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).”

6.1.3. 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 heterogenous 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 23, 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

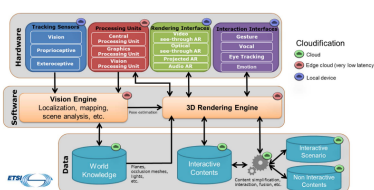


Figure 23. Global overview of the architecture of an AR system (from ETSI ARF)

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.

6.1.4. 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

6.1.4.1. 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 24](#)
- 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 25](#)

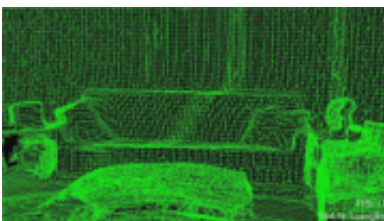


Figure 24. Mesh representation of the real environment after a spatial mapping computation



Figure 25. 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

```

6.1.4.2. 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 26](#), a semantic labeling is performed for the real objects that ARKit can identify such as ceiling, door, floor, seat, table, wall and window labels.

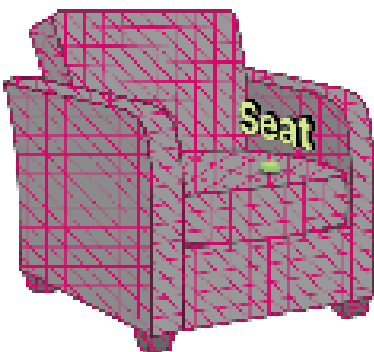


Figure 26. Semantic labeling of Apple's ARKit

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

serializing/deserializing an ARWorldMap as shown in [Figure 27](#).

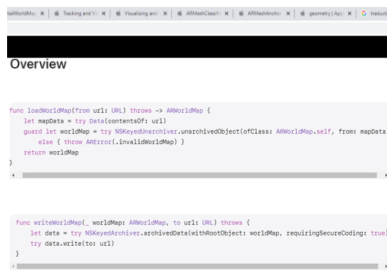


Figure 27. Saving and loading an Apple's ARKit ARWorldMap

6.1.4.3. 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 28](#).

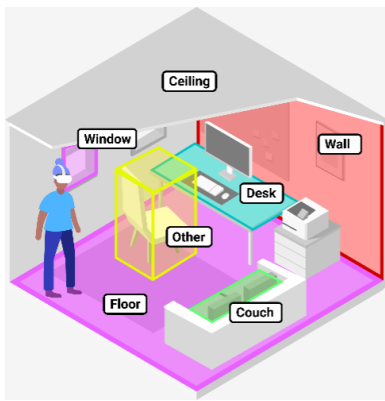


Figure 28. Semantic labeling of the Meta/Oculus Scene Understanding

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()
```

6.2. The support of XR Spatial Computing of real environment

Source: [m67595](#)

6.2.1. Introduction

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

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

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

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

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

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

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

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

6.2.2. Configuration examples of XR Spatial Computing

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

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

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

ARView.Environment.SceneUnderstanding.Options

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

static let default: ARView.Environment.SceneUnderstanding.Options

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

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

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

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

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

In the case of the computation of segmented meshes, the XR Spatial Computing is capable of providing a classification.

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

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

case window : The face is a part of a real-world window.

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

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

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

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

6.2.3. Approach proposal to support XR Spatial Computing

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

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

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

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

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

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

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

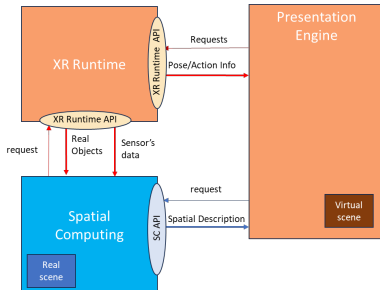


Figure 29. Proposed high-level architecture for the XR Spatial Computing support

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

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

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

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

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

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

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

Appendix A: Disclaimer



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



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