ISO/IEC JTC 1/SC 29/WG 03 N292

**ISO/IEC JTC 1/SC 29/WG 03  
MPEG Systems   
Convenorship: KATS (Korea, Republic of)**

**Document type:** Output Document

**Title:** Technologies under Consideration on Scene Description for MPEG Media

**Status:** Approved

**Date of document:** 2021-07-19

**Source:** ISO/IEC JTC 1/SC 29/WG 03

**No. of pages:** 55 (with cover page)

**Email of Convenor:** young.L @ samsung . com

**Committee URL:** <https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg3>

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 03 MPEG SYSTEMS**

**ISO/IEC JTC 1/SC 29/WG 03 N292**

**July 2021, Virtual**

|  |  |
| --- | --- |
| **Title** | **Technologies under Consideration on Scene Description for MPEG Media** |
| **Source** | **WG 03, MPEG Systems** |
| **Status** | **Approved** |
| **Serial Number** | **20546** |

# Contents

[Contents 3](#_Toc77580866)

[Introduction 6](#_Toc77580867)

[Phase 1 8](#_Toc77580868)

[1 Extensions 8](#_Toc77580869)

[1.1 MPEG\_Media extension control support (m56047) 8](#_Toc77580870)

[1.1.1 General 8](#_Toc77580871)

[1.1.2 Semantics 8](#_Toc77580872)

[1.1.3 JSON Syntax/Schema 11](#_Toc77580873)

[1.1.4 Processing Model 14](#_Toc77580874)

[1.1.5 Example 14](#_Toc77580875)

[1.2 Support for dynamic texture coordinates 15](#_Toc77580876)

[1.3 Texture transforms and compression 15](#_Toc77580877)

[1.4 MPEG Audio Extension (m55113, m55132) 15](#_Toc77580878)

[1.4.1 Overview 15](#_Toc77580879)

[1.4.2 Syntax and Semantics 16](#_Toc77580880)

[1.4.3 Example Audio Scene 19](#_Toc77580881)

[1.5 MPEG\_geo\_coordinates Extension 20](#_Toc77580882)

[2 Storage in ISOBMFF 22](#_Toc77580883)

[2.1 Carriage Format for dynamic scene information (m55453) 22](#_Toc77580884)

[2.1.1 Animation Sample Entry 22](#_Toc77580885)

[2.1.2 Dynamic Scene Info Sample Format 23](#_Toc77580886)

[2.2 Carriage Format for animation timing (m56039) 23](#_Toc77580887)

[2.2.1 Multiple animations 23](#_Toc77580888)

[2.2.2 Interaction of animation and dynamic 3D object 25](#_Toc77580889)

[3 Random Access Support 26](#_Toc77580890)

[3.1 m56249 (41.1)[SD] Solutions for Random Access Support 26](#_Toc77580891)

[3.1.1 Scene Update Mechanism 26](#_Toc77580892)

[3.1.2 Problem Statement 27](#_Toc77580893)

[3.1.3 Potential Solutions 27](#_Toc77580894)

[3.1.4 Criteria for Evaluation of the Solution 29](#_Toc77580895)

[3.2 m56323 [SD] On Random Access Support for Dynamic Scenes 29](#_Toc77580896)

[4 V3C Support 32](#_Toc77580897)

[4.1 m56240 [41] On V3C Support in Scene Description 32](#_Toc77580898)

[4.2 Partial Access in V-PCC (m56686) 33](#_Toc77580899)

[4.2.1 Consideration 33](#_Toc77580900)

[4.2.2 Proposal 36](#_Toc77580901)

[4.2.3 Proposed text 37](#_Toc77580902)

[5 Data formats 40](#_Toc77580903)

[5.1 m56102 (41.1) [SD] Support of glTF CBOR binary format 40](#_Toc77580904)

[5.1.1 Problem Statement 40](#_Toc77580905)

[5.1.2 Benefit of CBOR file/data format: 40](#_Toc77580906)

[5.1.3 CBOR data size comparison example: 40](#_Toc77580907)

[5.1.4 Use Cases 40](#_Toc77580908)

[5.1.5 Potential Solutions 41](#_Toc77580909)

[5.1.6 Open Issue Discussion 42](#_Toc77580910)

[6 Viewpoint based Dynamic Bitrate Adaption 42](#_Toc77580911)

[6.1 m56094 [SD] On DASH Dynamic Bitrate Adaption with Viewpoint Update 42](#_Toc77580912)

[6.1.1 Problem Statement 42](#_Toc77580913)

[6.1.2 Use Cases 42](#_Toc77580914)

[6.1.3 Current Scene Description Support and Gasps 44](#_Toc77580915)

[Phase 2 45](#_Toc77580916)

[1 Extensions 45](#_Toc77580917)

[1.1 MPEG\_scene\_anchor extensions (m56781) 45](#_Toc77580918)

[1.1.1 Introduction 45](#_Toc77580919)

[1.1.2 Semantics 46](#_Toc77580920)

[1.1.3 Examples 48](#_Toc77580921)

[1.2 MPEG\_camera\_control extension (m56337) 48](#_Toc77580922)

[1.2.1 General 48](#_Toc77580923)

[1.2.2 Semantics 49](#_Toc77580924)

[1.2.3 Processing Model 50](#_Toc77580925)

[1.3 MPEG\_mesh\_collision extension (m56337) 50](#_Toc77580926)

[1.3.1 General 50](#_Toc77580927)

[1.3.2 Semantics 50](#_Toc77580928)

[1.3.3 Processing Model 51](#_Toc77580929)

[1.4 MPEG\_node\_transformation\_external extension (m56440) 51](#_Toc77580930)

[1.4.1 General 51](#_Toc77580931)

[1.4.2 Semantics 52](#_Toc77580932)

[1.4.3 JSON syntax/schema 52](#_Toc77580933)

[1.4.4 Processing Model 53](#_Toc77580934)

[1.4.5 Example 53](#_Toc77580935)

[2 Interfaces 54](#_Toc77580936)

[2.1 CoAP API support in MAF (m56739) 54](#_Toc77580937)

[2.1.1 General 54](#_Toc77580938)

[2.1.2 MAF as CoAP Client 54](#_Toc77580939)

[2.1.3 MAF as HTTP-CoAP Proxy 55](#_Toc77580940)

[References 55](#_Toc77580941)

# Introduction

This document contains technologies under consideration on support of MPEG Media in Scene Description.

From MPEG#129, documents based on the following gitlab issues were covered:

* m52463 Timed Accessors
* m52838 MPEG media and MPEG video texture extensions for glTF2

From MPEG#130, documents based on the following gitlab issues were covered:

* [m53366 [SD] Real-time media support in scene description](http://mpegx.int-evry.fr/software/MPEG/SceneDescription/MPEG-Contributions/issues/10)
* [m53365 [SD] Basic support for Audio in scene description](http://mpegx.int-evry.fr/software/MPEG/SceneDescription/MPEG-Contributions/issues/9)
* [m53364 [SD] Scene Description Updates](http://mpegx.int-evry.fr/software/MPEG/SceneDescription/MPEG-Contributions/issues/8)
* [m53326 [SD] Support for Geographical Coordinate System](http://mpegx.int-evry.fr/software/MPEG/SceneDescription/MPEG-Contributions/issues/6)
* [m52973 On relationship among scene description, video decoding interface and file format to support partial access of V-PCC](http://mpegx.int-evry.fr/software/MPEG/SceneDescription/MPEG-Contributions/issues/4)

From MPEG#131, documents based on the following gitlab issues were covered:

* [m54312 [SD] Further Support for Dynamic Scene Updates](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/issues/31#note_9793)
* [m54249 [SD] On timed metadata support for Scene Description](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/issues/25#note_9799)
* [m54519 [SD] Media Access Function](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/issues/40#note_9804)
* [m54534 [SD]](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/issues/39) Streaming and Cloud Access of MPEG-I Scenes and Objects

From MPEG#132, documents based on the following gitlab issues were covered:

* [m54887 [SD] Further Aspects of Dynamic Scene Updates](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/issues/50#note_13477)
* [m55044 (39.1)[SD] Improvement on MPEG\_media extension](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/issues/61)
* [m54964 Alignment of glTF animation and MPEG media](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/issues/58)
* [m55183 [SD] Updates to Media Access Function](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/issues/65)
* [m55453 Updated Solution for Dynamic Scene Updates](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/issues/71)
* [m55113 [SD] Basic support for Audio in scene description](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/issues/62)
* [m55132 (39.1) [SD] Web friendly spatial audio integration](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/issues/63#note_17284)

From MPEG#132 following contributions/topics were moved to CD:

* m52973
* m54534
* m54519
* m55453
* m54312
* m54887
* m55453
* m52838, m55044 MPEG\_Media, MPEG\_texture\_video
* m52463 MPEG\_accessor\_timed
* m55113 MPEG\_audio
* m54249 MPEG\_recommended\_viewport
* m54964 MPEG\_animation\_time
* Media Access API
* Buffer API
* MPEG\_circular\_buffer

From MPEG#133, documents based on the following gitlab issues were covered:

* [[m55928/m56249](41.1)[SD] On Random Access Support for Scene Description with Scene Updates](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/-/issues/83)
* [[m56039][41.1] On timed animations for glTF](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/-/issues/88)
* [[m56040][41.1] On animations and interactivity for volumetric videos](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/-/issues/89)
* [[m56047][41.1] On MPEG Media Extension Signaling](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/-/issues/90)
* [[m56102] [SD] on Support of glTF CBOR binary](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/-/issues/95)
* [[m56240][41] On V3C Support in Scene Description](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/-/issues/96)
* [[m56094][SD] on DASH Dynamic Bitrate Adaptation with Viewport Update](http://mpegx.int-evry.fr/software/MPEG/Systems/SceneDescription/MPEG-Contributions/-/issues/94)

From MPEG#134, documents based on the following gitlab issues were covered:

* MPEG\_mesh\_linking extension (m56040) moved to DIS based on m56811
* Referencing real-time media moved to DIS based on m56780
* Carriage Format for glTF Objects moved to CD/DIS
* Carriage Format for glTF Object as non-timed item moved to CD/DIS
* Carriage Format for animation timing (m54964, part of m56039) based on m56812
* Carriage mesh correspondence values moved to DIS based on m56811
* m56323 On Random Access Support for Dynamic Scenes integrated in Phase 1
* m56781 Support for AR in Scene Description integrated in Phase 2
* m56337 Interactivity in Scene Description integrated in Phase 2
* m56739 On CoAP Support for IoT streaming devices in Scene Description integrated in Phase 2
* m56440 Nodes with External Transformation integrated in Phase 2
* m56323 On Random Access Support for Dynamic Scenes
* m56686 On V-PCC Partial Access Support for Scene Description

From MPEG#135, documents based on the following gitlab issues were covered:

* m57336 (42.1) [SD] Clarification of type of V-PCC track referenced from MPEG\_media
* m57410 [SD] Dynamic mesh support in scene description
* m57408 [SD] AR support in scene description
* m57409 [SD] Interactivity support in scene description

# Phase 1

# Extensions

## MPEG\_Media extension control support (m56047)

Proposal 1: It is proposed to support signaling more detailed playback control information about the MPEG media in MPEG\_media extension.

Currently in MPEG\_media extension a Boolean “controls” is signalled which has the semantics that it “specifies that media controls should be displayed (such as a play/pause button etc).”.

It is asserted that for MPEG-I scene description, a more detailed information should be allowed to be signaled in the MPEG-media extension to specify the supported playback control for the MPEG media.

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

The proposed modifications are shown highlighted below compared to SC29WG3\_N00026

### General

MPEG media extension, identified by MPEG\_media, provides an array of MPEG media items used in the scene.

If MPEG scene description is supported, then the MPEG\_media extension shall be supported. The MPEG media extension shall be included in the extensionsUsed and extensionsRequired of the scene description document for scene descriptions that require the use of MPEG media support.

The extension shall be declared at the top-level as follows:

{  
 "extensionsRequired": [  
 "MPEG\_media"  
 ]  
 "extensionsUsed": [  
 "MPEG\_media"  
 ]  
 }

### Semantics

The definition of all objects within MPEG\_media extension is provided in the tables below.

**Table 1 – Definitions of top-level objects of MPEG\_media extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| Media | Array | N/A | An array of items that list the media referenced by timed Accessors in a scene. |

**Table 2 – Definitions of item in the media array of MPEG\_media extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| Name | String | N/A | Label of the media. |
| renderingRate | Number | 25.0 | The renderingRate attribute is used to indicate the frequency at which the timed media is expected to be updated as frames per second. |
| startTime | Number | 0 | The startTime gives the time at which the rendering of the timed media will be in seconds. By default, the referenced image will be rendered as a static texture until the startTime. A startTime of 0 means the presentation time of the current scene.  Either startTime or autoplay shall be present in glTF description. |
| timeOffset | Number | 0 | The timeOffset indicates the time offset into the source, starting from which the timed media shall be generated. The value is provided in seconds, where 0 corresponds to the start of the source. |
| autoplay | Boolean | N/A | Specifies that the media will start playing as soon as it is ready.  Either startTime or autoplay shall be present in glTF description. |
| loop | Boolean | False | Specifies that the media will start over again, every time it is finished. |
| controls | ~~Boolean~~object | N/A | Specifies information about ~~that~~ media playback controls that the media supports as specified in Table 5 and that should be displayed ~~(such as a play/pause button etc)~~. |
| alternatives | Array | N/A | An array of items that indicate alternatives of the same media (e.g. different video code used)"  Note: the client could select items (i.e. uri and track) included in alternatives depending on the client’s capability. |

**Table 3 – Definitions of items in the alternatives array of MPEG\_media extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| mimeType | string | N/A | The MPEG media's MIME type.  Note : the profiles parameter, as defined in RFC6381, might be included as a part of the mimeType to specify the profile of the MPEG media container. (e.g. the profiles parameter indicates the DASH profile when the uri specifies a DASH manifest) |
| uri | string | N/A | The uri of the media. Relative paths are relative to the .gltf file. If the reference media is a real-time media stream, then the uri shall follow the referencing scheme defined in section 4.5. |
| tracks | Array | N/A | An array of items that lists the referenced tracks in them MPEG media container (e.g. mp4 file or DASH manifest). |

**Table 4 – Definitions of items in the tracks array of MPEG\_media extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| track | string | N/A | URL fragments to access the track within MPEG media.  e.g.  DASH : Using MPD Anchors (URL fragments) as defined in Annex C of 23009-1 (Table C.1).  MP4: URL fragments as specified in Annex L of 14496-12. |
| codecs | string | N/A | The codecs parameter, as defined in RFC 6381, of the media included in the track.  Note: When the track includes different types of codecs (e.g. the AdaptationSet includes Representations with different codecs), the codecs parameter could be signaled by comma-separated list of values of the codecs. |

**Table 5 – Definitions of controls object of MPEG\_media extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| pauseControl | boolean | true | pauseControl equal to true specifies that pause control should be displayed for the specified MPEG media.  pauseControl equal to false specifies that pause control should not be displayed for the specified MPEG media. |
| fastForwardControl | boolean | true | fastForwardControl equal to true specifies that a control which allows the user to fast forward on the media timeline of the specified MPEG media should be displayed.  fastForwardControl equal to false specifies that a control which allows the user to fast forward on the media timeline of the specified MPEG media should not be displayed. |
| fastBackwardControl | boolean | true | fastBackwardControl equal to true specifies that a control which allows the user to fast backward (reverse) on the media timeline of the specified MPEG media should be displayed.  fastBackwardControl equal to false specifies that a control which allows the user to fast backward (reverse) on the media timeline of the specified MPEG media should not be displayed. |

### JSON Syntax/Schema

|  |
| --- |
| {  "$schema": "http://json-schema.org/draft-04/schema",  "title": "Media",  "type": "object",  "description": "MPEG media used to create a texture, audio source, or any other media type defined by MPEG.",  "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],  "properties": {  "name": { },  "renderingRate": {  "type": "number",  "minimum": 0.0,  "exclusiveMinimum": true,  "default": 25.0,  "description": "The renderingRate attribute is used to indicate the frequency at which the timed texture is expected to be updated as frames per second."  },  "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. By default, the referenced image will be rendered as a static texture until the startTime. A startTime of 0 means the presentation time of the current scene."  },  "timeOffset": {  "type": "number",  "minimum": 0.0,  "default": 0.0,  "exclusiveMinimum": false,  "description": "The timeOffset indicates the time offset into the source, starting from which the timed texture shall be 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."  },  "loop": {  "type": "boolean",  "default": false,  "description": "Specifies that the MPEG media start over again, every time it is finished."  },  "controls": {  "type": "~~boolean~~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  }  },  "required": []  }  },  "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": [  {  "enum": [ "video/mp4" ]  },  {  "enum": [ "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 23009-1 (Table C.1). MP4: URL fragments as specified in Annex L of ISOBMFF."  },  "codec": {  "type": "string",  "description": "The codecs parameter, as defined in RFC 6381, of the media included in the track."  }  }  },  "required": ["uri", "mimeType" ]  },  "minItems": 1  }  },  "oneOf": [  { "required": [ "startTime" ] },  { "required": [ "autoplay" ] }  ]  } |

### Processing Model

Processing of the MPEG\_media extension depends on the referenced media.

### Example

In the example below, two media items are listed by MPEG\_media object. The first media item contains only one item within alternatives, which is a DASH manifest that contains one track. Even though there are no alternatives at the MPEG media level, DASH manifest may still have different Representations within the Adaptation Set (but this is outside of the scope of the extension). The second media item contains two items within the alternatives. The first one lists an mp4 file that contains data compressed using AVC codec, while the second one lists an mp4 file that contains data compress using HEVC codec. Each item within alternatives array has to have the same amount of track items within tracks object. However, each track item may contain different information, which depends on the structure of the MP4 file.

|  |
| --- |
| {  "extensions": {  "MPEG\_media": {  media: [  {  "name": "source 0",  "renderingRate": 25.0,  "timeOffset": 0.0,  "autoplay": "true",  "loop": "true",  "controls": ~~"false"~~{  "pauseControl": true,  "fastForwardControl": true,  "fastBackwardControl": false  },  "alternatives": [  {  "mimeType": "application/dash+xml",  "uri": "manifest.mpd",  "tracks": [  {  "track": "#track=1"  }  ]  }  ]  },  {  "name": "source 1",  "renderingRate": 30.0,  "startTime": 9.0,  "timeOffset": 10.0,  "loop": "true",  "controls": "false",  "alternatives": [  {  "mimeType": "video/mp4;codecs=\"avc1.42E01E\"",  "uri": "video1.mp4",  "tracks": [  {  "track": "#track\_ID=1"  },  {  "track": "#track\_ID=2"  }  ]  },  {  "mimeType": "video/mp4;codecs=\"hev1.1.6.L93.B0\"",  "uri": "video2.mp4",  "tracks": [  {  "track": "#track\_ID=3"  },  {  "track": "#track\_ID=1"  }  ]  }  ]  }  ]  }  }  } |

## Support for dynamic texture coordinates

UV coordinates may vary with every new frame of the timed texture. The texture coordinates for a primitive as referenced by TEXCOORD\_0 and TEXCOORD\_1 will be expected to change. Furthermore, the geometry of the primitive itself and by consequence the vertex indices may change as well. The dynamic changes are described through references to timed accessors.

## Texture transforms and compression

A single timed texture may be used to provide texture for multiple primitives. This operation is typical when a texture atlas is used. The transform is performed prior to the texture coordinate mapping. In order to support texture transforms, the KHR\_texture\_transform shall be supported.

The texture shall be provided in an internal format that is supported by the GPU at rendering time. A compressed texture format is preferable to reduce the memory usage on the GPU.

## MPEG Audio Extension (m55113, m55132)

[Ed. Note: During MPEG132 it was agreed that the subclause (but not the yellow highlights) will be moved to CD of 23090-14 and should be removed from TuC during MPEG133 meeting]

[Ed. Note: Text highlighted with yellow orginated from document m55132 and is a use case for the AHG mandate from MPEG 132 meeting]

### Overview

The MPEG audio extension adds support for spatialized audio to the MPEG scene description based on glTF 2.0. This extension is identified by “MPEG\_spatial\_audio”, which can be included at top level or attached to any node in the scene.

The MPEG\_spatial\_audio extension supports four different node types:

* AudioSource: an audio source that provides input audio data into the scene. Currently, only mono sources are supported.
  + 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.
* AudioReverb: A reverb effect can be attached to the output of an audio source. sceneSeveral 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
* AudioListener: An audio listener represents the output of audio in the scene. They are usually attached to camera nodes 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.
* AudioCluster: A pre-mixed selection of audio sources represented by a single AudioSource.

The following diagram depicts the processing chain for audio in a scene. Note that specification of any effects processing (green arrows) is out of scope for Scene Description:



Note that the characteristics of Audio Listener depend on the actual output devices available to the audio renderer.

### Syntax and Semantics

The following structure and properties are defined for the “MPEG\_audio”

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| sources | Array of audio sources |  | Provides a list of AudioSource elements that are attached to this node |
| id | number |  | unique identifier of the audio source in the scene. |
| type | string |  | “Object”, “HOA”, or “Cluster” |
| pregain | float | 0 | provides a level-adjustment in dB for the signal associated with the source. |
| playbackSpeed | float | 1 | defines the playback speed of the audio signal. A value of 1.0 corresponds to playback at normal speed. |
| attenuation | enumeration | linear | provides the function used to calculate the attenuation of the audio signal based on the distance to the source. An enumeration of predefined attenuation functions is defined. |
| attenuationParameters | Array of float |  | array of parameters that are input to the attenuation function. The semantics of these parameters depend on the attenuation function itself. |
| referenceDistance | float | 1 | provides the distance in meters for which the distance gain is implicitly included in the source signal after application of pregain.  Disregarded for HOA audio sources. |
| timedAccessor | Array of numbers |  | provides one or more pointers to timed accessors that will provide the audio data for this source. |
| reverbFeed | Array of id’s |  | If present: One or more pointers to reverb units, optionally extended by a floating point scaling factor. |
| listener | object |  | places an audio listener node in the scene that is attached to a parent camera node. The audio listener characteristics depend on the available audio output devices. |
| id | number |  | unique identifier of the audio listener in the scene. |
| reverb | Array of objects |  | list of audio reverb units that are defined at the top level of the scene. |
| id | number |  | unique identifier of the audio reverb unit in the scene. |
| bypass | boolean | true | indicates if the reverb unit can by bypassed if the audio renderer does not support it. |
| properties | object |  | contains reverb unit specific parameters |
| fdValues | Array of objects |  | Contains frequency dependent property values |
| frequency | float |  | Frequency [Hz] for the provided RT60 and DSR value. |
| RT60 | float |  | RT60 value [s] for the frequency provided in the ‘frequency’ field. |
| DSR | float |  | Diffuse-to-Source Ratio value [dB] for the frequency provided in the ‘frequency’ field. See explanatory text below. |
| predelay | number |  | Delay [s] from onset of source to onset of late reverberation for which DSR is provided. |
| clusters | Array of objects |  | Provides a list of AudioCluster elements that are attached to this node. |
| id | number |  | unique identifier of the audio cluster in the scene. |
| sourceId | numer |  | indicates the AudioSource id that represents the aggregated sources comprising this AudioCluster. |
| audioSources | Array of numbers |  | array of audio sources that are aggregated and represented by this cluster. |
| radius | float |  | indicates the distance of the encompassed aggregated audio sources. |

The 60 dB reverberation time, short RT60, is defined as the time it takes for the sound pressure level in a room to reduce by 60 dB, measured after a generated steady-state test signal is abruptly ended. It is defined for a specific frequency as an attribute RT60 and specified in seconds.

The pre-delay time indicates the delay between the emission at the source and the onset of the diffuse late reverberation part of a signal (i.e. the sound after the early reflections) and is specified in seconds. It is frequency-independent.

The Diffuse-to-Source-Ratio (DSR) specifies the level of the diffuse reverberation relative to the level of the total emitted sound. This can be determined while making an RT60 measurement. It is defined for a specific frequency as an attribute DSR and can be computed as follows:

For example, a value of 0 indicates direct sound only, while large values will describe an almost completely reverberant (wet) acoustic environment. Note that the DSR values do not influence the amplitude of the direct sound in the process of rendering. While DSR is a general description of a room’s acoustic properties, rendering reverberation using DSR requires taking into account the source’s directivity pattern to find the total emitted energy from the PCM signal’s reference level. DSR values are independent of directivity and may be determined with a source of any directivity, e.g. an omni-directional source. The total diffuse reverb energy denotes the reverberation energy at any point in the region for which the acoustic environment is defined, and is therefore directly linked to the PCM signal’s reference level.

### Example Audio Scene

The following shows an example of a scene with audio content:

|  |
| --- |
| "asset": {  "generator": "MPEG",  "version": "2.0"  },  "scene": 0,  "scenes": [  {  "nodes": [  0, 1  ]  }  ],  "nodes": [  {  "mesh": 0,  "children": [2]  },  {  "camera": 0,  "children": [3]  },  {  "extensions": {  "MPEG\_spatial\_audio": {  “source”: {  "id": 0,  "volume": 110,  "distance": 30,  "attenuation": "linear",  "attenuationParameters": [-5.0],  "timedAccessors": 0  }  }  }  },  {  "extensions": {  "MPEG\_spatial\_audio": {  "listener": {  "id": 0  }  }  }  }  ],  "extensions": {  "MPEG\_spatial\_audio": {  "effects": {  "id": 0,  "effecttype": "urn:vender:effect:example-effect",  "bypass": true,  "properties": {  …  }  }  },  "MPEG\_media": {  "media": [  {  "name": "audio\_source\_1",  "loop": true,  "alternatives": [  {  "mimeType": "audio/aac",  "uri": "https://example.com/audio\_source\_0.aac"  }  ]  }  ]  }  }  } |

## MPEG\_geo\_coordinates Extension

Khronos glTF2 format [3] currently adopts the Cartesian x-y-z coordinate system. Following the Cartesian coordinate system, the translations of the various objects is specified through the nodes object. In particular, a node can have either a matrix or any combination of translation/rotation/scale (TRS) properties. TRS properties are converted to matrices and post-multiplied in the T \* R \* S order to compose the transformation matrix; first the scale is applied to the vertices, then the rotation, and then the translation.

In common graphics APIs like OpenGL, Direct3D, or Vulkan, the vertex positions of 3D objects are usually stored as single-precision (32-bit) floating-point values. Rendering objects over large distances is common for geospatial programs, and when done incorrectly, the objects may visually jitter. The problem becomes more noticeable as the viewer nears the object. Due to this limited precision, objects that have a large distance to the origin of the rendering coordinate system cannot be represented accurately.

The proposed glTF extension introduces the metadata required to implement the Relative To Center (RTC) high-precision rendering technique in order to address this problem. In this technique, each position is defined relative to an origin (the center) such that 32-bit floating-point precision is adequate to describe the distance between each position and the center. These relative positions are stored in the glTF vertex data. At runtime, the positions are transformed with a modified model-view matrix that makes the center relative to the eye. This avoids 32-bit subtraction of large translation components on the GPU. Further details can be found in [9-10].

The RTC center based high-precision rendering is used as part of OGC 3D Tiles specification [5]. OGC 3D Tiles is an open specification for sharing, visualizing, fusing, interacting with, and analyzing massive heterogeneous 3D geospatial content across desktop, web, and mobile applications. 3D Tiles is built on glTF for efficient streaming and rendering of 3D models and scenes.

The RTC center provides the coordinates of the origin an array of three numbers (x, y, z) in Cartesian coordinates. Accordingly, all point positions are treated as relative to the value of this center. In addition, the extension also contains additional parameters to provide the geographical coordinates associated with the origin toward fulfilling the associated MPEG-I Phase 2 requirement.

MPEG extension, identified by MPEG\_geo\_coordinates, serves as an extensions property of the top-level glTF object. If MPEG\_geo\_coordinates is supported as part of the MPEG scene description, then MPEG\_geo\_coordinates shall be included in the extensionsUsed and extensionsRequired of the scene description document for scene descriptions.

The extension shall be declared at the top-level as follows:

{

"extensionsRequired": [

" MPEG\_geo\_coordinates"

]

"extensionsUsed": [

" MPEG\_geo\_coordinates"

]

}

The definition of the items within MPEG\_geo\_coordinates extension is provided in the table below.

**Table 1** Definitions ofitem in the media array of MPEG\_geo\_coordinates

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| RTC\_x | Number | 0 | Indicates the x-coordinate of the RTC center |
| RTC\_y | Number | 0 | Indicates the y-coordinate of the RTC center |
| RTC\_z | Number | 0 | Indicates the z-coordinate of the RTC center |
| Longitude | Number | 0 | Indicates the longitude of the geolocation of the center in units of 2−23 degrees. The value shall be in range of −180 \* 223 to 180 \* 223 − 1, inclusive. Positive values represent eastern longitude and negative values represent western longitude. |
| Latitude | Number | 0 | Indicates the latitude of the geolocation of the center in units of 2−23 degrees. The value shall be in range of −90 \* 223 to 90 \* 223 − 1, inclusive. Positive value represents northern latitude and negative value represents southern latitude. |
| Altitude | Number | 0 | Indicates the altitude of the geolocation of the center in units of milimeters above the WGS 84 reference ellipsoid. |

An example is shown as follows:

{

"extensions": [

"MPEG\_geo\_coordinates": [

"RTC\_x": 6378137,

"RTC\_y": 0,

"RTC\_z": 0,

"longitude": -1509949440,

"latitude": 0,

"altitude": 7137000,

]

}

]

}

# Storage in ISOBMFF

## Carriage Format for dynamic scene information (m55453)

### Animation Sample Entry

#### Definition

Sample Entry Type: 'dsce'  
Container: Sample Description Box ('stsd')  
Mandatory: No  
Quantity: 0 or 1

The sample entry for dynamic scene update information associated to the JSON patch document format is defined by the DynamicSceneInfoSampleEntry. Each sample carries the set of attributes from **Error! Reference source not found.**.

#### Syntax

aligned(8) class DynamicSceneInfoSampleEntry()

extends MetadataSampleEntry ('dsce') {

}

### Dynamic Scene Info Sample Format

#### General

#### Syntax

aligned(8) class DynamicSceneInfoSample() {

unsigned int(5) version\_id;

unsigned int(5) event\_id;

unsigned int(6) reserved;

unsigned int(32) absolute\_time\_utc;

unsigned int(32) absolute\_time\_tai;

}

#### Semantics

The corresponding semantics is as follows:

version\_id identifies the version of the corresponding sample carrying the JSON patch document with the dynamic scene update.

event\_id identifies the event triggering the dynamic scene update carried in the corresponding sample containing the JSON patch document.

absolute\_time\_utc wall clock time identifying the execution time of the scene update transaction on the glTF object based on the corresponding sample carrying the JSON patch document with the dynamic scene update. The value is denoted in UTC.

absolute\_time\_tai wall clock time identifying the execution time of the scene update transaction on the glTF object based on the corresponding sample carrying the JSON patch document with the dynamic scene update. The value is denoted in TAI.

## Carriage Format for animation timing (m56039)

### Multiple animations

#### 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

#### 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];

}

}

}

#### 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.

### Interaction of animation and dynamic 3D object

#### 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.

#### 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];

}

}

}

#### 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.

# Random Access Support

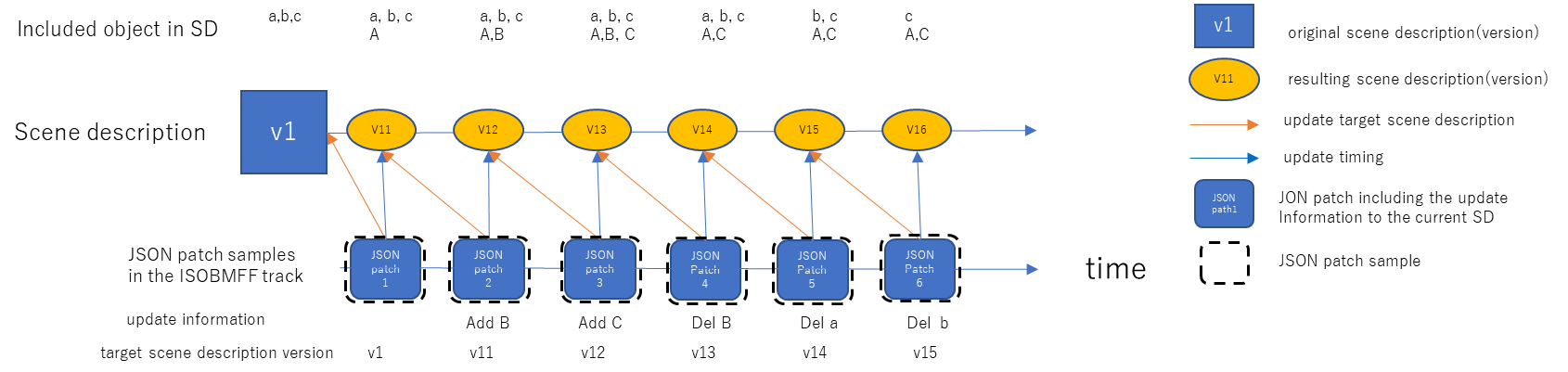
## m56249 (41.1)[SD] Solutions for Random Access Support

### Scene Update Mechanism

First of all, the method of scene update defined in the current CD is explained.**Figure 1** depicts original scene description (i.e. initial scene description provided to the client) and the JSON patch documents which provide update information for the current scene description. The current scene description means the scene description which has been in client’s memory at the time the update information is received. The current scene description is obtained by applying all previous update information to the original scene description.

Once the client applies the update operation to the current scene description, the resulting scene description is stored in its memories. The following update is applied to this resulting scene description.

In this way, update operations are performed sequentially.



**Figure 1 The current scene update method in CD**

### Problem Statement

In this method, if the client processes the content from the beginning, the update to the scene description can be applied, because the update information is for the current scene description and the client has maintained the current scene description in its memory.

On the other hand, if the client processes the content from the middle (i.e. random access), the update to the scene description may not be applied. Because the update information is for the current scene description and the client has not maintained the current scene description in its memory except the original scene description.

If the client acquires all of the update information from the beginning of the content up to the random access point, it can get the current scene description, so the update to the current scene description can be applied. But this process is inefficient.

The reason why this problem is caused is that the provided update information can only be applied to the current scene description.

### Potential Solutions

#### Using the update information for the original Scene Description

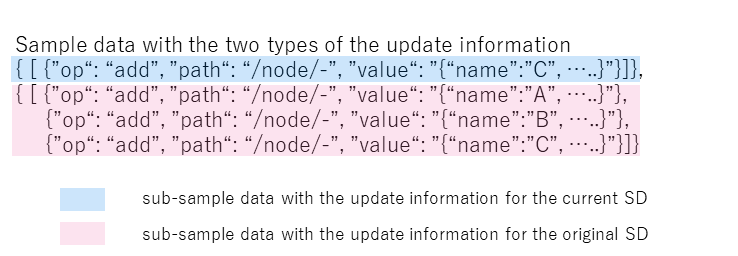
This solution is that the update information for the original scene description is provided for random access in addition to the update information for the current scene description.

#### Using sub-sample

This method allows sample to have two types of update information.

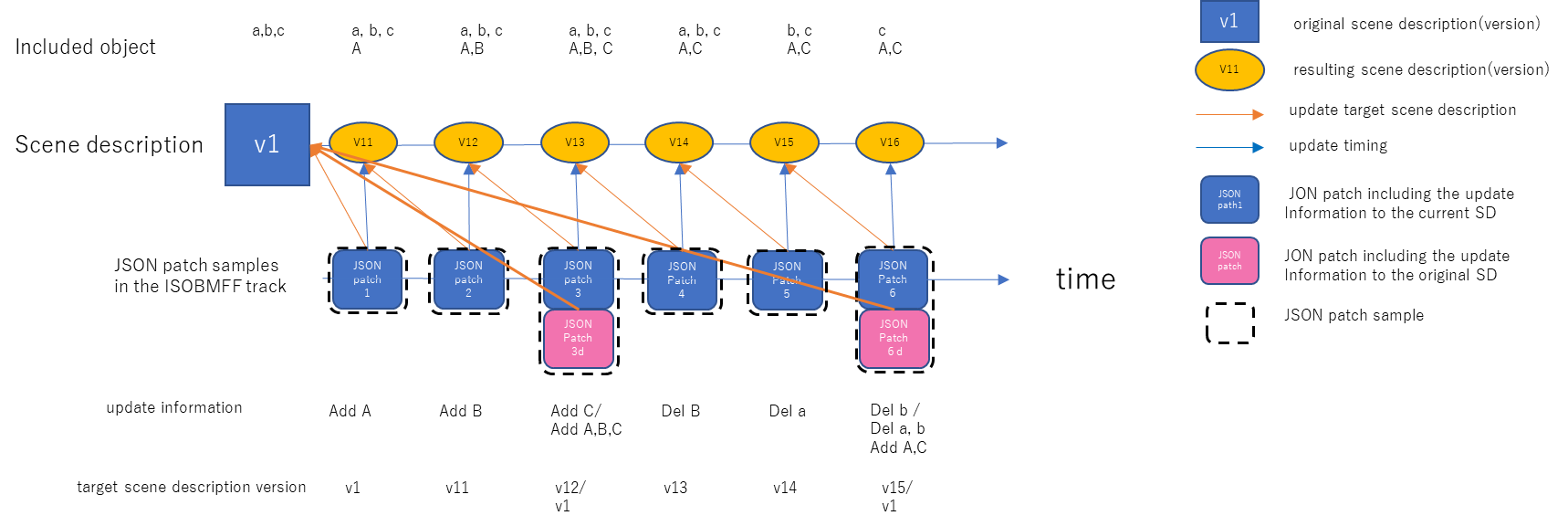
More specifically:

* To allow the sample to include either one or both of two types of the update information as a sub-sample, one is for the current scene description and the other is for the original scene description(**Figure 2**).
* To use the codec\_specific\_parameters in SubSampleInformationBox to identify the update information included in the sub-sample.
* Each sample which includes the update information for the original scene description is marked as a sync-sample.
* in current CD, it is described that all samples are marked as sync-samples, hence we have to change this description.



**Figure 2 Example for the sample data containing two types of update information**

The concept of the proposed signaling method is depicted in **Figure 3**.



**Figure 3 Conceptual diagram of the solution using sub-sample**

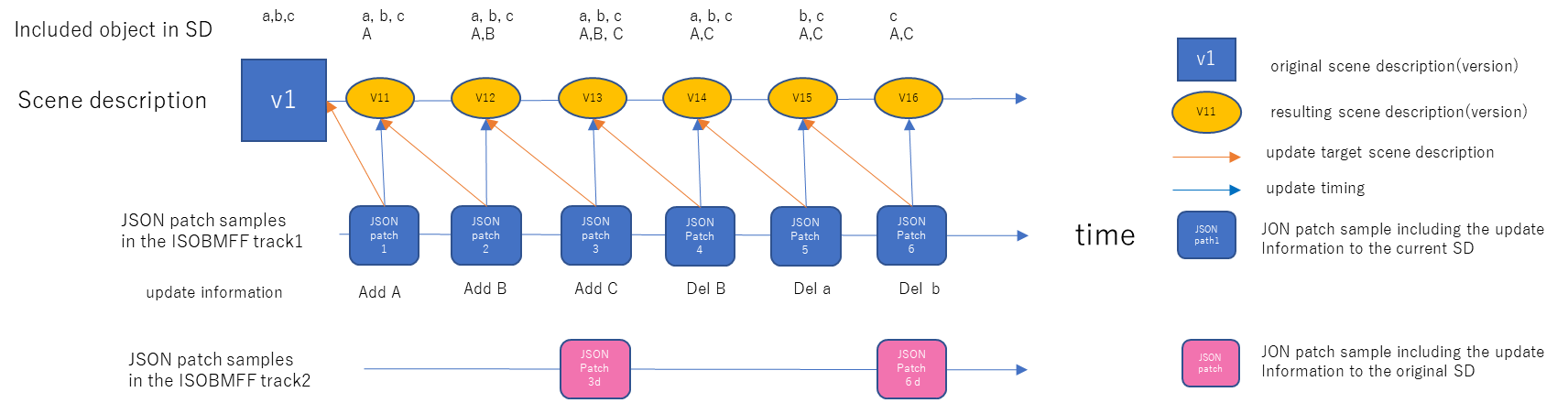
The client process for the scene update at the random access is as follows:

* The scene description document is provided to the client.
* The client parses the scene description and confirms if there is the track which includes the JSON patch samples.
* The client gets the sync sample, that is the closest to the desired start time of the process, from the track.
* By using the information of SubSampleInformationBox, the client gets the sub-sample which includes the update information for the original scene description.
* The client applies the update to the original scene description, gets the latest scene description and renders a 6DoF presentation.
* After that, client retrieve only the update information for the current scene description (ignoring the update information for the original scene description) and apply it to the current scene description.

#### Using two tracks

This method allow two tracks to provide each type of update information individually.

The concept of the signaling method is depicted in **Figure 4**

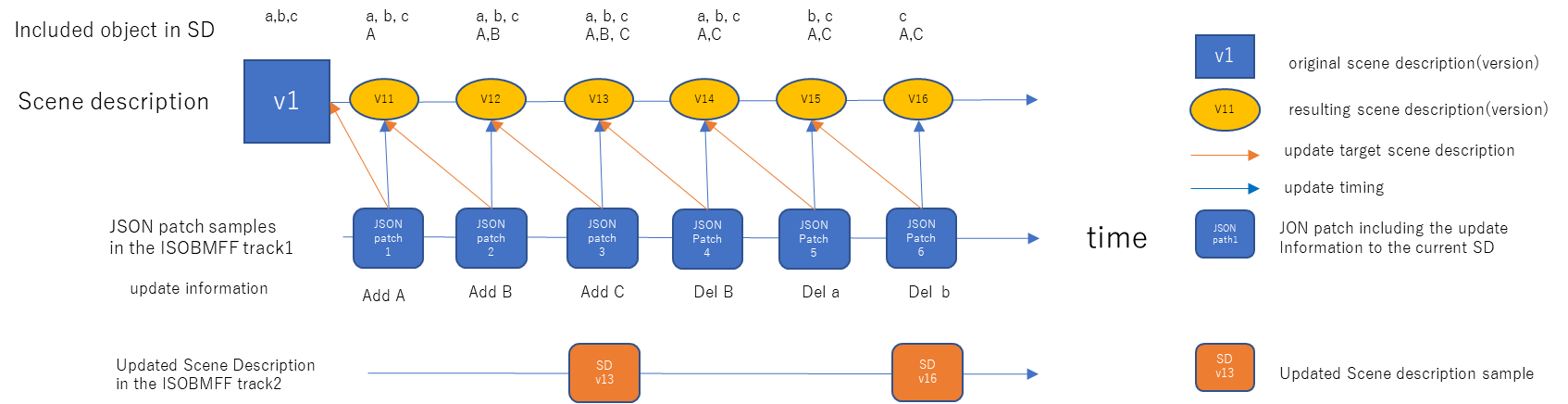


**Figure 4 Conceptual diagram of the solution using two tracks**

#### Using the updated Scene Description

This solution is that the updated scene description itself is provided in another track in order to random access.

The concept of the signaling method is depicted in Figure 5

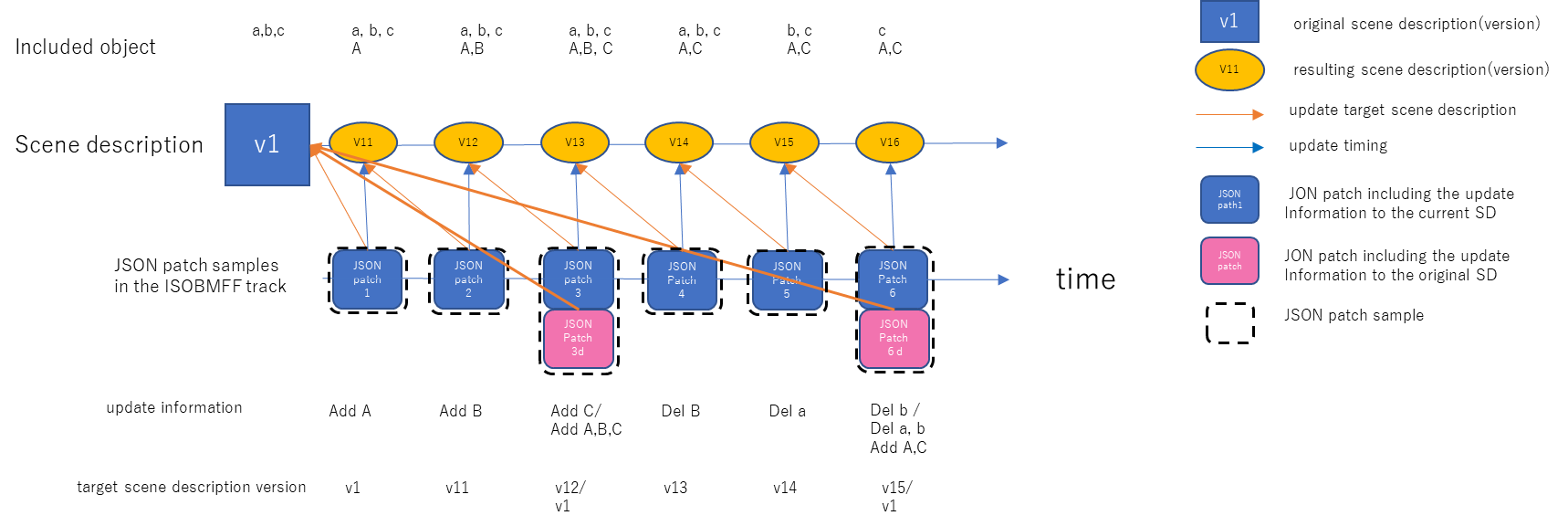


**Figure 5 Conceptual diagram of the solution using updated SD**

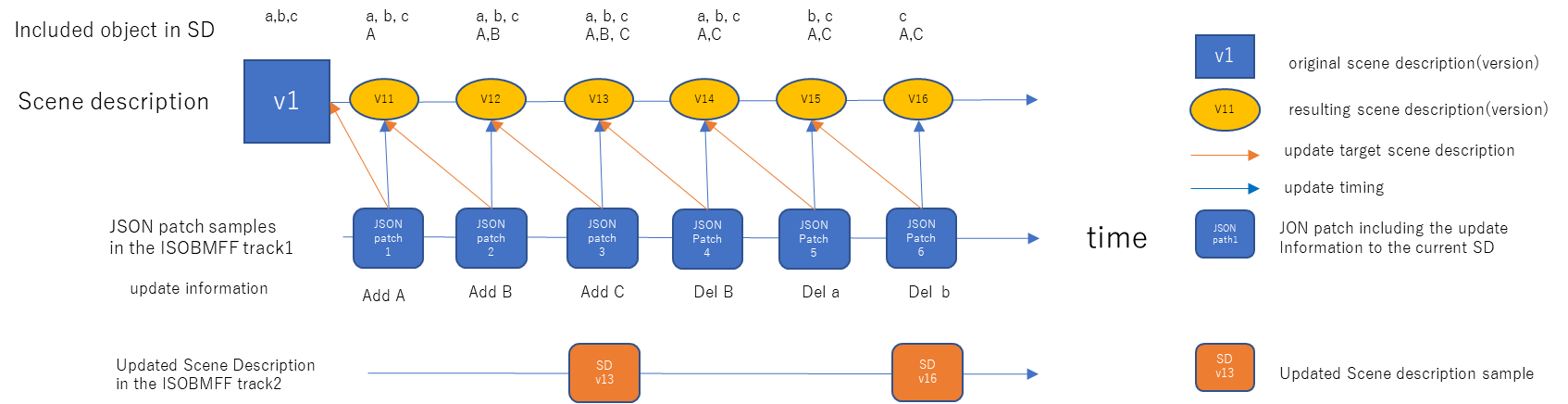
### Criteria for Evaluation of the Solution

* Size of the JSON document
* Simplicity of file structure and client implementation
* Startup delay
* Processing model
* Document management in client and server

## m56323 [SD] On Random Access Support for Dynamic Scenes



**Figure 6 Option A: Using the update information for the original Scene Description:**



**Figure 7 Option B: Using the updated Scene Description:**

**Table 5**presents the pros and cons of Options A and B, considering the evaluation criteria described in the TuC.

**Table 5 – Comparison of Options A and B for Random Access in Dynamic Scenes**

|  |  |  |
| --- | --- | --- |
|  | Option A: Dynamic scenes using the update information for the original Scene Description | Option B: Dynamic Scenes using the updated Scene Description |
| Random Access Framework | Client can join the scene at any given time by downloading the original scene description (‘gltf’ or ‘glti’) for the dynamic component of the scene, JSON patch (‘scen’) of pink type corresponding to the closest time plus JSON patches of blue type needed to reconstruct the glTF scene description for the current time. It is assumed that the fixed or static portions of the scene can be accessed as an item and therefore the corresponding description (‘glti’) is downloaded only once. | Client can join the scene at any given time by downloading the scene description (‘gltf’) for the dynamic component of the scene corresponding to the closest time plus JSON patches (‘scen’) of blue type needed to reconstruct the glTF scene description for the current time. It is assumed that the fixed or static portions of the scene can be accessed as an item and therefore the corresponding description (‘glti’) is downloaded only once. Accordingly, the ‘gltf’ track would carry the dynamic components of the scene that will vary over time. This framework is similar to the group-of-pictures (GOP) framework, where ‘gltf’ track carries the I-frames (e.g., IDR, CRA, etc.) to be utilized for random access and JSON patch documents resemble the P-frames. |
| Cumulative Size of the JSON documents | If the dynamic scene size is growing over time (i.e., original scene description is much smaller in size compared to the latest scene description), JSON patches of pink type are expected to grow in size for later times in comparison with the original description. Thus, no storage savings are achieved over Option B.  If the scene size is fixed over time (i.e., original scene description is about the same size as the latest scene description), there is a bandwidth penalty of downloading the JSON patches of pink type. | If the dynamic scene size is growing over time (i.e., original scene description is much smaller in size compared to the latest scene description), scene descriptions for the later times are expected to grow in size compared to the original scene description. However, this size should be similar to (or perhaps less than) the size of the sum of the original scene description and corresponding JSON patch of pink type.  If the scene size is fixed over time (i.e., original scene description is about the same size as the latest scene description), Option B will be more efficient compared to Option A, since there will be no download of JSON patches of pink type.  In the case of large, mostly unchanging scenes, Option A may be more efficient than Option B. However, static or immutable scene elements should be described using ‘glti’, eliminating the need to signal any updates to those elements and making Option B more efficient. |
| Simplicity of file structure and client implementation | Works with both track-based and item-based carriage of the glTF object corresponding to the dynamic component of the scene (i.e., ‘gtlf’ or ‘glti’) for random access, but requires additional track-based carriage of JSON patches of the pink type (in addition to those for the blue type)  For each random access, two kinds of computations are required: 1) Application of the JSON patch of the pink type to the original scene description, 2) Application of the JSON patch(es) of the blue type to the result of 1) | Requires track-based carriage of the glTF object corresponding to the dynamic component of the scene (‘gltf’) for random access. In case of large mostly unchanging scenes, using item-based scene description ‘glti’ is also desirable to represent static components of the scene.  Carriage of JSON patches of the pink type is not required. Only carriage of JSON patches of the blue type is required for random access  For each random access, only one kind of computation is required: 1) Application of the JSON patch(es) of the blue type to the latest scene description |
| Startup delay | Larger than Option B since additional download time is required to access JSON patch documents of the pink type, and further computational steps are required for each random access as outlined above. | Less than Option A since JSON patch documents of the pink type do not need to be downloaded and computations required for each random access are simpler as outlined above. |
| Document management in client and server | More complex than Option B since 4 different types of data needs to be managed: i) Carriage of glTF objects as timed data via brand 'gltf' or as non-timed data via brand 'glti' for the initial dynamic component of the scene, ii) Carriage of JSON patch documents of the blue type as timed data via brand 'scen', iii) Carriage of JSON patch documents of the pink type based on solutions outlined in the TuC | Simpler than Option A, since Option B only depends on the existing carriage formats in ISO/IEC 23090-14, and does not require definition of new track types. In particular, Option B requires: i) Carriage of glTF objects as timed data via brand 'gltf' for the dynamic component of the scene, ii) Carriage of glTF objects as non-timed data via brand 'gltf' for the static component of the scene, iii) Carriage of JSON patch documents of the blue type as timed data via brand 'scen’ |

# V3C Support

## m56240 [41] On V3C Support in Scene Description

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.

The following diagram shows different ways of creating media pipelines for the processing and rendering of V3C compressed data.



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:

1. Define specific primitive attribute for each component (see contribution m54514)
2. 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.

## Partial Access in V-PCC (m56686)

### Consideration

**Background**

Fig. 1 is shown on the CD**Error! Reference source not found.** to explain different usage examples of pipeline.

Both Pipeline#1 and #2 can branch into Option#1 or #2 when handing over the data to the Presentation Engine.

In Option#1, point cloud reconstruct processing is performed by the MAF, and point cloud data is stored in the buffer.

In Option#2, point cloud reconstruct processing is performed by the PE, and V-PCC decoded component video data is stored in the buffer.

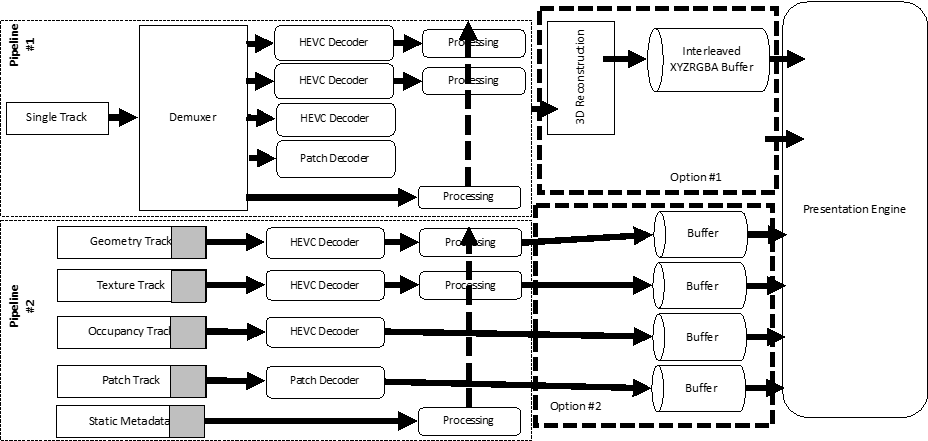


Fig. 1 Media Pipelines in Scene Description

Fig. 2 shows a diagram of the signal method of the scene description in each Option case. As for Option#2, one of the signal method which is being considered to specify the V-PCC specifc attribute data in section 6.1 in TuC**Error! Reference source not found.** is depicted.

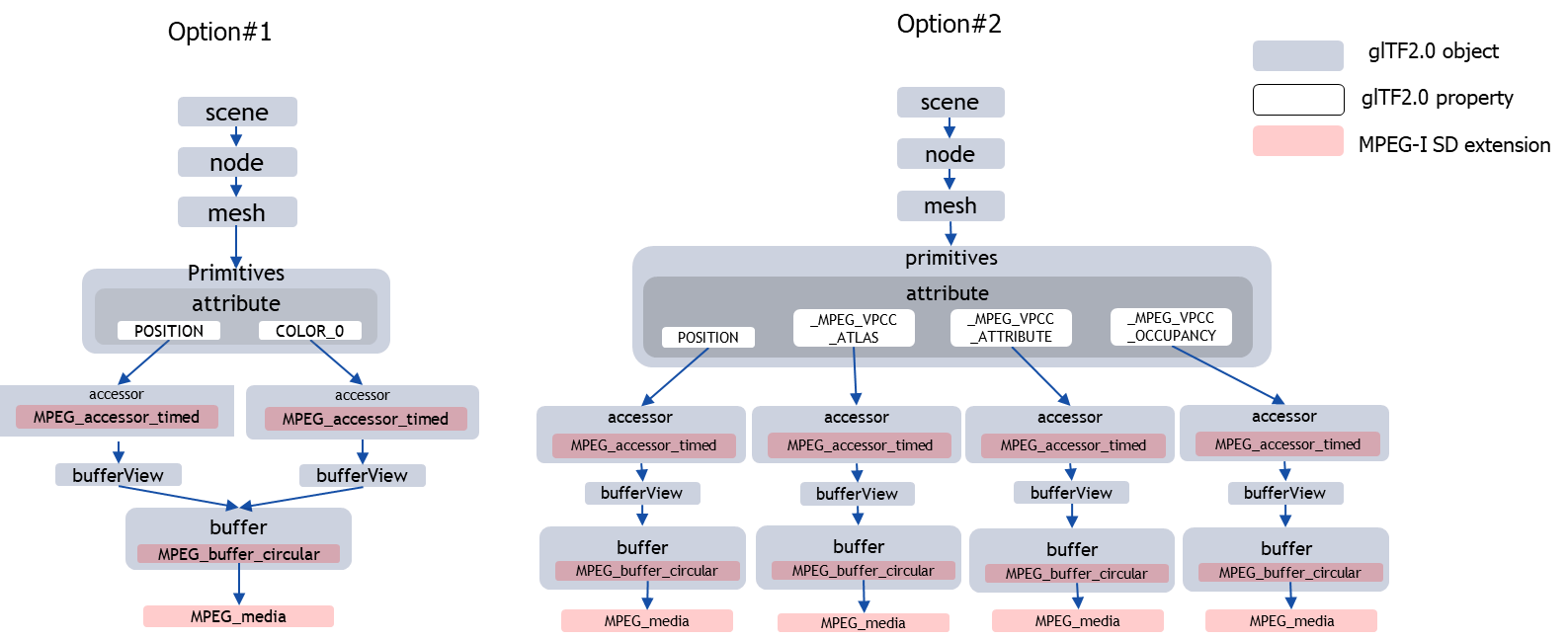


Fig. 2 diagram of the signal method of the scene description

Option#1:

The mesh.primitives.attributes has position and color attribute and sets the mesh.primitives.mode to pure vertices (points).

These accessors point to the buffer via bufferViews, and MPEG\_buffer\_circular points to the MPEG\_media source which includes V-PCC data.

Option#2:

The mesh.primitives.attributes has position and three new attributes and sets the mesh.primitivies.mode to pure vertices (points).

**Assumption**

1. partial access processing

There are two types of partial access processing. One is partial access by MAF and the other is partial access by PE.

As partial access by MAF utilize the bandwidth effectively, this contribution describes this case.

1. How to use buffer/accessor

There are two types of tile assignment to buffer/accessor. One is one buffer/accessor per tile, and the other is one buffer/accessor for multiple tiles.

case1) one buffer/accessor per tile

case2) one buffer/accessor for all acquired tile (my proposal)

・Acquired tile means the tiles which MAF acquires partially by using node position information, user’s viewport information, and spatial region information (from　V3CSpatialRegionsBox etc.)

(To simplify the discussion, it is excluded to consider each component in the tile)

Scene Description size aspect

case1)SD has as many buffer/accessor as there are tiles.

case2)SD has only one buffer/accessor for tiles.(SD size is smaller)

Client processing load aspect

case1)As SD has some buffer/accessor, there are same number of

MPEG\_timed\_accessor. So, client has to write and read the “timed

accessor information header” field for all tiles whether or not tile is

acquired.

case2)as SD has only one buffer/accessor for tiles, there is only one

MPEG\_timed\_accessor. So client has to write and read the “timed

accessor information header” field for only one file.(client processing

load is lower)

Simplicity of the signaling aspect

case1)SD has as many buffer/accessor as there are tiles. This is different

from the signaling method of V-PCC partially inaccessible file.

case2)SD has only one buffer/accessor for tiles. This is the same with the

signaling method of V-PCC partially inaccessible file.(more simple)

For the above reasons, I prefer to use case2, and this contribution describes case2.

1. Processing flow

The flow for MAF to store data in buffer is as follows:

1. The scene description signals the association of the 3D object with the related V-PCC file.(I think V3C atlas track is suitable for this.)
2. PE pass this V3C atlas track uri/track, node position information(a), user viewport information(b), and buffer information(c) to MAF by using API.
3. MAF acquires the V3C atlas track and analyze it's V3CspatialRegionBox to get th information of relations between spatial region and tile(d).
4. 4)MAF partially acquires tiles data by using(a), (b), (d).
5. 5)MAF put acquired tiles into the buffer using(c)

**Consideration**

We consider whether these two signal methods(Fig. 2) work in case of dealing with partially accessible V-PCC data files in the MPEG-I scene description.

In case that point cloud reconstruct processing is performed by the MAF.

The MAF partially acquires some tiles data, decodes them and reconstructs 3D object from these, and stores the reconstructed 3D object data in the buffer.

The PE could read the 3D object data from the buffer and renders it properly.

Thus, the partial access does work without any updates on the current specification text.

In case that point cloud reconstruct processing is performed by the PE.

The MAF acquires some tiles data partially, decodes them and stores them into a buffer. The buffer which is accessed by an accessor includes some interleaved tiles data(Fig.3).

The PE can read the data with the accessor, but it cannot recognize the interleaved tiles data as separate tile data, because PE cannot recognize the boundaries of the tile data.

Recognition the interleaved tiles data as separate tile data is necessary to reconstruct 3D object from each tile data.

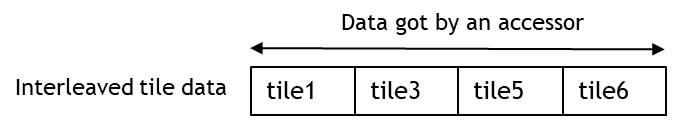


Fig. Interleaved tile data which is got by an accessor

### Proposal

Hence, in this section we propose how to signal the boundaries of the tile data.

It is proposed to specify the field to identify the boundaries of the tile data in the “timed accessor information header”.

Fig.4 depicts the added fields in “timed accessor information header”.

“acquiredNumOfTile”, tileId” and “countPerTile” are defined additionally.

And in order to indicate if these added field are signaled in “timed accessor information header” or not, the “flagVpccPartialAccess” flag is defined in MPEG\_accessor\_timed extension.

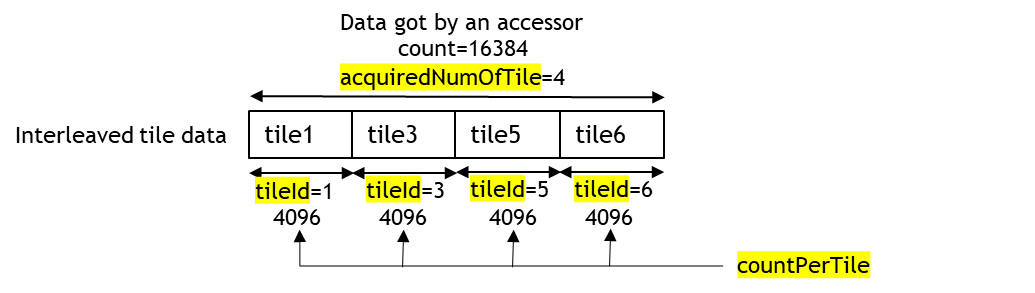


Fig. added fields to timed accessor header information

With the addition of these tile data boundary information, “timed accessor information header” will be variable length of the data. Thus, the length of bufferView which includes “timed accessor information header” shall be set to the value equal to or more than the length of timed accessor information header which includes tile data boundary information for all of tiles.

### Proposed text

#### Semantics

The "MPEG\_accessor\_timed " extension shall be defined on "accessors" structures. It may contain the following properties:

**Table 6 – Definition of MPEG\_accessor\_time extension**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| immutable | boolean | False | This flag indicates if the accessor information such as the componentType, bufferView, and type are allowed to change over time. Note that count, max, min, and byteOffset are expected to change and are always included as part of the timed accessor information header. |
| bufferView | integer | N/A | This provides the reference id of a bufferView that points to the timed accessor information header.  If the flagVpccPartialAccess is set to Ture, the byteLength of this bufferView shall be set to the value equal to or more than the length of timed accessor information header which includes tile data boundary information for all tiles. |
| updateRate | number | 25.0 | The updateRate provides the frequency at which the underlying buffer data is expected to change. The rate is provided in number of changes per second. |
| flagVpccPartialAccess | boolean | False | This flag indicates if the buffer referenced by this accessor includes partially acquired and decoded V-PCC component data. |

The timed accessor information header contains the dynamic metadata that is needed to access the timed data.

The following table describes the syntax and semantics of the timed accessor information header:

**Table 7 – Definition of timed accessor information header fields**

|  |  |  |  |
| --- | --- | --- | --- |
| **Syntax** | **Length (bits)** | **type** | **Semantics** |
| timestamp\_delta | 32 | float | Provides a delta in seconds that is added to the timestamp of the referenced buffer to determine the timestamp of the referenced timed media. |
| if (!immutable) {  componentType  bufferView  type  normalized  reserved  } | 32  32  8  1  7 | integer  integer  integer  boolean | These fields correspond to the accessor properties as defined in [glTF2.0]. The type differs from the definition in [glTF2.0] in that it provides a 0-based index of the allowed types as defined in [glTF2.0]. For example a type of 0 indicates that the data is a "SCALAR". |
| byteOffset  count  max  min | 32  32  32  32 | integer  integer  float  float | These fields correspond to the accessor properties as defined in [glTF2.0]. |
| bufferViewByteOffset  bufferViewByteLength  bufferViewByteStride | 32  32  32 | integer  integer  integer | These fields correspond to the bufferView fields byteOffset, byteLength, and byteStride respectively. |
| if(flagVpccPartialAccess){  acquiredNumOfTile  For(i=0; i<acquiredNumOfTile; i++){  tileId  countPerTile  }  } | 32  32  32 | integer  integer  integer | acquiredNumOfTile indicates the number of tiles included in the buffer referenced by the accessor. This number shall be same among the component data of one V-PCC file.  tileId indicates the tile ID of the tile. tileId is equal to value of afti\_tile\_id syntax element in atlas frame tile information, defined in ISO/IEC FDIS 23090-5. .  countPerTile provides count per tile, in which count has same semantics as the accessor properties with same name as defined in [glTF2.0]. |

Note that the timed accessor information header is provided as binary data as part of the buffer data and is accessible through the bufferView of the timed accessor extension.

#### JSON Syntax/Schema

|  |
| --- |
| {  "$schema" : "http://json-schema.org/draft-04/schema",  "title" : "MPEG\_accessor\_timed extension",  "type" : "object",  "description": "glTF extension to specify timed accessor format formats",  "allOf": [ { "$ref": "glTFChildOfRootProperty.schema.json" } ],  "properties" : {  "immutable": {  "type": "boolean",  "default": false,  "description": "This flag indicates if the accessor information such as the componentType, bufferView, and type are allowed to change over time. Note that count, max, min, and byteOffset are expected to change and are always included as part of the timed accessor information header. "  },  "bufferView": {  "allOf": [ { "$ref": "glTFid.schema.json" } ],  "description": "This provides the reference id of a bufferView that points to the timed accessor information header. "  },  "updateRate": {  "type": "number",  "default": 25.0,  "description": "The updateRate provides the frequency at which the underlying buffer data is expected to change. The rate is provided in number of changes per second."  },  "flagVpccPartialAccess": {  "type": "boolean",  "default": false,  "description": "This flag indicates if the buffer referenced by this accessor includes partially acquired and decoded V-PCC component data."  }  },  "required": [ "bufferView" ]  } |

## m57336 (42.1) [SD] Clarification of type of V-PCC track referenced from MPEG\_media

### 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.

### 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.

# Data formats

## m56102 (41.1) [SD] Support of glTF CBOR binary format

### 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](https://www.iana.org/assignments/media-types/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.

### 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.

### CBOR data size comparison example:

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

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

### Use Cases

#### 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](https://www.iana.org/assignments/media-types/application/cbor)”. When CBOR is used, binary data may be associated directly under the “url” parameter as follows:

|  |
| --- |
| {  “url”: “application/cbor:xxxxxxxx”  } |

#### 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.

#### 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.

### Potential Solutions

#### 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. xx – 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();  }; |

#### 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.

### Open Issue Discussion

#### CBOR IPR:

No IPR disclosures associated with IETF RFC 8949.

#### 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.

#### Implementation

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

#### 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.

# Viewpoint based Dynamic Bitrate Adaption

## m56094 [SD] On DASH Dynamic Bitrate Adaption with Viewpoint Update

### 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.

### Use Cases

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

|  |  |
| --- | --- |
| **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 |

#### 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 8**. 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 8 One DASH-based Timed Media Playback**

#### More than one timed media playback

When there is more than one timed media is played at the same time, as shown in **Figure 9**, network bandwidth usage is similar to the use case in 6.1.2.1. 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 9 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.

### Current Scene Description Support and Gasps

#### 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

#### Gaps Analysis

It is unclear how API and “viewinfo” data structure specified in 6.1.3.1 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.

# Dynamic Mesh Support

## m57410 [SD] Dynamic mesh support in scene description

### 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.

### 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.

### Assets and Implementation

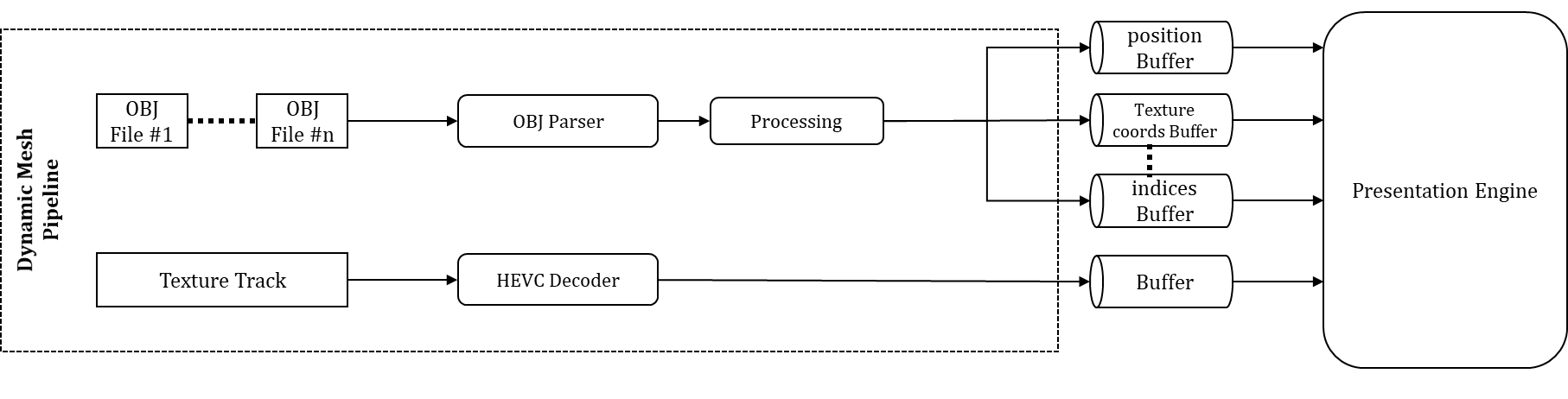
Adding support for timed meshes coincides with the start of the activity by the 3DG group on mesh coding. Similar to the point cloud support, the support for dynamic meshes can be done irrespective of whether the mesh is compressed or in raw format. Different pipeline variants maybe created to handle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

The following figure depicts the setup:



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

# Phase 2

# Interactivity (m57409r3)

The following basic interactivity features have been agreed:

* Camera restrictions: this feature allows the scene author to restrict the movement of the viewer in the scene. Currently there are two alternative proposals to do that. In the first one, camera paths are defined along which the camera can move. Each path segment has an associated bounding volume and may also allow for changes to the camera intrinsic parameters. This approach is defined in document m57409. In the second approach, a set of bounding volumes is defined and the camera is only allowed to be contained with that bounding volume set at any point in time. This approach is defined in clause F.2.1 of the MIV specification. <https://dms.mpeg.expert/doc_end_user/documents/133_OnLine/wg11/MDS20001_WG04_N00049.zip>
* Collision: this feature allows the scene author to define boundaries of rather simplified geometry for the objects in the scene. This information is used by the renderer (e.g. by a physics engine) to detect collision and emulate the impact of that collision. Document m57409 describes an extension to support this interactivity feature. A collision can be associated with an animation that is triggered when the collision is detected. The Boundaries of an object may be associated with physical properties that can be used to emulate a realistic reaction to the collision (e.g. bounciness and friction of the material, etc.)

In addition to these 2 basic features, interactivity through processing and handling of user actions, such as pushing a button on a controller, will be studied. Interactivity through haptics and haptics feedback are also relevant and will be studied as part of this EE. The MPEG-I haptics use case 2.4 is of particular interest: <https://dms.mpeg.expert/doc_end_user/documents/131_OnLine/wg11/w19513.zip> .

# Extensions

## MPEG\_scene\_anchor extensions (m56781, m57408)

### Introduction

The scene description scene node may contain a reference to an XR space, which indicates that the scene is anchored to that space. The anchor XR space may be a reference space, e.g. local, view, or stage. Alternatively, it may be an application-defined space that is setup at runtime based on application or user input. The scene description indicates the type of the anchor XR space. If the anchor space is an application-defined space, the scene description may provide some constraints on the XR space that the application should enforce when creating that XR Space.

The scene description may indicate a transformation of the scene coordinate system into the anchor XR space. For example, rotation, translation and scaling of the origin of the scene description coordinate space may be performed to align to the XR space. In the absence of any transform, the coordinate system of the scene description is aligned with that of the anchor space.

XR runtime systems such as OpenXR allow querying the bounding space for an XR space. The scene description may request that the presentation engine aligns the scene extents, i.e. the bounding box of the scene, to the bounding box of the anchor XR space. The scene description may indicate if this alignment involves scaling to match the two spaces or not. The scaling maybe in both dimensions, or it may conserve the spatial extent relationships of the scene. If no scaling is applied, the presentation engine aligns the long edge of the scene bounding box to that of the XR space and then centers the scene bounding box to be collocated with the center of the XR space bounding box.

This is depicted in the following figure:



In this example, the anchor XR space is of type stage, corresponding to the floor of the viewer’s living room.

The MPEG\_scene\_anchor is proposed as a glTF 2.0 extension to the scene node. An AR scene shall contain the MPEG\_scene\_anchor extension to describe the anchoring of the scene to real-world XR space. The extension must be declared in the extensionsUsed and should be included in the extensionsRequired of the glTF 2.0 scene.

In case the extension is not supported, the Presentation Engine falls back to rendering a complete virtual scene.

### Semantics

The “MPEG\_scene\_anchor” extension shall be defined on “scene” elements. It contains the following properties:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| xrReferenceSpaceType | enumeration | STAGE | the reference type may be one of VIEW=1, LOCAL=2, STAGE=3, or APPLICATION=4. |
| aligned | enumeration | NOT\_ALIGNED | the aligned flag may take one of the following values: NOT\_ALIGNED=0, ALIGNED\_NOTSCALED=1, ALIGNED\_SCALED=2.  If ALIGNED\_SCALED is set, the scene bounding box is aligned to the bounding box of the XR space. |
| if (aligned==NOT\_ALIGNED) { |  |  |  |
| transformation | matrix4x4 | N/A | a transformation of the scene space to anchor it to the XR space. |
| position | array(number) |  | position of the origin of the scene coordinate system in the XR space. |
| orientation | array(number) |  | Quaternion describing the rotation of the scene in the anchor space.  centerPosition and orientation are used as alternatives to transformation. |
| } |  |  |  |
| if (xrReferenceType==APPLICATION) { |  |  |  |
| xrActionSpace | string |  | An application defined Action Space, e.g. that corresponds to controller actions. The XRAction to be used as anchor is identified by the path, e.g.  “input/thumbstick/right” |
| xrObjectId | number |  | identifier shared between application and content server. The client may send a list of applications to the server or it may send a media streams to the server for object detection and then gets a list of identified objects and their locations. The shared objects may be used as anchor points. An XR Space may be created for anchor objects to track them in space. |
| } |  |  |  |
| poseFeedback | number |  | accessor that describes an uplink stream to which pose information is to be sent. |
| sceneUnderstandingStream | number |  | accessor that describes an uplink stream to which the local captured view is to be streamed. |

### Examples

|  |
| --- |
| {  "scene": 0,  "scenes": [  {  "extensions": {  "MPEG\_scene\_anchor": {  "xrReferenceSpaceType": 4, # Application  "aligned ": 0, # NOT\_ALIGNED  "xrActionSpace ": "/user/hand/right/input/grip",  "sceneUnderstandingStream": 15  }  },  "name": "Scene",  "nodes": [  759,  760,  761,  762,  763,  764  ]  }  ]  } |

### Processing model

A presentation engine that supports AR rendering, shall support the MPEG\_scene\_anchor extension. When AR rendering is enabled, the Presentation Engine shall setup the XR session and identify the reference XR space. Based on the aligned mode, the Presentation engine shall align the coordinate systems of the scene and the XRSpace for correct rendering.

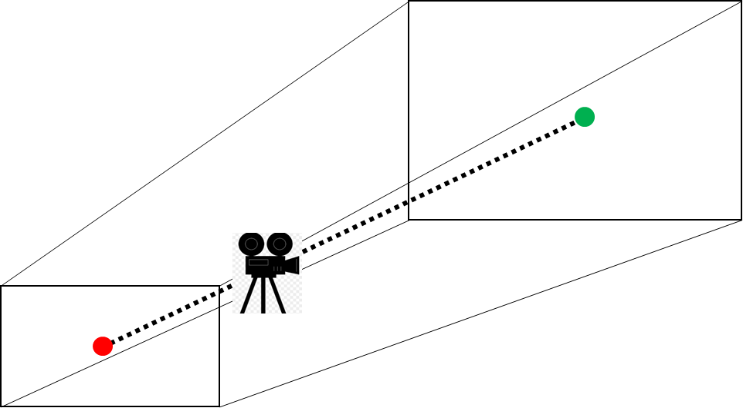
If a poseFeedback timed accessor is present, then the Presentation Engine shall send pose feedback to the location referenced in the MPEG\_media extension and according to the format described by the timed accessor and buffer view.

If sceneUnderstaningStream time access is present, then the Presentation Engine shall send a camera feed to the location referenced in the MPEG\_media extension and according to the format described by the timed accessor and buffer view.

## MPEG\_camera\_control extension (m56337, m57409)

### 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 following figure depicts this behavior.



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.

### Semantics

The “MPEG\_camera\_control” extension shall be defined on “camera” elements. It contains the following properties:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| anchors | number | N/A | Number of anchor points in the camera paths. |
| segments | number | N/A | Number of path segments in the camera paths. |
| boundingVolume | number | BV\_NONE | 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. |
| cameraIntrinsics | boolean | false | 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. |
| accessor | number | N/A | The index of the accessor or timed accessor that provides the camera control information. |

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.

### Processing Model

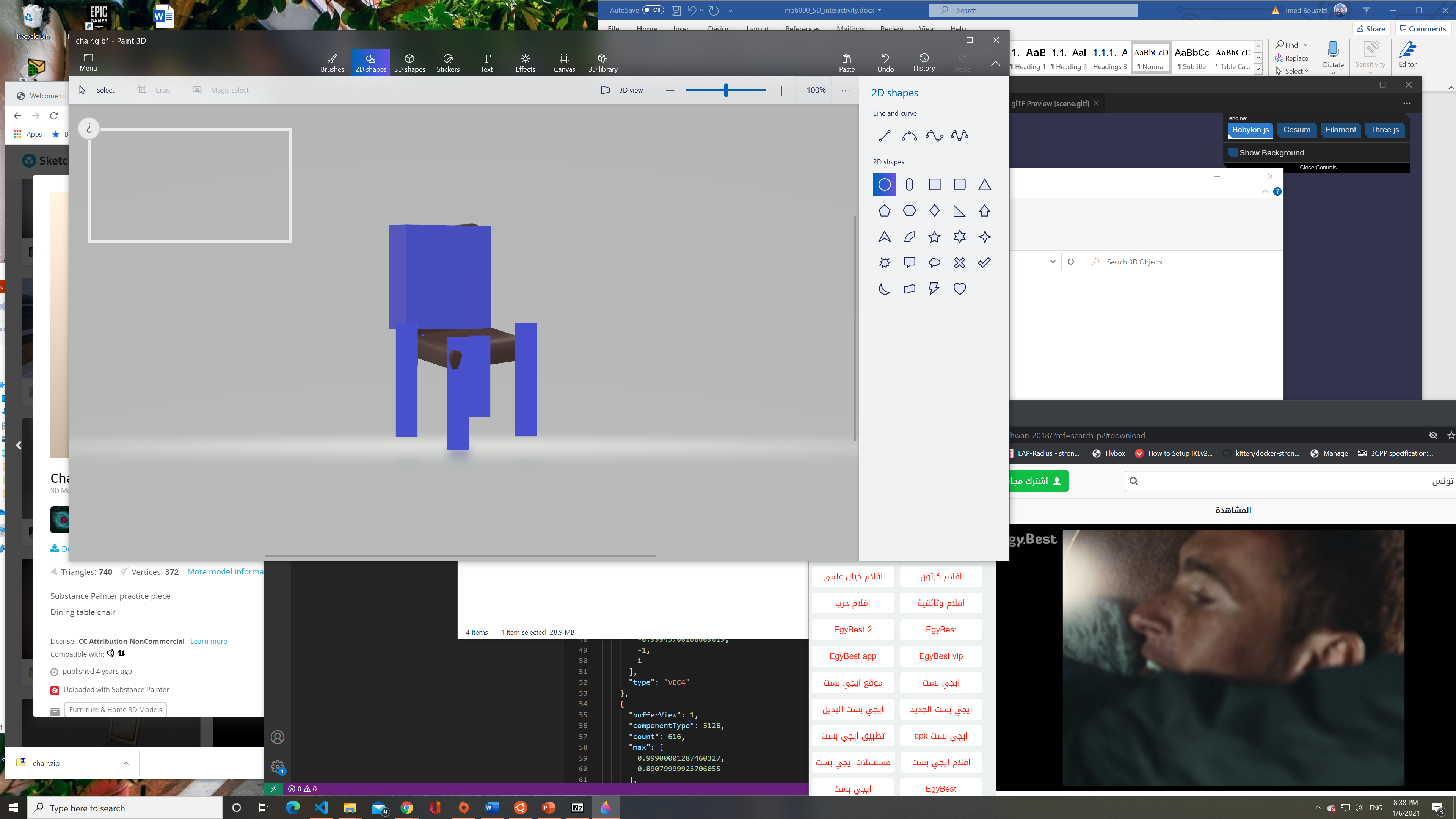
The Presentation Engine shall support the MPEG\_camera\_control extension. If the scene provides camera control information, the Presentation Engine shall limit the camera movement to the indicated paths, so that the (x,y,z) coordinates of the camera always lie on a path segment or within the bounding 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.

## MPEG\_mesh\_collision extension (m56337, m57409)

### General

In order to provide an immersive experience to the viewer, it is important that the viewer interacts properly with objects in the scene. The viewer should not be able to walk through solid objects in the scene, such as walls, chairs, and tables.

The following figure depicts a 3D mesh representation of a chair, together with its collision boundaries, defined as a set of cuboids.



### Semantics

The “MPEG\_mesh\_collision” extension is defined to provide a description of the collision boundaries of a mesh. The extension shall be defined on mesh objects as a set of cuboids around the mesh geometry.

It contains the following properties.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| boundaries | Array(object) | N/A | Array of boundary shapes that are used to define the collision boundaries of the mesh object. The boundaries may be spheroids or cuboids, as defined in the MPEG\_camera\_control extension. |
| static | boolean | True | Determines if the object is affected by collisions or not. An object that is static will not be affected by collisions, which means that when the viewer or another object collides with this object, its position will not be altered. |
| material | number | N/A | The index of a collision material that defines how colliding objects or viewers will interact with this object. This may include bounciness, friction, etc. |
| animations | Array(object) | N/A | Defines animations that are triggered by a collision or action on this object. The animation may be limited to a subset of other objects, e.g. only the viewer may trigger this animation. It also contains a pointer to the animation that is to be executed when triggered. |

The mesh collision information consists of the cuboid vertex coordinates (x,y,z) for cuboid boundaries or the sphere center and radius for spherical boundaries. The values are provided as float numbers.

### Processing Model

The Presentation Engine shall support the MPEG\_mesh\_collision extension. The camera position (x,y,z) shall not be contained within one of the defined mesh cuboids at any point of time. Collision may be signaled to the viewer through visual, acoustic, and/or haptic feedback.

This information on the boundaries for the nodes may be used to initialize and configure a 3D physics engine that will detect collisions.

## MPEG\_node\_transformation\_external extension (m56440)

### 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.

### Semantics

The semantics of the extension are defined in Table 8.

Table 8 – Semantics of MPEG node transformation external extension

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Default** | **Description** |
| matrix | string | N/A | The uri provides node's source of the transformation matrix |
| rotation | string | N/A | The uri provides node's source of the rotation |
| translation | string | N/A | The uri provides node's source of the translation |
| scale | string | N/A | The uri provides node's source of the scaling |

### JSON syntax/schema

|  |
| --- |
| {  "$schema" : "http://json-schema.org/draft-04/schema",  "title" : "MPEG\_node\_transformation\_external extension",  "type" : "object",  "description": "glTF extension to specify pose in scene is dependent on external information",  "allOf": [ { "$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."  }  }  }  } |

### Processing Model

The processing model could be the following.

1. Application parses scene description document, which contains MPEG node transformation external extension.
2. MPEG node transformation extension identifies that a node in glTF document may be linked with external information.
3. The application decides, which type of external information suits the handle defined in MEPG node transformation external extension.
4. The application applies transformation properties, as described by the extension, received from the external information source to the related node.
5. 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.

### 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 above 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’

# Interfaces

## CoAP API support in MAF (m56739)

### 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).

### MAF as CoAP Client

In this clause, the proposed MAF API in Table 1 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 1 – 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 |

### MAF as HTTP-CoAP Proxy

In this clause, the proposed MAF API in Table 2 applies to the case where the MAF acts as an HTTP-CoAP proxy.

**Table 2 – Description of HTTP-CoAP proxy API**

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

# References

[1] “WD on Support of MPEG Media in Scene Descriptions”, ISO/IEC JTC1/SC29/WG11 output document N19289, April 2020, Alpbach, AT.

[2] “Procedures for standard development and reference software of ISO/IEC 23090-14”, ISO/IEC JTC1/SC29/WG11 output document N19291, April 2020, Alpbach, AT.

[3] Khronos Group, The GL Transmission Format (glTF), version 2.0, https://github.com/KhronosGroup/glTF/tree/master/specification/2.0#specifying-extensions

[4] “Considerations for a WD on MPEG-I Scene Descriptions”, ISO/IEC JTC1/SC29/WG11 output document N18869, October 2019, Geneva, CH.

[5] “[SD] MPEG media extensions for glTF2”, ISO/IEC JTC1/SC29/WG11 input document m52179, January 2020, Brussels, BE.

[6] “Timed Video Textures”, ISO/IEC JTC1/SC29/WG11 input document m52462, January 2020, Brussels, BE.

[7] IETF RFC 6902, “JavaScript Object Notation (JSON) Patch”, April 2013

[8] ISO/IEC 23001-15, Carriage of Web Resources in ISOBMFF

[9] <https://github.com/KhronosGroup/glTF/blob/master/extensions/1.0/Vendor/CESIUM_RTC/README.md>

[10] <https://help.agi.com/AGIComponents/html/BlogPrecisionsPrecisions.htm>