ISO/IEC JTC 1/SC 29/WG 03 N0418

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

**Document type:** Output Document

**Title:** Technologies under Consideration for Immersive Media Metadata

**Status:** Approved

**Date of document:** 2021-10-15

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

**No. of pages:** 1 (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 N0418**

**October 2021, Virtual**

|  |  |
| --- | --- |
| **Title** | **Technologies under Consideration for Immersive Media Metadata** |
| **Source** | **WG 03, MPEG Systems** |
| **Status** | **Approved** |
| **Serial Number** | **20895** |

Contents

[**1** **Introduction** 2](#_Toc85211465)

[**2** **On harmonization of metadata for MPEG-I systems (m57815 [9])** 2](#_Toc85211466)

[**2.1** **Coordinate systems** 2](#_Toc85211467)

[**2.2** **Positions, offsets, dimensions, translation and scaling** 3](#_Toc85211468)

[**2.3** **Rotations and orientations** 4](#_Toc85211469)

[**2.4** **Bounding box** 4](#_Toc85211470)

[**2.5** **Viewport** 5](#_Toc85211471)

[**2.5.1** **Extrinsic camera parameters** 5](#_Toc85211472)

[**2.5.2** **Intrinsic camera parameters** 6](#_Toc85211473)

[**2.5.3** **Viewport** 8](#_Toc85211474)

[**2.6** **DASH Streaming** 8](#_Toc85211475)

[**3** **Defining Common Metadata for MPEG-I Immersive Media (m57821 [10])** 9](#_Toc85211476)

[**3.1** **“Viewing Space” in 23090-2 [2]** 9](#_Toc85211477)

[**3.2** **“Spatial region bounding box” and “Spatial Region” in 23090-10 [3]** 10](#_Toc85211478)

[**3.3** **“Bounding box” and “Spatial Region” in 23090-18 [4]** 11](#_Toc85211479)

[**3.4** **“2DRange” and “3DRange” in 23001-10 [5]** 11](#_Toc85211480)

[**4** **References** 12](#_Toc85211481)

1. **Introduction**

This document is a collection of technologies under consideration for MPEG-I Part 7 (ISO/IEC 23009-7), “MPEG-I Immersive Media Metadata”, currently for the AMD1, “Common Metadata for Immersive Media”.

1. **On harmonization of metadata for MPEG-I systems (m57815 [9])**
   1. **Coordinate systems**

Several documents define coordinate system with slightly different semantics. Considering that understanding the coordinate system for each specification is essential, it would be beneficial to align the terminology between the specifications. Especially the reference coordinate system in ISO/IEC DIS 23090-7 [6] seems particularly confusing as it is specific to a unit sphere, it would be preferrable to rename it as unit sphere coordinate system to avoid confusing it with other definitions.

Common reference coordinate system: 3D Cartesian coordinate system with the centre being (X, Y, Z) equal to (0, 0, 0), used as the reference coordinate system for all viewpoints within a viewpoint group – 23090-7

Cartesian co-ordinates: three scalars (x, y, z) with finite precision and dynamic range that indicate the location of a point relative to a fixed reference point – 23090-18

Cartesian coordinates: three scalars (x, y, z) with finite precision and dynamic range that indicate the location of a point relative to a fixed reference point (the origin) – 23090-5

Cartesian co-ordinates: three scalars (x, y, z) with finite precision and dynamic range that indicate the location of a point relative to a fixed reference point – 23090-9

Reference coordinate system: The coordinate system consists of a unit sphere and three coordinate axes, namely the X (back-to-front) axis, the Y (lateral, side-to-side) axis, and the Z (vertical, up) axis, where the three axes cross at the centre of the sphere. 23090-7

ISO/IEC DIS 23090-7 also defines terms for global coordinate axes and local coordinate axes. 3D graphics generally use terms object space and world space, and optionally local space. World space is the coordinate system for the entire scene, with its origin at the centre of the scene. Object space is the coordinate system from an object’s point of view. The origin of object space is at the object’s pivot or anchor point. World and object space axes may be oriented differently.

Global coordinate axes: coordinate axes that are associated with audio, video, and images representing the same acquisition position and intended to be rendered together – 23090-7

Local coordinate axes: the coordinate axes obtained after applying rotation to the global coordinate axes – 23090-7

It might be beneficial to introduce concepts of object space and world space and describe the related transformations moving from one space to the other to improve the current definitions in ISO/IEC DIS 23090-7 [6].

* 1. **Positions, offsets, dimensions, translation and scaling**

All the surveyed specifications contain triplets of values describing positions, offsets, dimensions, scaling or translation. These commonly consist of x-, y- and z-components that are applied to the corresponding world or object space axes. E.g. as illustrated by GPCCSpatialRegionStruct in ISO/IEC 23090-18 [4] below.

aligned(8) class GPCCSpatialRegionStruct(dimension\_included) {

unsigned int(16) 3d\_region\_id;

unsinged int(16) anchor\_x;

unsinged int(16) anchor\_y;

unsinged int(16) anchor\_z;

if (dimension\_included)

{

unsinged int(16) region\_dx;

unsinged int(16) region\_dy;

unsinged int(16) region\_dz;

}

}

ISO/IEC FDIS 23090-10 [3] has taken a different approach and defined a common syntax structure that can be used to provide these triplets of data as illustrated below. The syntax structure allows to define precision for the x-, y- and z-components of the structure. Vector3 may then be used in every occurrence where these triplets are used to describe positions, offsets, dimensions translation or scaling.

aligned(8) class Vector3(int precision = 32) {

int reserved\_bits = 8 - (precision\*3) % 8;

if (reserved\_bits != 8) {

bit(reserved\_bits) reserved = 0;

}

unsigned int(precision) x;

unsigned int(precision) y;

unsigned int(precision) z;

}

Adoption of Vector3 would allow to rewrite the GPCCSpatialRegionStruct as follows:

aligned(8) class GPCCSpatialRegionStruct(dimension\_included) {

unsigned int(16) 3d\_region\_id;

Vector3 anchor(16);

if (dimension\_included)

{

Vector3 dimensions(16);

}

}

This would simplify the syntax structures and result in less specification text. Individual components of the vector may be addressed in semantics by referring to anchor.x or dimensions.z.

Specifications seem to use different type of values of describing position, some use unsigned integers, some use signed integers and, in some cases, even floating points are used. This could mean that at least three types of Vector3 need to be created, i.e. Vector3Float, Vector3UInt, Vector3Int. The decision of which structure to use should not be made arbitrarily. E.g. for normalized positions as described in ISO/IEC DIS 23090-9 Vector3UInt should be used. Whereas viewport related signaling may require use of Vector3Float.

Geometry coding related position: (x, y, z) co-ordinates of a point, where the values are normalized by the bounding box so that the values of the positions shall be equal to or greater than 0. – 23090-9

* 1. **Rotations and orientations**

Specifications should clarify the difference between orientation and rotation. Orientation is the result after applying a rotation. These should not be used interchangeably. Different terminology for rotations and orientations exist in specifications. Some use quaternions (x,y,z,w), some use rotations with azimuth, elevation and tilt or yaw, pitch and roll.

23090-7 Clause 5.3.1.2 [6] contains the following

aligned(8) class RotationStruct() {

signed int(32) rotation\_yaw;

signed int(32) rotation\_pitch;

signed int(32) rotation\_roll;

}

Conscious decision should be made between usage of rotation and orientation. E.g. orientation is idea for describing how an object should be oriented at given point in time, whereas rotation is ideal for describing how the object should rotate to reach said orientation. Furthermore, when a common syntax for rotation and orientation is specified, it should be used consistently between specifications.

Rotation is always dependent on the order of applying individual rotations to an object. E.g. applying yaw, pitch and roll may lead in different orientation than applying pitch, roll and yaw. It would be therefore important to also specify the order of rotations in a shared document.

* 1. **Bounding box**

Several documents describe bounding box, which seems to mostly mean the same thing.

3D bounding box: volume defined as a cuboid solid having six rectangular faces placed at right angles. 23090-5

Bounding box: rectangular cuboid in which the source point cloud frame is included – 230990-18

bounding box: rectangular cuboid in which the source point cloud frame is included. – 230990-9

The syntax for bounding box however seems to be somewhat different, but consist of the same components. V3CBoundingBox is defined in ISO/IEC FDIS 23090-10 [3], GPCCSpatialRegionStruct in ISO/IEC FDIS 23090-18 [4] contains essentially a bounding box in it and vui\_parameters() in ISO/IEC FDIS 23090-5 [7] also define a bounding box.

aligned(8) class V3CBoundingBox (anchor\_included, scale\_included)

{

if (anchor\_included) { // anchor is not 0,0,0

unsigned int(8) bb\_pos\_precision;

Vector3 bb\_position(bb\_pos\_precision);

}

if (scale\_included) {

unsigned int(8) bb\_scale\_precision;

Vector3 bb\_scale(bb\_scale\_precision);

}

}

aligned(8) class GPCCSpatialRegionStruct(dimension\_included) {

unsigned int(16) 3d\_region\_id;

unsinged int(16) anchor\_x;

unsinged int(16) anchor\_y;

unsinged int(16) anchor\_z;

if (dimension\_included)

{

unsinged int(16) region\_dx;

unsinged int(16) region\_dy;

unsinged int(16) region\_dz;

}

|  |  |
| --- | --- |
| if( **vui\_display\_box\_info\_present\_flag**) { |  |
| for( d = 0; d < 3; d++ ) { |  |
| **vui\_display\_box\_origin**[ d ] | u(v) |
| **vui\_display\_box\_size**[ d ] | u(v) |
| } |  |
| } |  |

Common definition for a bounding box and consistent usage would help consolidating specifications and results in less specification text overall.

* 1. **Viewport**

Many specifications seem to utilize viewport signalling for different purpose. The viewport is typically signalled using extrinsic and intrinsic camera parameters. Viewport related terminology exists in 23090-10 clause 10 [3], 23090-5 Clause F.2.15 [7], and 23090-12 Clause 8.2.1.6.2 [8]. There are also ongoing discussions in 23090-18 [4]. In addition 23090-7 [6] contains a definition for viewport which seems quite specific to their use case.

viewport: region of omnidirectional image or video suitable for display and viewing by the user – 23090-7 [6]

* + 1. **Extrinsic camera parameters**

Extrinsic camera parameters are commonly used to place a virtual camera in a scene. It describes the position and orientation of the camera in world space. E.g. 23090-10 [3] describes it as follows:

Aligned(8) class ExtCameraInfo () {

unsigned int(8)[4] cam\_pos\_x;

unsigned int(8)[4] cam\_pos\_y;

unsigned int(8)[4] cam\_pos\_z;

signed int(32) cam\_quat\_x;

signed int(32) cam\_quat\_y;

signed int(32) cam\_quat\_z;

}

23090-12 [8] has almost exactly same definition but different precision for the orientation.

|  |  |
| --- | --- |
| camera\_extrinsics( viewID ) { | **Descriptor** |
| **ce\_view\_pos\_x**[ viewID ] | fl(32) |
| **ce\_view\_pos\_y**[ viewID ] | fl(32) |
| **ce\_view\_pos\_z**[ viewID ] | fl(32) |
| **ce\_view\_quat\_x**[ viewID ] | i(16) |
| **ce\_view\_quat\_y**[ viewID ] | i(16) |
| **ce\_view\_quat\_z**[ viewID ] | i(16) |
| } |  |

23090-5 [7] contains slightly more information as it is intended for dynamic signalling of the extrinsic information.

|  |  |
| --- | --- |
| viewport\_position( payloadSize ) { | **Descriptor** |
| **vp\_viewport\_id** | ue(v) |
| **vp\_camera\_parameters\_present\_flag** | u(1) |
| if( vp\_camera\_parameters\_present\_flag ) |  |
| **vp\_vcp\_camera\_id** | u(10) |
| **vp\_cancel\_flag** | u(1) |
| if( !vp\_cancel\_flag ) { |  |
| **vp\_persistence\_flag** | u(1) |
| for( d = 0 ; d < 3; d++) |  |
| **vp\_position**[d] | fl(32) |
| **vp\_rotation\_qx** | i(16) |
| **vp\_rotation\_qy** | i(16) |
| **vp\_rotation\_qz** | i(16) |
| **vp\_center\_view\_flag** | u(1) |
| if( !vp\_center\_view\_flag ) |  |
| **vp\_left\_view\_flag** | u(1) |
| } |  |
| } |  |

Defining the position of extrinsic parameters should be done in a way that allows the viewport to move in the world space, thus several specifications have opted for using floating points to define such data.

The rotation component requires slightly more thought. It seems that quaternions are preferred for this purpose, which makes sense as it describes the orientation of the virtual camera. However, if the intention is to describe how a camera rotates to the target orientation, a rotation could be considered equally as well. This could be useful for more compact description of camera rotation, e.g. consider what is required to describe 360° degree rotation around z-axis with quaternion or rotation.

* + 1. **Intrinsic camera parameters**

Intrinsic camera parameters are combined with extrinsic parameters to create a viewport into the scene. Whereas extrinsic parameters are used to describe how camera is positioned and oriented in space. The intrinsic parameters defined how the 3D scene is projected into 2D plane, also called as viewport. There are a couple of different projection formats which are also reflected in the intrinsic parameter syntax structure.

Defined in 23090-10 [3]

aligned(8) class IntCameraInfo () {

unsigned int(10) camera\_id;

bit(3) reserved = 0;

unsigned int(3) camera\_type;

if (camera\_type == 0) {

signed int(32) erp\_horizontal\_fov;

signed int(32) erp\_vertical\_fov;

}

if (camera\_type == 1) {

signed int(32) perspective\_horizontal\_fov;

unsigned int(8)[4] perspective\_aspect\_ratio;

}

if (camera\_type == 2) {

unsigned int(8)[4] ortho\_aspect\_ratio;

unsigned int(8)[4] ortho\_horizontal\_size;

}

unsigned int(8)[4] clipping\_near\_plane;

unsigned int(8)[4] clipping\_far\_plane;

}

And In 23090-12 [8]

|  |  |
| --- | --- |
| camera\_intrinsics( viewID, mode ) { | **Descriptor** |
| **ci\_cam\_type**[ viewID ] | u(8) |
| **ci\_projection\_plane\_width\_minus1**[ viewID ] | u(16) |
| **ci\_projection\_plane\_height\_minus1**[ viewID ] | u(16) |
| if( ci\_cam\_type[ viewID ] == 0 ) { /\* equirectangular \*/ |  |
| **ci\_erp\_phi\_min**[ viewID ] | fl(32) |
| **ci\_erp\_phi\_max**[ viewID ] | fl(32) |
| **ci\_erp\_theta\_min**[ viewID ] | fl(32) |
| **ci\_erp\_theta\_max**[ viewID ] | fl(32) |
| } else if( ci\_cam\_type[ viewID ] == 1 ) { /\* perspective \*/ |  |
| **ci\_perspective\_focal\_hor**[ viewID ] | fl(32) |
| **ci\_perspective\_focal\_ver**[ viewID ] | fl(32) |
| **ci\_perspective\_principal\_point\_hor**[ viewID ] | fl(32) |
| **ci\_perspective\_principal\_point\_ver**[ viewID ] | fl(32) |
| } else if( ci\_cam\_type[viewID] == 2 ) { /\* orthographic \*/ |  |
| **ci\_ortho\_width**[ viewID ] | fl(32) |
| **ci\_ortho\_height**[ viewID ] | fl(32) |
| } |  |
| } |  |

And in 23090-5 [7]

|  |  |
| --- | --- |
| viewport\_camera\_parameters( payloadSize ) { | **Descriptor** |
| **vcp\_camera\_id** | u(10) |
| **vcp\_cancel\_flag** | u(1) |
| if( vcp\_camera\_id > 0 && !vcp\_cancel\_flag ) { |  |
| **vcp\_persistence\_flag** | u(1) |
| **vcp\_camera\_type** | u(3) |
| if( vcp\_camera\_type == 0 ) { /\* equirectangular \*/ |  |
| **vcp\_erp\_horizontal\_fov** | u(32) |
| **vcp\_erp\_vertical\_fov** | u(32) |
| } else if( vcp\_camera\_type == 1 ) { /\* perspective \*/ |  |
| **vcp\_perspective\_aspect\_ratio** | fl(32) |
| **vcp\_perspective\_horizontal\_fov** | u(32) |
| } else if( vcp\_camera\_type == 2 ) { /\* orthographic \*/ |  |
| **vcp\_ortho\_aspect\_ratio** | fl(32) |
| **vcp\_ortho\_horizontal\_size** | fl(32) |
| } |  |
| **vcp\_clipping\_near\_plane** | fl(32) |
| **vcp\_clipping\_far\_plane** | fl(32) |
| } |  |
| } |  |

* + 1. **Viewport**

Combining the extrinsic and intrinsic parameters defines the viewport to the scene. Different specifications combine them differently. 23090-5 [7] and 23090-12 [8] use camera or view identifiers to combine them in different structures, and 23090-10 [3] explicitly wraps them in the same structure as follows:

aligned(8) class ViewportInfo (ext\_camera\_flag, int\_camera\_flag) {

if (ext\_camera\_flag == 1) {

unsigned int(1) center\_view\_flag;

bit(6) reserved = 0;

if (center\_view\_flag == 0) {

unsigned int(1) left\_view\_flag;

} else {

bit(1) reserved = 0;

}

ExtCameraInfo extCamInfo();

}

if (int\_camera\_flag == 1) {

IntCameraInfo intCamInfo();

}

}

For defining recommended viewport it should be considered if syntax for signalling center-view or left-view is actually required. Viewport describes how the scene is projected into 2d space and does not depend on which eye the result is displayed. Applications that intend to render 3D content as stereo typically depend on the viewing device, which provide information about the viewer such as interpupillary distance or gaze orientation data. It is unclear how center-view and left-view would be used by an application.

* 1. **DASH Streaming**

It may make sense to add DASH streaming related syntax structures in a common specification. E.g. 23090-10 [3] and 23090-18 [4] define identical v3sr.spatialRegion and gpsr.spatialRegion descriptors.

1. **Defining Common Metadata for MPEG-I Immersive Media (m57821 [10])**
   1. **“Viewing Space” in 23090-2 [2]**

aligned(8) class ViewingSpaceStruct() {  
 unsigned int(8) viewing\_space\_shape\_type;  
 unsigned int(16) distance\_scale;  
 bit(1) guard\_range\_flag;  
 bit(7) reserved;  
 if(viewing\_space\_shape\_type==0)  
 CuboidStruct(guard\_range\_flag);  
 else if(viewing\_space\_shape\_type==1)  
 SphereStruct(guard\_range\_flag);  
 else if(viewing\_space\_shape\_type==2)  
 CylinderStruct(guard\_range\_flag);   
 else if(viewing\_space\_shape\_type==3)  
 EllipsoidStruct(guard\_range\_flag);  
}

aligned(8) class CuboidStruct(guard\_range\_flag) {  
 signed int(32) x\_Min;  
 signed int(32) x\_Max;  
 signed int(32) y\_Min;  
 signed int(32) y\_Max;  
 signed int(32) z\_Min;  
 signed int(32) z\_Max;  
 if (guard\_range\_flag) {  
 unsigned int(8) guard\_range\_X;  
 unsigned int(8) guard\_range\_Y;  
 unsigned int(8) guard\_range\_Z;  
 }  
}

aligned(8) class SphereStruct(guard\_range\_flag) {  
 unsigned int (32) sphere\_radius;   
 if (guard\_range\_flag){   
 bit(1) reserved;  
 unsigned int(7) guard\_radius\_diff;  
 }  
}

aligned(8) class CylinderStruct(guard\_range\_flag) {  
 unsigned int (32) cylinder\_radius;  
 Point(0);  
 Point(1);  
 if (guard\_range\_flag)  
 unsigned int(8) cylinder\_guard\_radius\_diff;  
}

aligned(8) class Point(i) {  
 signed int(32) x\_pt[i];  
 signed int(32) y\_pt[i];  
 signed int(32) z\_pt[i];  
}

aligned(8) class EllipsoidStruct(guard\_range\_flag) {  
 unsigned int (32) length\_X;  
 unsigned int (32) length\_Y;  
 unsigned int (32) length\_Z;  
 if (guard\_range\_flag) {  
 unsigned int(8) guard\_lenghthX\_diff;  
 unsigned int(8) guard\_lenghthY\_diff;  
 unsigned int(8) guard\_lenghthZ\_diff;  
 }  
}

aligned(8) SphereRegionStruct(range\_included\_flag, interpolate\_included\_flag) {  
 signed int(32) centre\_azimuth;  
 signed int(32) centre\_elevation;  
 signed int(32) centre\_tilt;  
 if (range\_included\_flag) {  
 unsigned int(32) azimuth\_range;  
 unsigned int(32) elevation\_range;  
 }  
 if (interpolate\_included\_flag) {  
 unsigned int(1) interpolate;  
 bit(7) reserved = 0;  
 }  
}

* 1. **“Spatial region bounding box” and “Spatial Region” in 23090-10 [3]**

aligned(8) class V3CBoundingBox (anchor\_included, scale\_included) {

if (anchor\_included) { // anchor is not 0,0,0

unsigned int(8) bb\_pos\_precision;

Vector3 bb\_position(bb\_pos\_precision);

}

if (scale\_included) {

unsigned int(8) bb\_scale\_precision;

Vector3 bb\_scale(bb\_scale\_precision);

}

}

aligned(8) class Vector3(int precision = 32) {

int reserved\_bits = 8 - (precision\*3) % 8;

if (reserved\_bits != 8) {

bit(reserved\_bits) reserved = 0;

}

unsigned int(precision) x;

unsigned int(precision) y;

unsigned int(precision) z;

}

aligned(8) class V3CSpatialRegion {

unsigned int(32) size;

unsigned int(16) region\_id;

unsigned int(1) bb\_anchor\_present\_flag;

unsigned int(1) bb\_scale\_present\_flag;

unsigned int(1) tile\_mapping\_present\_flag;

unsigned\_int(1) tm\_spatial\_scalability\_flag;

unsigned\_int(1) object\_collection\_present\_flag;

bit(3) reserved = 0;

if (bb\_anchor\_present\_flag || bb\_scale\_present\_flag) {

V3CBoundingBox bounding\_box(bb\_anchor\_present\_flag, bb\_scale\_present\_flag);

}

if (tile\_mapping\_present\_flag) {

TileMapping tile\_map(tm\_spatial\_scalability\_flag);

}

if (object\_collection\_present\_flag) {

ObjectCollection object\_collection;

}

}

* 1. **“Bounding box” and “Spatial Region” in 23090-18 [4]**

**Bounding box** [note: no metadata structure/box is given]

rectangular cuboid in which the source point cloud frame is included.

aligned(8) class GPCCSpatialRegionStruct(dimension\_included) {  
 unsigned int(16) 3d\_region\_id;

unsinged int(16) anchor\_x;

unsinged int(16) anchor\_y;

unsinged int(16) anchor\_z;

if (dimension\_included)

{  
 unsinged int(16) region\_dx;

unsinged int(16) region\_dy;

unsinged int(16) region\_dz;  
}

}

|  |  |  |  |
| --- | --- | --- | --- |
| **gpsr.spatialRegion**@x | OD | xs:int | The x-coordinate of the reference point for the bounding box defining the spatial region.  If not present, the default value is 0. |
| **gpsr.spatialRegion**@y | OD | xs:int | The y-coordinate of the reference point for the bounding box defining the spatial region.  If not present, the default value is 0. |
| **gpsr.spatialRegion**@z | OD | xs:int | The z-coordinate of the reference point for the bounding box defining the spatial region.  If not present, the default value is 0. |
| **gpsr.spatialRegion**@dx | M | xs:int | The length of the bounding box along the x-axis (i.e., width). Negative values indicate a length that extends in the negative direction of the axis. |
| **gpsr.spatialRegion**@dy | M | xs:int | The length of the bounding box along the y-axis (i.e., height). Negative values indicate a length that extends in the negative direction of the axis. |
| **gpsr.spatialRegion**@dz | M | xs:int | The length of the bounding box along the z-axis (i.e., depth). Negative values indicate a length that extends in the negative direction of the axis. |

* 1. **“2DRange” and “3DRange” in 23001-10 [5]**

aligned(8) class 2DRangeStruct(shape\_type) {  
 if (shape\_type == 0) { // 2D rectangle   
 unsigned int(32) range\_width;  
 unsigned int(32) range\_height;  
 }  
 if (shape\_type == 1) { // 2D circle  
 unsigned int(32) range\_radius;  
 }  
 // other values of shape\_type are reserved  
}

aligned(8) class 3DRangeStruct(shape\_type) {  
 2DRangeStruct(shape\_type); // including 2D shape types  
 if (shape\_type == 2) { // 3D tile  
 unsigned int(32) range\_width;  
 unsigned int(32) range\_height;  
 unsinged int(32) range\_depth;  
 }  
 if (shape\_type == 3) { // 3D spherical region  
 unsigned int(32) range\_width;  
 unsigned int(32) range\_height;  
 unsinged int(32) range\_depth;  
 }  
 if (shape\_type == 4) { // 3D sphere  
 unsigned int(32) range\_radius;  
 }  
 // other values of shape\_type are reserved  
}

1. **References**
2. WG03N0286\_20540. “Technologies under Consideration for ISO/IEC 23001-10”. The 135th MPEG meeting. July 2021.
3. WG3N0072. “Text of ISO/IEC FDIS 23090-2 2nd edition OMAF”. The 132nd MPEG meeting. October 2020.
4. WG03N00241. “Text of ISO/IEC FDIS 23090-10 Carriage of Visual Volumetric Video-based Coding Data”. The 135th MPEG meeting. July 2021.
5. WG03N0242\_20304. “Draft text of ISO/IEC FDIS 23090-18 Carriage of Geometry-based Point Cloud Compression Data”. The 134th MPEG meeting. April 2021.
6. SC29WG11\_N18637. “Technologies under Consideration for ISO/IEC 23001-10”. The 127th MPEG meeting. July 2019.
7. WG03N00193\_20255. “Text of ISO/IEC DIS 23090-7 Metadata for Immersive Media”. The 134th MPEG meeting. April 2021.
8. ISO/IEC 23090-5:2021. “Information technology — Coded representation of immersive media — Part 5: Visual volumetric video-based coding (V3C) and video-based point cloud compression (V-PCC)”, June 2021.
9. MDS20001. “Text of ISO/IEC DIS 23090-12 MPEG Immersive Video”, January 2021.
10. m57815. “On harmonization of metadata for MPEG-I systems”. The 136th MPEG meeting. October
11. m57821. “On Defining Common Metadata for MPEG-I Immersive Media”. The 136th MPEG meeting. October 2021.