INTERNATIONAL STANDARD

ISO/IEC 23090-7

First edition
2022-11

Information technology — Coded representation of immersive media —

Part 7: Immersive media metadata

Technologies de l'information — Représentation codée de média immersifs —

Partie 7: Métadonnées de media immersifs

Reference number
ISO/IEC 23090-7:2022(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see https://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO/IEC 23090 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

This document is organized as follows.

— Clauses 5 describes common metadata applicable to immersive media. This includes reference co-ordinate system related metadata and other common metadata syntax and semantics.

— Clauses 6 describes metadata that applies to video and images. This includes projection formats and packing region-wise formats metadata which applies to video and images.

The goal of this document is to allow reuse of the commonly defined metadata to be referenced by other standards.

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO and IEC that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and IEC. Information may be obtained from the patent database available at www.iso.org/patents or https://patents.iec.ch.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those in the patent database. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — Coded representation of immersive media —

Part 7: Immersive media metadata

1 Scope

This document specifies common immersive media metadata focusing on immersive videos (including 360° videos) and images.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14496-12, *Information technology — Coding of audio-visual objects — Part 12: ISO base media file format*

ISO/IEC 23008-12, *Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 12: Image file format*

3 Terms, definitions and symbols

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 14496-12 and ISO/IEC 23008-12 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1.1
azimuth
first of the two *sphere coordinates* (3.1.22) describing the location of a point on the sphere

3.1.2
azimuth circle
circle on the sphere connecting all points with the same *azimuth* (3.1.1) value

Note 1 to entry: An azimuth circle is always a *great circle* (3.1.12).

3.1.3
circular image
image captured with a *fisheye lens* (3.1.9)

3.1.4
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* (3.1.27)

3.1.5
content coverage
one or more *sphere regions* (3.1.23) that are covered by the content represented by the track or by an image item

3.1.6
elevation
second of the two *sphere coordinates* (3.1.22) describing the location of a point on the sphere

3.1.7
elevation circle
circle on the sphere connecting all points with the same *elevation* (3.1.6) value

Note 1 to entry: When the elevation is zero, an elevation circle is also a *great circle* (3.1.12). This coincides with the equator on Earth.

3.1.8
field of view
extent of the observable world in captured/recorded content or in a physical display device

3.1.9
fisheye lens
wide-angle camera lens that usually captures an approximately hemispherical *field of view* (3.1.8) and projects it as a *circular image* (3.1.3)

3.1.10
fisheye video
video captured by *fisheye lenses* (3.1.9)

3.1.11
global coordinate axes
coordinate axes that are associated with audio, video, and images representing the same acquisition position and intended to be rendered together

3.1.12
great circle
intersection of the sphere and a plane that passes through the centre point of the sphere

Note 1 to entry: A great circle is also known as an orthodrome or Riemannian circle.

Note 2 to entry: The centre of the sphere and the centre of a great circle are co-located.

3.1.13
guard band
area in a *packed picture* (3.1.16) that is not rendered but may be used to improve the rendered part of the packed picture to avoid or mitigate visual artifacts such as seams

Note 1 to entry: Guard bands are associated with *packed regions* (3.1.17) as described in 6.5.2.

3.1.14
local coordinate axes
coordinate axes obtained after applying rotation to the *global coordinate axes* (3.1.11)

3.1.15
omnidirectional video
video and its associated audio that enable rendering according to the user's *viewing orientation* (3.1.26), if consumed with a head-mounted device, or according to user's desired *viewport* (3.1.28), otherwise, as if the user was in the spot where and when the media was captured

3.1.16
packed picture
picture that is represented as a coded picture in the coded video bitstream

3.1.17
packed region
region in a *packed picture* (3.1.16) that is mapped to a *projected region* (3.1.19) as specified by the *region-wise packing* (3.1.21) signalling

3.1.18
projected picture
picture that has a representation format specified by an *omnidirectional video* (3.1.15) *projection* (3.1.20) format

3.1.19
projected region
region in a *projected picture* (3.1.18) that is mapped to a *packed region* (3.1.17) as specified by the *region-wise packing* (3.1.21) signalling

3.1.20
projection
inverse of the process by which the samples of a *projected picture* (3.1.18) are mapped to a set of positions identified by a set of *azimuth* (3.1.1) and *elevation* (3.1.6) coordinates on a unit sphere

3.1.21
region-wise packing
inverse of the process of transformation, resizing, and relocating of *packed regions* (3.1.17) of a *packed picture* (3.1.16) to remap to *projected regions* (3.1.19) of a *projected picture* (3.1.18)

3.1.22
sphere coordinates
azimuth (ϕ) (3.1.1) and *elevation* (θ) (3.1.6) that identify a location of a point on the unit sphere

3.1.23
sphere region
region on a sphere, specified either by four *great circles* (3.1.12) or by two *azimuth circles* (3.1.2) and two *elevation circles* (3.1.7), or such a region on the rotated sphere after applying certain amount of yaw, pitch, and roll rotations

3.1.24
SDL
syntactic description language
language that allows the description of a bitstream's syntax

Note 1 to entry: Syntactic description language is defined in ISO/IEC 14496-1:2010, Clause 8.

3.1.25
tilt angle
angle indicating the amount of tilt of a *sphere region* (3.1.23), measured as the amount of rotation of the sphere region along the axis originating from the sphere origin passing through the centre point of the sphere region, where the angle value increases clockwise when looking from the origin towards the positive end of the axis

3.1.26
viewing orientation
triple of *azimuth* (3.1.1), *elevation* (3.1.6), and *tilt angle* (3.1.25) characterizing the orientation that a user is consuming the audio-visual content

Note 1 to entry: In case of image or video, viewing orientation characterizes the orientation of the *viewport* (3.1.28).

3.1.27
viewpoint group
group of viewpoints that share the same *common reference coordinate system* (3.1.4)

3.1.28
viewport
region of omnidirectional image or video suitable for display and viewing by the user

3.2 Symbols

+	Addition.
−	Subtraction (as a two-argument operator) or negation (as a unary prefix operator).
*	Multiplication, including matrix multiplication.
x^y	Exponentiation. Specifies x to the power of y. In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation.
/	Integer division with truncation of the result toward zero. For example, 7 / 4 and −7 / −4 are truncated to 1 and −7 / 4 and 7 / −4 are truncated to −1.
÷	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\frac{x}{y}$	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\sum_{i=x}^{y} f(i)$	The summation of f(i) with i taking all integer values from x up to and including y.
x % y	Modulus. Remainder of x divided by y, defined only for integers x and y with x >= 0 and y > 0.
Asin(x)	The trigonometric inverse sine function, operating on an argument x that is in the range of −1.0 to 1.0, inclusive, with an output value in the range of −π÷2 to π÷2, inclusive, in units of radians.
Atan(x)	The trigonometric invers tangent function, operating on an argument x that is any real number, with an output value in the range of −π÷2 to π÷2, inclusive, in units of radians.

$$\mathrm{Atan2}(y,x)=\begin{cases}\mathrm{Atan}\left(\frac{y}{x}\right) & ; \text{ if } x>0\\ \mathrm{Atan}\left(\frac{y}{x}\right)+\pi & ; \text{ if } x<0 \;\&\&\; y>=0\\ \mathrm{Atan}\left(\frac{y}{x}\right)-\pi & ; \text{ if } x<0 \;\&\&\; y<0\\ +\frac{\pi}{2} & ; \text{ if } x==0 \;\&\&\; y>=0\\ -\frac{\pi}{2} & ; \text{ otherwise}\end{cases} \tag{3-1}$$

Cos(*x*) The trigonometric cosine function operating on an argument *x* in units of radians.

Floor(*x*) The the largest integer less than or equal to *x*.

Sin(*x*) The trigonometric sine function operating on an argument *x* in units of radians.

Tan(*x*) The trigonometric tangent function operating on an argument *x* in units of radians.

4 Overview

4.1 General

This document specifies common immersive media metadata focusing on immersive videos (including 360° videos) and images. The metadata includes co-ordinate system, projection format, and packing region-wise formats metadata.

4.2 Variables

This document derives variables that are named by a mixture of lower case and upper case letter and without any underscore characters.

4.3 Processes

Processes are used to describe the various operations. A process has a set of one or more inputs, a set of one or more outputs and a sequence of operation steps.

4.4 Syntax structures

Syntax structures in this document are specified with the syntactic description language (SDL) specified in ISO/IEC 14496-1:2010, Clause 8, with the following change: Unlike specified in ISO/IEC 14496-1:2010, Clause 8, this document allows a variable declaration in `expression1` of a for loop `for(expression1; expression2; expression3)`. Such a variable declaration may be used for a loop index variable with a data type.

NOTE As specified in ISO/IEC 14496-1:2010, 8.3.6, this document allows declaring a syntax element that is an individual element in an array. Such a declaration follows ISO/IEC 14496-1:2010, Rule A.2: `typespec name[[index]]`; which declares the `index`-th element of the array `name` as an individual syntax element having the data `typespec`. In the context of this document, `typespec name[[index]]` is only used to refer to the index in the semantics and is actually equivalent to `typespec name`.

5 Common metadata

5.1 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.

The location of a point on the sphere is identified by a pair of sphere coordinates azimuth (ϕ) and elevation (θ).

Figure 5.1 specifies the relation of the sphere coordinates azimuth (ϕ) and elevation (θ) to the X, Y, and Z coordinate axes.

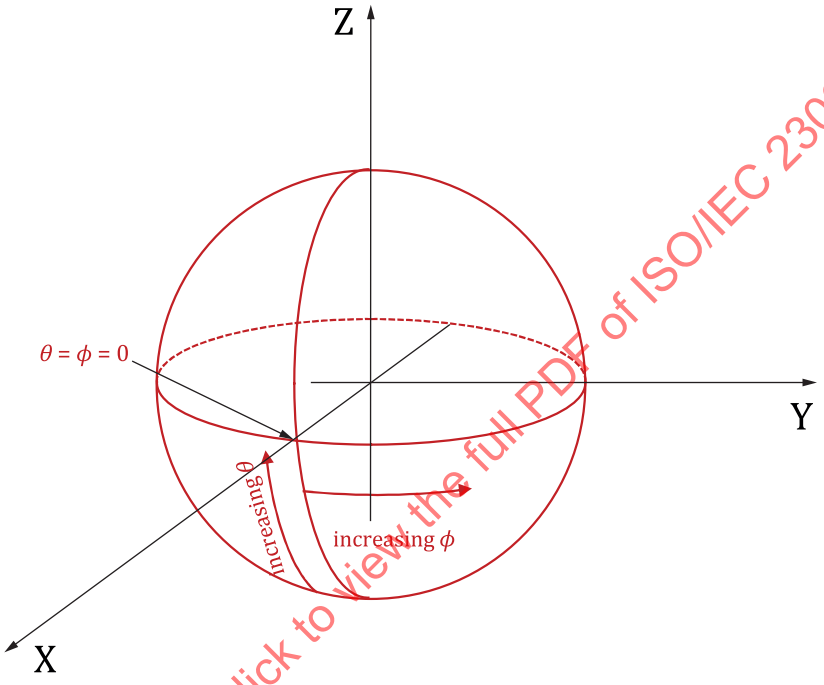


Figure 5.1 — Coordinate axes and their relation to the sphere coordinates

The value ranges of azimuth is −180.0, inclusive, to 180.0, exclusive, degrees. The value range of elevation is −90.0 to 90.0, inclusive, degrees.

5.2 Coordinate system rotation

Inputs to this process are:

— `rotation_yaw` (α_d), `rotation_pitch` (β_d), `rotation_roll` (γ_d), all in units of degrees, where `rotation_yaw` (α_d) and `rotation_roll` (γ_d), are in the range of −180.0, inclusive, to 180.0, exclusive, and `rotation_pitch` (β_d) is in the range of −90.0 to 90.0, inclusive, and

— sphere coordinates (ϕ_d, θ_d) relative to the local coordinate axes.

Outputs of this process are:

— sphere coordinates (ϕ', θ') in degrees relative to the global coordinate axes.

This process specifies rotations around the three axes of the coordinate system of 5.1 where yaw (α_d) expresses a rotation around the Z axis, pitch (β_d) rotates around the Y axis, and roll (γ_d) rotates around the X axis. Rotations are extrinsic, i.e. around X, Y, and Z fixed reference axes. The angles increase clockwise when looking from the origin towards the positive end of an axis, as illustrated in Figure 5.2.

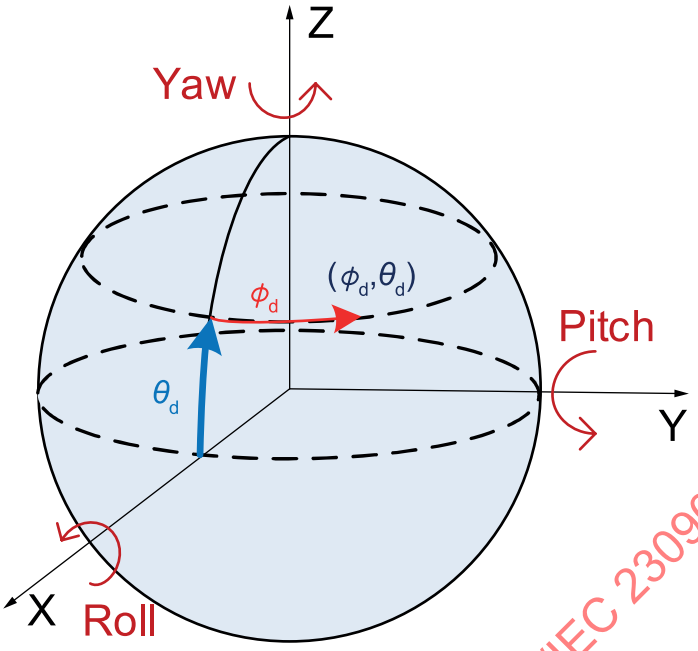


Figure 5.2 — Illustration of the directions of the yaw, pitch, and roll rotations

When any of the yaw (α_d), pitch (β_d) and roll (γ_d) rotation angles is not equal to zero, an OMAF player needs to apply the sphere rotation process specified in this clause to convert the local coordinate axes to the global coordinate axes.

It is assumed that the global coordinate systems for different media types were made aligned during content production.

The outputs are derived as follows:

$$\phi = \phi_d * \pi \div 180$$

$$\theta = \theta_d * \pi \div 180$$

$$\alpha = \alpha_d * \pi \div 180$$

$$\beta = \beta_d * \pi \div 180$$

$$\gamma = \gamma_d * \pi \div 180$$

$$x_1 = \mathrm{Cos}(\phi) * \mathrm{Cos}(\theta)$$

$$y_1 = \mathrm{Sin}(\phi) * \mathrm{Cos}(\theta)$$

$$z_1 = \mathrm{Sin}(\theta)$$

$$x_2 = \mathrm{Cos}(\beta) * \mathrm{Cos}(\alpha) * x_1 - \mathrm{Cos}(\beta) * \mathrm{Sin}(\alpha) * y_1 + \mathrm{Sin}(\beta) * z_1$$

$$y_2 = (\mathrm{Cos}(\gamma) * \mathrm{Sin}(\alpha) + \mathrm{Sin}(\gamma) * \mathrm{Sin}(\beta) * \mathrm{Cos}(\alpha)) * x_1 + (\mathrm{Cos}(\gamma) * \mathrm{Cos}(\alpha) - \mathrm{Sin}(\gamma) * \mathrm{Sin}(\beta) * \mathrm{Sin}(\alpha)) * y_1 - \mathrm{Sin}(\gamma) * \mathrm{Cos}(\beta) * z_1$$

$$z_2 = (\mathrm{Sin}(\gamma) * \mathrm{Sin}(\alpha) - \mathrm{Cos}(\gamma) * \mathrm{Sin}(\beta) * \mathrm{Cos}(\alpha)) * x_1 + (\mathrm{Sin}(\gamma) * \mathrm{Cos}(\alpha) + \mathrm{Cos}(\gamma) * \mathrm{Sin}(\beta) * \mathrm{Sin}(\alpha)) * y_1 + \mathrm{Cos}(\gamma) * \mathrm{Cos}(\beta) * z_1$$

$\phi' = \text{Atan2}(y_2, x_2) * 180 \div \pi$

$\theta' = \text{Asin}(z_2) * 180 \div \pi$

5.3 Common metadata data structures

5.3.1 Rotation structure

5.3.1.1 Definition

The fields in this structure provides the yaw, pitch, and roll angles, respectively, of the rotation to be applied to convert the local coordinate axes to the global coordinate axes. In the case of stereoscopic omnidirectional video, the fields apply to each view individually.

5.3.1.2 Syntax

```
aligned(8) class RotationStruct() {
   signed int(32) rotation_yaw;
   signed int(32) rotation_pitch;
   signed int(32) rotation_roll;
}
```

5.3.2 Content coverage structure

5.3.2.1 Definition

The fields in this structure provides the content coverage, which is expressed by one or more sphere regions covered by the content, relative to the global coordinate axes.

5.3.2.2 Syntax

```
aligned(8) class ContentCoverageStruct() {
   unsigned int(8) coverage_shape_type;
   unsigned int(8) num_regions;
   unsigned int(1) view_idc_presence_flag;
   if (view_idc_presence_flag == 0) {
      unsigned int(2) default_view_idc;
      bit(5) reserved = 0;
   } else
      bit(7) reserved = 0;
   for ( i = 0; i < num_regions; i++) {
      if (view_idc_presence_flag == 1) {
         unsigned int(2) view_idc[i];
         bit(6) reserved = 0;
      }
      SphereRegionStruct(1, 1);
   }
}
```

5.3.3 Viewpoint information structures

5.3.3.1 Definition

The `ViewpointPosStruct()`, `ViewpointGpsPositionStruct()`, `ViewpointGeomagneticInfoStruct()`, `ViewpointGlobalCoordinateSysRotationStruct()`, and `ViewpointGroupStruct()` provide information of a viewpoint, including (X, Y, Z) position of the viewpoint, GPS position of the viewpoint, geomagnetic position information for the viewpoint, and the yaw, pitch, and roll rotation angles of X, Y, and Z axes, respectively, of the global coordinate system of the viewpoint relative to the common reference coordinate system, and viewpoint group information.

5.3.3.2 Syntax

```
aligned(8) ViewpointPosStruct() {
   signed int(32) viewpoint_pos_x;
   signed int(32) viewpoint_pos_y;
   signed int(32) viewpoint_pos_z;
}
aligned(8) class ViewpointGpsPositionStruct() {
   signed int(32) viewpoint_gpspos_longitude;
   signed int(32) viewpoint_gpspos_latitude;
   signed int(32) viewpoint_gpspos_altitude;
}
aligned(8) class ViewpointGeomagneticInfoStruct() {
   signed int(32) viewpoint_geomagnetic_yaw;
   signed int(32) viewpoint_geomagnetic_pitch;
   signed int(32) viewpoint_geomagnetic_roll;
}
aligned(8) class ViewpointGlobalCoordinateSysRotationStruct() {
   signed int(32) viewpoint_gcs_yaw;
   signed int(32) viewpoint_gcs_pitch;
   signed int(32) viewpoint_gcs_roll;
}
aligned(8) class ViewpointGroupStruct() {
   unsigned int(8) vwpt_group_id;
   utf8string vwpt_group_description;
}
```

5.3.4 Sphere region structure

5.3.4.1 Definition

The sphere region structure (`SphereRegionStruct`) specifies a sphere region.

When `centre_tilt` is equal to 0, the sphere region specified by this structure is derived as follows:

— If both `azimuth_range` and `elevation_range` are equal to 0, the sphere region specified by this structure is a point on a spherical surface.

— Otherwise, the sphere region is defined using variables centreAzimuth, centreElevation, cAzimuth1, cAzimuth, cElevation1, and cElevation2 derived as follows:

centreAzimuth = `centre_azimuth` ÷ 65536
centreElevation = `centre_elevation` ÷ 65536
cAzimuth1 = (`centre_azimuth` − `azimuth_range` ÷ 2) ÷ 65536
cAzimuth2 = (`centre_azimuth` + `azimuth_range` ÷ 2) ÷ 65536
cElevation1 = (`centre_elevation` − `elevation_range` ÷ 2) ÷ 65536
cElevation2 = (`centre_elevation` + `elevation_range` ÷ 2) ÷ 65536 (5-1)

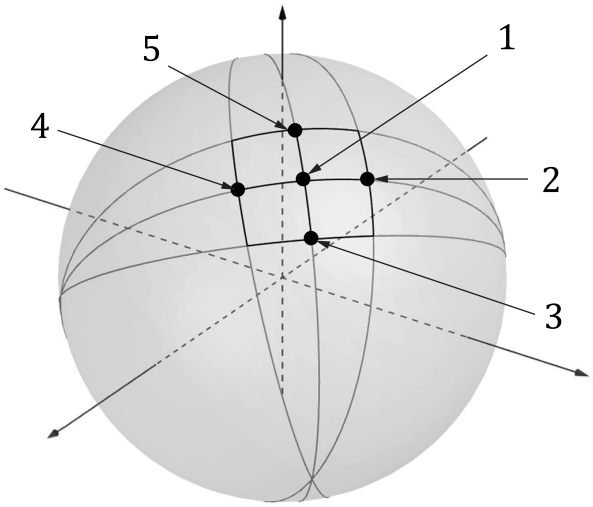
The sphere region is defined as follows with reference to the shape type value specified in the semantics of the structure containing this instance of `SphereRegionStruct`:

— When the shape type value is equal to 0, the sphere region is specified by four great circles defined by four points cAzimuth1, cAzimuth2, cElevation1, cElevation2 and the centre point defined by centreAzimuth and centreElevation and as shown in Figure 5.3.

— When the shape type value is equal to 1, the sphere region is specified by two azimuth circles and two elevation circles defined by four points cAzimuth1, cAzimuth2, cElevation1, cElevation2 and the centre point defined by centreAzimuth and centreElevation and as shown in Figure 5.4.

When `centre_tilt` is not equal to 0, the sphere region is firstly derived as above and then a tilt rotation is applied along the axis originating from the sphere origin passing through the centre point of the sphere region, where the angle value increases clockwise when looking from the origin towards the positive end of the axis. The final sphere region is the one after applying the tilt rotation.

Shape type value equal to 0 specifies that the sphere region is specified by four great circles as illustrated in Figure 5.3.

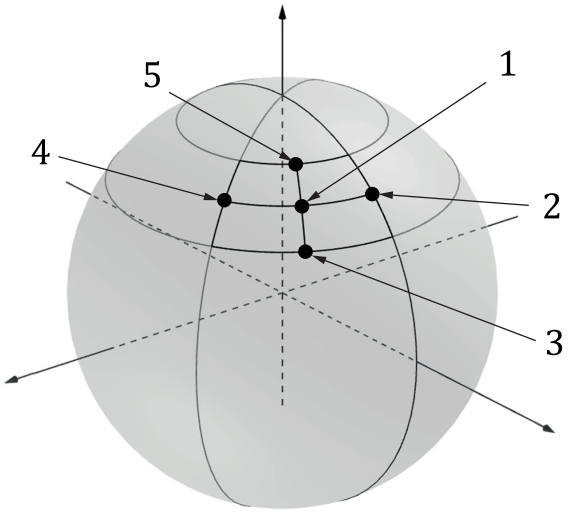


Key

1 (centreAzimuth, centreElevation)
2 cAzimuth2
3 cElevation1
4 cAzimuth1
5 cElevation2

Figure 5.3 — A sphere region specified by four great circles

Shape type value equal to 1 specifies that the sphere region is specified by two azimuth circles and two elevation circles as illustrated in Figure 5.4.



Key

1 (centreAzimuth, centreElevation)
2 cAzimuth2
3 cElevation1
4 cAzimuth1
5 cElevation2

Figure 5.4 — A sphere region specified by two azimuth circles and two elevation circles

Shape type values greater than 1 are reserved.

5.3.4.2 Syntax

```
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;
   }
}
```

5.3.5 Spherical region-wise quality ranking - Syntax

```
aligned(8) class SphereRegionQualityRankingStruct() {
   unsigned int(8) region_definition_type;
   unsigned int(8) num_regions;
   unsigned int(1) remaining_area_flag;
   unsigned int(1) view_idc_presence_flag;
   unsigned int(1) unspecified_flag;
   unsigned int(4) quality_type;
   bit(1) reserved = 0;
   if (view_idc_presence_flag == 0) {
      unsigned int(2) default_view_idc;
      bit(6) reserved = 0;
   }
   for (i = 0; i < num_regions; i++) {
      unsigned int(8) quality_ranking;
      if (view_idc_presence_flag == 1) {
         unsigned int(2) view_idc;
```

```
            bit(6) reserved = 0;
        }
        if (quality_type == 1) {
            unsigned int(16) orig_width;
            unsigned int(16) orig_height;
        }
        if ((i < (num_regions - 1)) || (remaining_area_flag == 0))
            SphereRegionStruct(1, 1);
    }
}
```

5.3.6 2D region-wise quality ranking structure- Syntax

```
aligned(8) class 2DRegionQualityRankingStruct() {
    unsigned int(8) num_regions;
    unsigned int(1) remaining_area_flag;
    unsigned int(1) view_idc_presence_flag;
    unsigned int(1) unspecified_flag;
    unsigned int(4) quality_type;
    bit(1) reserved = 0;
    if (view_idc_presence_flag == 0) {
        unsigned int(2) default_view_idc;
        bit(6) reserved = 0;
    }
    for (i = 0; i < num_regions; i++) {
        unsigned int(8) quality_ranking;
        if (view_idc_presence_flag == 1) {
            unsigned int(2) view_idc;
            bit(6) reserved = 0;
        }
        if (quality_type == 1) {
            unsigned int(16) orig_width;
            unsigned int(16) orig_height;
        }
        if ((i < (num_regions - 1)) || (remaining_area_flag == 0)) {
            unsigned int(16) left_offset;
            unsigned int(16) top_offset;
            unsigned int(16) region_width;
            unsigned int(16) region_height;
        }
    }
}
```

5.4 Common metadata semantics

5.4.1 Rotation structure - Semantics

`rotation_yaw`, `rotation_pitch`, and `rotation_roll` specify the yaw, pitch, and roll angles, respectively, of the rotation that is applied to the unit sphere to convert the local coordinate axes to the global coordinate axes, in units of 2^{-16} degrees, relative to the global coordinate axes. `rotation_yaw` shall be in the range of $-180 * 2^{16}$ to $180 * 2^{16} - 1$, inclusive. `rotation_pitch` shall be in the range of $-90 * 2^{16}$ to $90 * 2^{16}$, inclusive. `rotation_roll` shall be in the range of $-180 * 2^{16}$ to $180 * 2^{16} - 1$, inclusive.

5.4.2 Content coverage structure - Semantics

`coverage_shape_type` specifies the shape of the sphere regions expressing the content coverage. `coverage_shape_type` has the same semantics as `shape_type` specified in 5.4.1. The value of `coverage_shape_type` is used as the shape type value when applying 5.3.4 to the semantics of `ContentCoverageStruct`.

`num_regions` specifies the number of sphere regions. Value 0 is reserved.

`view_idc_presence_flag` equal to 0 specifies that `view_idc[i]` is not present. `view_idc_presence_flag` equal to 1 specifies that `view_idc[i]` is present and indicates the association of sphere regions with particular (left, right, or both) views.

`default_view_idc` equal to 0 indicates that each sphere region is monoscopic, 1 indicates that each sphere region is on the left view of a stereoscopic content, 2 indicates that each sphere region is on the right view of a stereoscopic content, 3 indicates that each sphere region is on both the left and right views.

`view_idc[i]` equal to 1 indicates that the i-th sphere region is on the left view of a stereoscopic content, 2 indicates the i-th sphere region is on the right view of a stereoscopic content, and 3 indicates that the i-th sphere region is on both the left and right views. `view_idc[i]` equal to 0 is reserved.

NOTE `view_idc_presence_flag` equal to 1 enables indicating asymmetric stereoscopic coverage. For example, one example of an asymmetric stereoscopic coverage can be described by setting `num_regions` equal to 2, indicating one sphere region to be on the left view covering the azimuth range of −90° to 90°, inclusive, and indicating the other sphere region to be on the right view covering the azimuth range of −60 to 60°, inclusive.

When `SphereRegionStruct(1, 1)` is included in the `ContentCoverageStruct()`, 5.3.4 applies and `interpolate` shall be equal to 0.

The content coverage is specified by the union of num_regions `SphereRegionStruct(1, 1)` structure(s). When `num_regions` is greater than 1, the content coverage may be non-contiguous.

5.4.3 Viewpoint information structures - Semantics

`viewpoint_pos_x`, `viewpoint_pos_y`, and `viewpoint_pos_z` specify the position of the viewpoint (when the position of the viewpoint is static) or the initial position of viewpoint (when the position of the viewpoint is dynamic), in units of 10^{-1} millimeters, in 3D space, relative to the common reference coordinate system. If a viewpoint is associated with a timed metadata track with sample entry type `'dyvp'`, the position of the viewpoint is dynamic. Otherwise, the position of the viewpoint is static. In the former case, the dynamic position of the viewpoint is signalled in the associated timed metadata track with sample entry type `'dyvp'`.

`viewpoint_gpspos_longitude` indicates the longitude of the geolocation of the viewpoint in units of 2^{-23} degrees. `viewpoint_gpspos_longitude` shall be in range of $-180 * 2^{23}$ to $180 * 2^{23} - 1$, inclusive. Positive values represent eastern longitude and negative values represent western longitude.

`viewpoint_gpspos_latitude` indicates the latitude of the geolocation of the viewpoint in units of 2^{-23} degrees. `viewpoint_gpspos_latitude` shall be in range of $-90 * 2^{23}$ to $90 * 2^{23} - 1$, inclusive. Positive value represents northern latitude and negative value represents southern latitude.

`viewpoint_gpspos_altitude` indicates the altitude of the geolocation of the viewpoint in units of milimeters above the WGS 84 reference ellipsoid as specified in the EPSG:4326 database available at https://epsg.org.

`viewpoint_geomagnetic_yaw`, `viewpoint_geomagnetic_pitch`, and `viewpoint_geomagnetic_roll` specify the yaw, pitch, and roll angles, respectively, of the rotation angles of X, Y, Z axes of the common reference coordinate system relative to the geomagnetic North direction, in units of 2^{-16} degrees. `viewpoint_geomagnetic_yaw` shall be in the range of $-180 * 2^{16}$ to $180 * 2^{16} - 1$, inclusive. `viewpoint_geomagnetic_pitch` shall be in the range of $-90 * 2^{16}$ to $90 * 2^{16}$, inclusive. `viewpoint_geomagnetic_roll` shall be in the range of $-180 * 2^{16}$ to $180 * 2^{16} - 1$, inclusive.

`viewpoint_gcs_yaw`, `viewpoint_gcs_pitch`, and `viewpoint_gcs_roll` specify the yaw, pitch, and roll angles, respectively, of the rotation angles of X, Y, Z axes of the global coordinate system of the viewpoint relative to the common reference coordinate system, in units of 2^{-16} degrees. `viewpoint_gcs_yaw` shall be in the range of $-180 * 2^{16}$ to $180 * 2^{16} - 1$, inclusive. `viewpoint_gcs_pitch` shall be in the range of $-90 * 2^{16}$ to $90 * 2^{16}$, inclusive. `viewpoint_gcs_roll` shall be in the range of $-180 * 2^{16}$ to $180 * 2^{16} - 1$, inclusive.

`vwpt_group_id` indicates the identifier of a viewpoint group. All viewpoints in a viewpoint group share a common reference coordinate system.

NOTE 1 When two viewpoints have different values of `vwpt_group_id`, their position coordinates are not comparable, because the viewpoints belong to different coordinate systems.

`vwpt_group_description` is a null-terminated UTF-8 string which indicates the description of a viewpoint group. A null string is allowed.

5.4.4 Sphere region structure - Semantics

`centre_azimuth` and `centre_elevation` specify the azimuth and elevation values, respectively, of the centre of the sphere region in units of 2^{-16} degrees. `centre_azimuth` shall be in the range of $-180 * 2^{16}$ to $180 * 2^{16} - 1$, inclusive. `centre_elevation` shall be in the range of $-90 * 2^{16}$ to $90 * 2^{16}$, inclusive.

`centre_tilt` specifies the tilt angle of the sphere region in units of 2^{-16} degrees. `centre_tilt` shall be in the range of $-180 * 2^{16}$ to $180 * 2^{16} - 1$, inclusive.

`azimuth_range` and `elevation_range`, when present, specify the azimuth and elevation ranges, respectively, of the sphere region specified by this structure in units of 2^{-16} degrees. `azimuth_range` and `elevation_range` specify the range through the centre point of the sphere region, as illustrated by Figure 5.3 or Figure 5.4. When `azimuth_range` and `elevation_range` are not present in this instance of `SphereRegionStruct`, they are inferred as specified in the semantics of the structure containing this instance of `SphereRegionStruct`. `azimuth_range` shall be in the range of 0 to $360 * 2^{16}$, inclusive. `elevation_range` shall be in the range of 0 to $180 * 2^{16}$, inclusive.

The semantics of `interpolate` are specified by the semantics of the structure containing this instance of `SphereRegionStruct`. When `interpolate` is not present in this instance of `SphereRegionStruct`, it is inferred as specified in the semantics of the syntax structure containing this instance of `SphereRegionStruct`.

5.4.5 Spherical region-wise quality ranking - Semantics

`region_definition_type` equal to 0 specifies that the sphere region is specified by four great circles. `region_definition_type` equal to 1 specifies that the sphere region is specified by two azimuth circles and two elevation circles. `region_definition_type` values greater than 1 are reserved.

`num_regions` specifies the number of quality ranking sphere regions for which the quality ranking information is given in this box. Value 0 is reserved. There shall be no point on the sphere that is contained in more than one of these quality ranking sphere regions.

`remaining_area_flag` equal to 0 specifies that all the quality ranking sphere regions are defined by the `SphereRegionStruct(1, 1)` structures. `remaining_area_flag` equal to 1 specifies that the first `num_regions - 1` quality ranking sphere regions are defined by `SphereRegionStruct(1, 1)` structure and the last remaining quality ranking sphere region is the sphere region within the content coverage, not covered by the union of the quality ranking sphere regions defined by the first `num_regions - 1` `SphereRegionStruct(1, 1)` structures. The last remaining quality ranking sphere region may be on both the left and right views.

`view_idc_presence_flag` equal to 0 specifies that `view_idc` is not present. `view_idc_presence_flag` equal to 1 specifies that `view_idc` is present and indicates the association of quality ranking sphere region with particular (left or right or both) views or monoscopic content.

`unspecified_flag` may be specified by a document that references this document.

`quality_type` indicates which factor causes the differences in the quality of packed regions on the picture. `quality_type` equal to 0 specifies that all packed regions correspond to the same projected picture resolution. `quality_type` equal to 1 specifies that at least one horRatio value, as derived in 6.2.2, may differ from other horRatio values among all pairs of packed and projected regions of the picture or at least one verRatio value, as derived in 6.2.2, may differ from other verRatio values among all pairs of packed and projected regions of the picture. `quality_type` values greater than 1 are reserved.

`default_view_idc` equal to 0 indicates that the quality ranking sphere region is monoscopic, 1 indicates that the quality ranking sphere region is on the left view of stereoscopic content, 2 indicates that the quality ranking sphere region is on the right view of stereoscopic content, 3 indicates that the quality ranking sphere region is on both the left and right views.

`quality_ranking` specifies a quality ranking value of the quality ranking sphere region. `quality_ranking` equal to 0 indicates that the quality ranking value is not defined.

`view_idc` equal to 0 indicates that the quality ranking sphere region is monoscopic, 1 indicates that the quality ranking sphere region is on the left view of stereoscopic content, 2 indicates that the quality ranking sphere region is on the right view of stereoscopic content, 3 indicates that the quality ranking sphere region is on both the left and right views. When not present, the value of `view_idc` is inferred to be equal to the value of `default_view_idc`.

`orig_width` and `orig_height` specify the width and height, respectively, of such a monoscopic projected picture for which both horRatio and verRatio, as derived in 6.2.2 for each of the packed regions that cover the quality ranking sphere region, are equal to 1.

NOTE `orig_width` and `orig_height` represent the width and height of the picture from which the packed region has been extracted without resampling.

`SphereRegionStruct(1, 1)` specifies the spherical location and size of the quality ranking sphere region relative to the global coordinate axes, while the shape type value of the quality ranking sphere regions is indicated by `region_definition_type`. The value of `interpolate` in `SphereRegionStruct(1, 1)` shall be equal to 0.

5.4.6 2D region-wise quality ranking structure - Semantics

`quality_ranking` specifies a quality ranking value of the quality ranking sphere region. `quality_ranking` equal to 0 indicates that the quality ranking value is not defined. When quality ranking region A has a non-zero quality ranking value less than that of quality ranking region B, quality ranking region A has a higher quality than quality ranking region B. When the quality ranking value is non-zero, the picture quality within the entire indicated quality ranking region is approximately constant.

`view_idc_presence_flag` equal to 0 specifies that `view_idc` is not present. `view_idc_presence_flag` equal to 1 specifies that `view_idc` is present and indicates the association of quality ranking region with particular (left or right or both) views or monoscopic content.

`view_idc` equal to 0 indicates that the quality ranking region is monoscopic, 1 indicates that the quality ranking region is on the left view of stereoscopic content, 2 indicates that the quality ranking region is on the right view of stereoscopic content, 3 indicates that the quality ranking region is on both the left and right views. When not present, the value of `view_idc` is inferred to be equal to the value of `default_view_idc`.

`default_view_idc` equal to 0 indicates that the quality ranking region is monoscopic, 1 indicates that the quality ranking region is on the left view of stereoscopic content, 2 indicates that the quality ranking region is on the right view of stereoscopic content, 3 indicates that the quality ranking region is on both the left and right views.

`num_regions` specifies the number of quality ranking 2D regions for which the quality ranking information is given in this structure. Value 0 is reserved. There shall be no pixel of the decoded picture that is contained in more than one of these quality ranking 2D regions.

`remaining_area_flag` equal to 0 specifies that all the quality ranking 2D regions are defined by the `left_offset`, `top_offset`, `region_width`, and `region_height`. `remaining_area_flag` equal to 1 specifies that the first `num_regions − 1` quality ranking 2D regions are defined by `left_offset`, `top_offset`, `region_width`, and `region_height` and the last remaining quality ranking 2D region is the area in the picture with width equal to `width` of `VisualSampleEntry` and height equal to `height`

of `VisualSampleEntry`, not covered by the union of the first `num_regions - 1` quality ranking 2D regions. The last remaining quality ranking 2D region may be on both the left and right views.

`unspecified_flag` may be specified by a document that references this document.

`quality_type` indicates which factor causes the differences in the quality of packed regions on the picture. `quality_type` equal to 0 specifies that all packed regions correspond to the same projected picture resolution. `quality_type` equal to 1 specifies that at least one horRatio value, as derived in 6.2.2, may differ from other horRatio values among all pairs of packed and projected regions of the picture or at least one verRatio value, as derived in 6.2.2, may differ from other verRatio values among all pairs of packed and projected regions of the picture. `quality_type` values greater than 1 are reserved.

`orig_width` and `orig_height` specify the width and height, respectively, of such a monoscopic projected picture for which both horRatio and verRatio, as derived in 6.2.2 for each of the packed regions that cover the quality ranking 2D region, are equal to 1.

NOTE `orig_width` and `orig_height` represent the width and height of the picture from which the packed region has been extracted without resampling.

`left_offset`, `top_offset`, `region_width`, and `region_height` are integer values that indicate the position and size of the quality ranking 2D region. `left_offset` and `top_offset` indicate the horizontal and vertical coordinates, respectively, of the upper left corner of the quality ranking 2D region within the picture, in units of luma samples. `region_width` and `region_height` indicate the width and height, respectively, of the quality ranking 2D region within the picture, in units of luma samples. `left_offset` + `region_width` shall be less than `width` of `VisualSampleEntry`. `top_offset` + `region_height` shall be less than `height` of `VisualSampleEntry`.

`region_width` shall be greater than 0.

`region_height` shall be greater than 0.

6 Video and image metadata

6.1 Projection formats

6.1.1 List of projection formats

This clause specifies the inverse of the projection process for remapping of one sample location of a monoscopic projected luma picture onto a position on the unit sphere identified by a pair of azimuth and elevation coordinates.

The omnidirectional projection formats specified in this document are identified by a 5-bit unsigned integer value. Table 6.1 specifies the omnidirectional projection format identifier values and provides references to the subclauses in which the respective inverse projection processes are specified.

Table 6.1 — Omnidirectional projection formats

Identifier value	Omnidirectional projection	Reference
0	Equirectangular projection	6.1.2
1	Cubemap projection	6.1.3
2..31	Reserved	N/A

6.1.2 Equirectangular projection process

Inputs to this process are:

— pictureWidth and pictureHeight, which are the width and height, respectively, of a monoscopic projected luma picture, in relative projected picture sample units, and

— the centre point of a sample location (hPos, vPos) along the horizontal and vertical axes, respectively, where hPos and vPos are in relative projected picture sample units and may have non-integer real values.

Outputs of this process are:

— sphere coordinates (ϕ, θ) for the sample location in degrees relative to the coordinate axes specified in 5.1.

The sphere coordinates (ϕ, θ) for the luma sample location, in degrees, are given by the following equations:

$$\phi = (0.5 - hPos \div pictureWidth) * 360$$
$$\theta = (0.5 - vPos \div pictureHeight) * 180$$

Figure 6.1 illustrates the azimuth and elevation ranges of a monoscopic projected picture with the equirectangular projection.

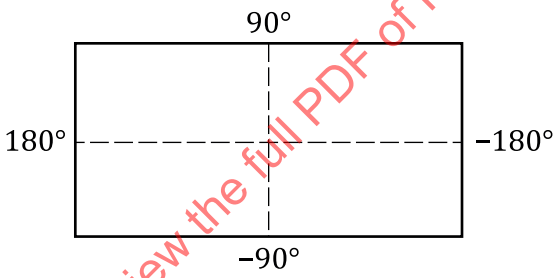


Figure 6.1 — Azimuth and elevation ranges of the monoscopic projected picture of the equirectangular projection

NOTE 1 Since an input to this process is the centre point of a sample location and the width and height of the sample are non-zero, the output ϕ is never equal to −180° or 180° and the output θ is never equal to −90° or 90°.

NOTE 2 The monoscopic projected picture represents the inside surface of the unit sphere observed from the origin of the coordinate system. Thus, the azimuth decreases from left to right.

6.1.3 Cubemap projection process

Inputs to this process are:

— pictureWidth and pictureHeight, which are the width and height, respectively, of a monoscopic projected luma picture, in relative projected picture sample units, and

— the centre point of a sample location (hPos, vPos) along the horizontal and vertical axes, respectively, where hPos and vPos are in relative projected picture sample units and may have non-integer real values.

Outputs of this process are:

— sphere coordinates (ϕ, θ) for the sample location in degrees relative to the coordinate axes specified in 5.1.

Figure 6.2 illustrates the cube face arrangement in the projected picture of the cubemap projection format and the mapping of the cube faces onto the coordinate axes specified in 5.1, where PX, NX, PY, NY, PZ, and NZ denote positive X, negative X, positive Y, negative Y, positive Z, and negative Z, respectively.

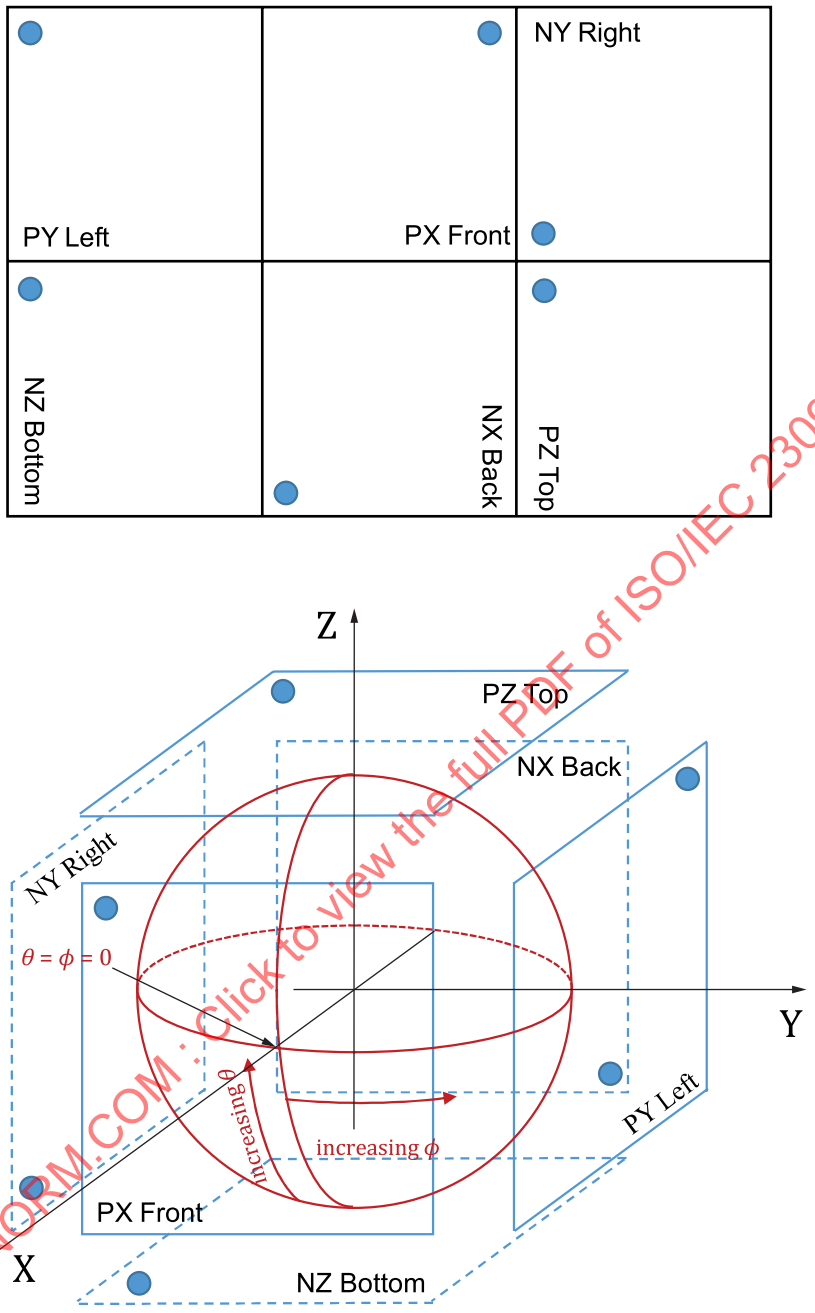


Figure 6.2 — Relation of the cube face arrangement of the projected picture to the sphere coordinates

The values of pictureWidth and pictureHeight shall be such that pictureWidth is a multiple of 3, pictureHeight is a multiple of 2, and pictureWidth / 3 is equal to pictureHeight / 2.

The sphere coordinates (ϕ, θ) for the luma sample location, in degrees, are given by the following equations:

lw = pictureWidth / 3

lh = pictureHeight / 2

tmpHorVal = hPos − Floor(hPos ÷ lw) * lw

tmpVerVal = vPos − Floor(vPos ÷ lh) * lh

hPos′ = −(2 * tmpHorVal ÷ lw) + 1

vPos′ = −(2 * tmpVerVal ÷ lh) + 1

w = Floor(hPos ÷ lw)

h = Floor(vPos ÷ lh)

if(*w* = = 1 && *h* = = 0) { // positive *x* front face

x = 1.0

y = hPos′

z = vPos′

} else if(*w* = = 1 && *h* = = 1) { // negative *x* back face

x = −1.0

y = −vPos′

z = −hPos′

} else if(*w* = = 2 && *h* = = 1) { // positive *z* top face

x = −hPos′

y = −vPos′

z = 1.0

} else if(*w* = = 0 && *h* = = 1) { // negative *z* bottom face

x = hPos′

y = −vPos′

z = −1.0

} else if(*w* = = 0 && *h* = = 0) { // positive *y* left face

x = −hPos′

y = 1.0

z = vPos′

} else { // (*w* = = 2 && *h* = = 0), negative *y* right face

x = hPos′

y = −1.0

z = vPos′

}

ϕ = Atan2(*y*, *x*) * 180 ÷ π

$$\theta = \text{Asin}\left(z \div \sqrt{x^2 + y^2 + z^2}\right) * 180 \div \pi$$

6.2 Region-wise packing formats

6.2.1 List of packing formats

This clause specifies the inverse processes of the region-wise packing for remapping of a luma sample location in a packed region onto a luma sample location of the corresponding projected region.

The inverse region-wise packing processes specified in this document are identified by a 4-bit unsigned integer value. Table 6.2 specifies the region-wise packing format identifier values and provides references to the subclauses in which the respective inverse processes are specified.

Table 6.2 — Region-wise packing formats

Identifier value	Region-wise packing format	Reference
0	Rectangular region-wise packing	6.2.2
1..15	Reserved	N/A

6.2.2 Rectangular region-wise packing process

This clause specifies the inverse of the rectangular region-wise packing process for remapping of a luma sample location in a packed region onto a luma sample location of the corresponding projected region.

Inputs to this process are:

— sample location (x, y) within the packed region, where x and y are in relative packed picture sample units, while the sample location is at an integer sample location within the packed picture,

— the width and the height (projRegWidth, projRegHeight) of the projected region, in relative projected picture sample units,

— the width and the height (packedRegWidth, packedRegHeight) of the packed region, in relative packed picture sample units,

— transform type (transformType), and

— offset values for the sampling position (offsetX, offsetY) in the range of 0, inclusive, to 1, exclusive, in horizontal and vertical relative packed picture sample units, respectively.

NOTE offsetX and offsetY both equal to 0.5 indicate a sampling position that is in the centre point of a sample in packed picture sample units.

Outputs of this process are:

— the centre point of the sample location (hPos, vPos) within the projected region, where hPos and vPos are in relative projected picture sample units and may have non-integer real values.

The outputs are derived as follows:

```
if( transformType = = 0 | | transformType = = 1 | | transformType = = 2 | | transformType = = 3 ) {
    horRatio = projRegWidth ÷ packedRegWidth
    verRatio = projRegHeight ÷ packedRegHeight
} else if ( transformType = = 4 | | transformType = = 5 | | transformType = = 6 | |
    transformType = = 7 ) {
```

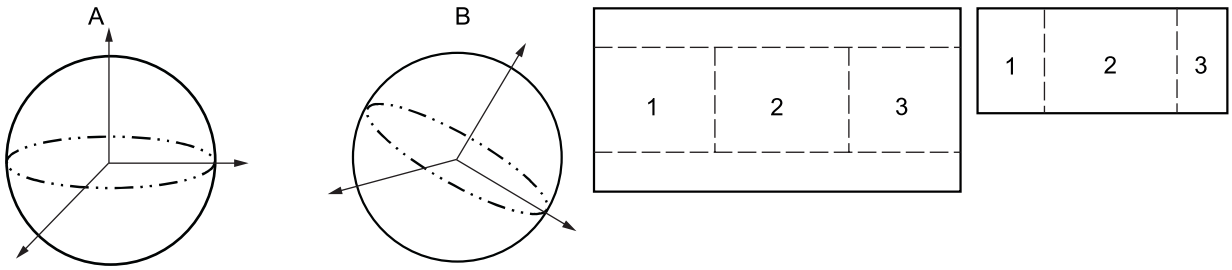
```
        horRatio = projRegWidth ÷ packedRegHeight
        verRatio = projRegHeight ÷ packedRegWidth
    }
    if( transformType = = 0 ) {
        hPos = horRatio * ( x + offsetX )
        vPos = verRatio * ( y + offsetY )
    } else if ( transformType = = 1 ) {
        hPos = horRatio * ( packedRegWidth − x − offsetX )
        vPos = verRatio * ( y + offsetY )
    } else if ( transformType = = 2 ) {
        hPos = horRatio * ( packedRegWidth − x − offsetX )
        vPos = verRatio * ( packedRegHeight − y − offsetY )
    } else if ( transformType = = 3 ) {
        hPos = horRatio * ( x + offsetX )
        vPos = verRatio * ( packedRegHeight − y − offsetY )
    } else if ( transformType = = 4 ) {
        hPos = horRatio * ( y + offsetY )
        vPos = verRatio * ( x + offsetX )
    } else if ( transformType = = 5 ) {
        hPos = horRatio * ( y + offsetY )
        vPos = verRatio * ( packedRegWidth − x − offsetX )
    } else if ( transformType = = 6 ) {
        hPos = horRatio * ( packedRegHeight − y − offsetY )
        vPos = verRatio * ( packedRegWidth − x − offsetX )
    } else if ( transformType = = 7 ) {
        hPos = horRatio * ( packedRegHeight − y − offsetY )
        vPos = verRatio * ( x + offsetX )
    }
```

6.3 Sample location mapping process

6.3.1 Relation of decoded pictures to global coordinate axes

Figure 6.3 illustrates the conversions from a spherical picture to a packed picture that can be used in content authoring and the corresponding conversions from a packed picture to a spherical picture to be rendered that can be used in an OMAF player. The example in this clause is described for a packed

picture that appears in a projected omnidirectional video track. Similar description can be derived for an image item.



a) Unit sphere aligned with global coordinate axes

b) Unit sphere aligned with local coordinate axes

c) Projected picture on which regions for region-wise packing are specified

d) Packed picture

Key

A global coordinate axes

B local coordinate axes

Figure 6.3 — Example of processing stages to derive a packed picture from a spherical image or vice versa

The content authoring can include the following ordered steps:

— The source images provided as input are stitched to generate a sphere picture on the unit sphere per the global coordinate axes as indicated in Figure 6.3 a).

— The unit sphere is then rotated relative to the global coordinate axes, as indicated in Figure 6.3 b). The amount of rotation to convert from the local coordinate axes to the global coordinate axes is specified by the rotation angles indicated in the `RotationBox`. The local coordinate axes of the unit sphere are the axes of the coordinate system that has been rotated. The absence of `RotationBox` indicates that the local coordinate axes are the same as the global coordinate axes.

— As illustrated in Figure 6.3 c), the spherical picture on the rotated unit sphere is then converted to a two-dimensional projected picture, for example, using the equirectangular projection specified in 6.1.1. When spatial packing of stereoscopic content is applied, two spherical pictures for the two views are converted to two constituent pictures, after which frame packing is applied to pack the two constituent pictures to one projected picture.

— Rectangular region-wise packing can be applied to obtain a packed picture from the projected picture. One example of packing is depicted in Figure 6.3 c) and Figure 6.3 d). The dashed rectangles in Figure 6.3 c) indicate the projected regions on a projected picture, and the respective areas in Figure 6.3 d) indicate the corresponding packed regions. In this example, projected regions 1 and 3 are horizontally downsampled, while projected region 2 is kept at its original resolution.

`CoverageInformationBox` can be used to indicate which part of the sphere is covered by the packed picture.

In order to map sample locations of a packed picture (such as that in Figure 6.3 d)) to a unit sphere used in rendering (Figure 6.3 a)), the OMAF player can perform the following ordered steps:

— A packed picture, such as that in Figure 6.3 d), is obtained as a result of decoding a picture from a video track or an image item.

— If needed, chroma sample arrays of the packed picture are upsampled to the resolution of the luma sample array of the packed picture, and colour space conversion can also be performed.

— If region-wise packing is indicated, the sample locations of the packed picture are converted to sample locations of the respective projected picture, such as that in Figure 6.3 c), as specified in 6.2.2. Otherwise, the projected picture is identical to the packed picture.

— If spatial frame packing of the projected picture is indicated, the sample locations of the projected picture are converted to sample locations of the respective constituent picture of the projected picture, as specified in 6.3.3. Otherwise, the constituent picture of the projected picture is identical to the projected picture.

— The sample locations of a constituent picture the projected picture are converted to sphere coordinates that are relative to local coordinate axes, as specified for the omnidirectional projection format being used in 6.5.1. The resulting sample locations correspond to a sphere picture depicted in in Figure 6.3b.

— If rotation is indicated, the sphere coordinates relative to the local coordinate axes are converted to sphere coordinates relative to the global coordinate axes as specified in 5.2. Otherwise, the global coordinate axes are identical to the local coordinate axes.

For projected omnidirectional video, the overall process for mapping of luma sample locations within a decoded picture to sphere coordinates relative to the global coordinate axes is normatively specified in 6.3.2.

For fisheye omnidirectional video, the process for mapping of luma sample locations within an active area with a decoded picture to sphere coordinates relative to the global coordinate axes is normatively specified in 6.3.4.

6.3.2 Mapping of luma sample locations within a decoded picture to sphere coordinates relative to the global coordinate axes

This clause specifies the semantics of luma sample locations within a decoded picture to sphere coordinates relative to the global coordinate axes. The decoded picture may be of any of the following:

— For video, the decoded picture is the decoding output resulting from a sample of the video track.

— For an image item, the decoded picture is a reconstructed image of the image item.

offsetX is set equal to 0.5 and offsetY is set equal to 0.5.

If RegionWisePackingFlag is equal to 1, the following applies for each packed region n in the range of 0 to NumRegions − 1, inclusive:

— For each sample location (xPackedPicture, yPackedPicture) belonging to the n-th packed region with PackingType[n] equal to 0 (i.e. with rectangular region-wise packing), the following applies:

  — The corresponding sample location (xProjPicture, yProjPicture) of the projected picture is derived as follows:

    — *x* is set equal to xPackedPicture – PackedRegLeft[n].

    — *y* is set equal to yPackedPicture – PackedRegTop[n].

    — 6.2.2 is invoked with *x*, *y*, PackedRegWidth[n], PackedRegHeight[n], ProjRegWidth[n], ProjRegHeight[n], TransformType[n], offsetX, and offsetY as inputs, and the output is assigned to sample location (hPos, vPos).

    — xProjPicture is set equal to ProjRegLeft[n] + hPos.

    — When SideBySideFlag is equal to 0, and when xProjPicture is greater than or equal to `proj_picture_width`, xProjPicture is set equal to xProjPicture − `proj_picture_width`.

- When SideBySideFlag is equal to 1, the following applies:
  - When ProjRegLeft[n] is less than `proj_picture_width` / 2 and xProjPicture is greater than or equal to `proj_picture_width` / 2, xProjPicture is set equal to xProjPicture – `proj_picture_width` / 2.
  - When ProjRegLeft[n] is greater than or equal to `proj_picture_width` / 2 and xProjPicture is greater than or equal to `proj_picture_width`, xProjPicture is set equal to xProjPicture – `proj_picture_width` / 2.
- yProjPicture is set equal to ProjRegTop[n] + vPos.

— 6.3.3 is invoked with xProjPicture, yProjPicture, ConstituentPicWidth, and ConstituentPicHeight as inputs, and the outputs indicating the sphere coordinates and the constituent frame index (for frame-packed stereoscopic video) for the luma sample location (xPackedPicture, yPackedPicture) belonging to the n-th packed region in the decoded picture.

Otherwise (RegionWisePackingFlag is equal to 0), the following applies for each sample location (*x*, *y*) within the decoded picture:

— xProjPicture is set equal to *x* + offsetX.

— yProjPicture is set equal to *y* + offsetY.

— 6.3.3 is invoked with xProjPicture, yProjPicture, ConstituentPicWidth, and ConstituentPicHeight as inputs, and the outputs indicating the sphere coordinates and the constituent frame index (for frame-packed stereoscopic video) for the sample location (*x*, *y*) within the decoded picture.

6.3.3 Conversion from a sample location in a projected picture to sphere coordinates relative to the global coordinate axes

Inputs to this process are

— the centre point of a sample location (xProjPicture, yProjPicture) within a projected picture, where xProjPicture and yProjPicture are in relative projected picture sample units and may have non-integer real values, and

— pictureWidth and pictureHeight, which are the width and height, respectively, of a monoscopic projected luma picture, in relative projected picture sample units.

NOTE The projected picture for which the sample location (xProjPicture, yProjPicture) is given as input can be a spatially frame-packed picture.

Outputs of this process are:

— sphere coordinates (azimuthGlobal, elevationGlobal), in units of degrees relative to the global coordinate axes, and

— when SpatiallyPackedStereoFlag is equal to 1, the index of the constituent picture (constituentPicture) equal to 0 or 1.

The outputs are derived with the following ordered steps:

— If xProjPicture is greater than or equal to pictureWidth or yProjPicture is greater than or equal to pictureHeight, the following applies:
  - constituentPicture is set equal to 1.
  - If xProjPicture is greater than or equal to pictureWidth, xProjPicture is set to xProjPicture – pictureWidth.

— If yProjPicture is greater than or equal to pictureHeight, yProjPicture is set to yProjPicture – pictureHeight.

— Otherwise, constituentPicture is set equal to 0.

— Depending on the projection format, the following applies:

— When the projection format is the equirectangular projection, 6.1.2 is invoked with pictureWidth, pictureHeight, xProjPicture, and yProjPicture as inputs, and the output is assigned to azimuthLocal, elevationLocal.

— When the projection format is the cubemap projection, 6.1.3 is invoked with pictureWidth, pictureHeight, xProjPicture, and yProjPicture as inputs, and the output is assigned to azimuthLocal, elevantionLocal.

— If RotationFlag is equal to 1, 5.2 is invoked with azimuthLocal, elevantionLocal, `rotation_yaw` ÷ 2^{16}, `rotation_pitch` ÷ 2^{16}, and `rotation_roll` ÷ 2^{16} as inputs, and the output is assigned to azimuthGlobal and elevationGlobal.

— Otherwise, azimuthGlobal is set equal to azimuthLocal and elevationGlobal is set equal to elevationLocal.

6.3.4 Conversion from a sample location of an active area in a fisheye decoded picture to sphere coordinates relative to the global coordinate axes

Inputs to this process are:

— the sample location (x, y) in units of luma samples,

— the centre location (x_c, y_c) and the radius (r_c) of the circular region that contains the i-th active area, given by `circular_image_centre_x`, `circular_image_centre_y`, and `circular_image_radius`, respectively, all in units of 2^{-16} luma samples,

— the field of view (θ_v) of the lens corresponding to the i-th active area, given by `field_of_view`, in units of 2^{-16} degrees,

— the rotation parameters $(\alpha_c, \beta_c, \gamma_c)$, given by `camera_centre_azimuth`, `camera_centre_elevation`, and `camera_centre_tilt`, respectively, all in units of 2^{-16} degrees,

— the lens projection type of the lens corresponding to the i-th active area given by `lens_projection_type`, and

— the number of polynomial coefficients numCoeffs and the polynomial coefficients coeffVal of the i-th active area, given by `num_polynomial_coeffs` and `polynomial_coeff`, respectively.

Outputs of this process are:

— sphere coordinates (ϕ, θ) relative to the global coordinate axes.

The method of converting a sample location of an active area to sphere coordinates is determined as follows:

— If numCoeffs is equal to 0, there is only one method of converting a sample location of an active area to sphere coordinates that is specified, which is to not use polynomial coefficients.

— Otherwise (numCoeffs is not equal to 0), there are two methods of converting a sample location of an active area to sphere coordinates that are specified, which are to not use polynomial coefficients or to use polynomial coefficients. The method using polynomial coefficients is preferred, as this method is intended to provide a more precise model of the fisheye characteristics. However, the other method may also be appropriate for some uses, as it provides a single conversion process that can be used regardless of whether numCoeffs is equal to 0 or not. This document does not prescribe which of the two methods is to be used in this case.

The outputs are derived as follows:

— If polynomial coefficients are not used and

if `lens_projection_type` equal to 0, the angle ϕ' is derived by

$$\phi' = (\text{Sqrt}((x - x_c \div 2^{16})^2 + (y - y_c \div 2^{16})^2) \div (r_c \div 2^{16})) * (\theta_v \div 2^{16} * \pi \div 180) \div 2$$

if `lens_projection_type` equal to 1, the angle ϕ' is derived by

$$\phi' = \text{Atan}(\text{Sqrt}((x - x_c \div 2^{16})^2 + (y - y_c \div 2^{16})^2) \div (r_c \div 2^{16})) * (\theta_v \div 2^{16} * \pi \div 180) \div 2$$

if `lens_projection_type` equal to 2, the angle ϕ' is derived by

$$\phi' = \text{Atan}(\text{Sqrt}((x - x_c \div 2^{16})^2 + (y - y_c \div 2^{16})^2) \div (r_c \div 2^{16})) * (\theta_v \div 2^{16} * \pi \div 180)$$

if `lens_projection_type` equal to 3, the angle ϕ' is derived by

$$\phi' = \text{Asin}(\text{Sqrt}((x - x_c \div 2^{16})^2 + (y - y_c \div 2^{16})^2) \div (r_c \div 2^{16})) * (\theta_v \div 2^{16} * \pi \div 180) \div 2$$

if `lens_projection_type` equal to 4, the angle ϕ' is derived by

$$\phi' = \text{Asin}(\text{Sqrt}((x - x_c \div 2^{16})^2 + (y - y_c \div 2^{16})^2) \div (r_c \div 2^{16})) * (\theta_v \div 2^{16} * \pi \div 180)$$

— Otherwise (polynomial coefficients are used), the angle ϕ' is derived by

$$\phi' = \sum_{j=0}^{numCoeffs-1} ((\text{coeffVal}[j] * 2^{-24}) * (\text{Sqrt}((x - x_c * 2^{-16})^2 + (y - y_c * 2^{-16})^2) \div (r_c * 2^{-16}))^j)$$

The outputs are then derived as follows:

$$\theta' = \text{Atan2}(y - y_c \div 2^{16}, x - x_c \div 2^{16})$$

$$x_1 = \text{Cos}(\phi')$$

$$y_1 = \text{Sin}(\phi') * \text{Cos}(\theta')$$

$$z_1 = \text{Sin}(\phi') * \text{Sin}(\theta')$$

$$\alpha = (\alpha_c \div 2^{16}) * \pi \div 180$$

$$\beta = (\beta_c \div 2^{16}) * \pi \div 180$$

$$\gamma = (\gamma_c \div 2^{16}) * \pi \div 180$$

$$x_2 = \text{Cos}(\beta) * \text{Cos}(\gamma) * x_1 - \text{Cos}(\beta) * \text{Sin}(\gamma) * y_1 + \text{Sin}(\beta) * z_1$$

$$y_2 = (\text{Cos}(\alpha) * \text{Sin}(\gamma) + \text{Sin}(\alpha) * \text{Sin}(\beta) * \text{Cos}(\gamma)) * x_1 +$$
$$(\text{Cos}(\alpha) * \text{Cos}(\gamma) - \text{Sin}(\alpha) * \text{Sin}(\beta) * \text{Sin}(\gamma)) * y_1 -$$
$$\text{Sin}(\alpha) * \text{Cos}(\beta) * z_1$$

$$z_2 = (\text{Sin}(\alpha) * \text{Sin}(\gamma) - \text{Cos}(\alpha) * \text{Sin}(\beta) * \text{Cos}(\gamma)) * x_1 +$$
$$(\text{Sin}(\alpha) * \text{Cos}(\gamma) + \text{Cos}(\alpha) * \text{Sin}(\beta) * \text{Sin}(\gamma)) * y_1 +$$
$$\text{Cos}(\alpha) * \text{Cos}(\beta) * z_1$$

$$\phi = \text{Atan2}(y_2, x_2) * 180 \div \pi$$

$$\theta = \text{Asin}(z_2) * 180 \div \pi$$

6.4 Fisheye omnidirectional video

Without the projection and region-wise packing processes specified in 6.1 and 6.2, multiple circular images captured by fisheye cameras may be directly projected onto a picture, which consists of fisheye omnidirectional video. In an OMAF player, the decoded fisheye omnidirectional video may be stitched and rendered according to the user's intended viewport using the signalled fisheye video parameters, including:

— region information of circular images in the coded picture,

— field of view and camera parameters of fisheye lens,

— lens distortion correction (LDC) parameters with local variation of FOV, and

— lens shading compensation (LSC) parameters with RGB gains.

6.5 Video and image metadata data structures

6.5.1 Projection format structure - Syntax

```
aligned(8) class ProjectionFormatStruct() {
      bit(3) reserved = 0;
      unsigned int(5) projection_type;
}
```

6.5.2 Region-wise packing structure

6.5.2.1 Definition

`RegionWisePackingStruct` specifies the mapping between packed regions and the respective projected regions and specifies the location and size of the guard bands, if any.

NOTE Among other information, the `RegionWisePackingStruct` also provides the content coverage information in the 2D Cartesian picture domain.

A decoded picture in the semantics of this clause is either one of the following depending on the container for this syntax structure:

— For video, the decoded picture is the decoding output resulting from a sample of the video track.

— For an image item, the decoded picture is a reconstructed image of the image item.

The content of `RegionWisePackingStruct` is informatively summarized below, while the normative semantics follow subsequently in this clause:

— The width and height of the projected picture are explicitly signalled with `proj_picture_width` and `proj_picture_height`, respectively.

— The width and height of the packed picture are explicitly signalled with `packed_picture_width` and `packed_picture_height`, respectively.

— When the projected picture is stereoscopic and has the top-bottom or side-by-side frame packing arrangement, `constituent_picture_matching_flag` equal to 1 specifies that

  — the projected region information, packed region information, and guard band region information in this syntax structure apply individually to each constituent picture,

  — the packed picture and the projected picture have the same stereoscopic frame packing format, and

  - the number of projected regions and packed regions is double of that indicated by the value of `num_regions` in the syntax structure.

- `RegionWisePackingStruct` contains a loop, in which a loop entry corresponds to the respective projected regions and packed regions in both constituent pictures (when `constituent_picture_matching_flag` equal to 1) or to a projected region and the respective packed region (when `constituent_picture_matching_flag` equal to 0), and the loop entry the contains the following:
  - a flag indicating the presence of guard bands for the packed region,
  - the packing type (however, only rectangular region-wise packing is specified in this document),
  - the mapping between a projected region and the respective packed region in the rectangular region packing structure `RectRegionPacking(i)`,
  - when guard bands are present, the guard band structure for the packed region `GuardBand(i)`.

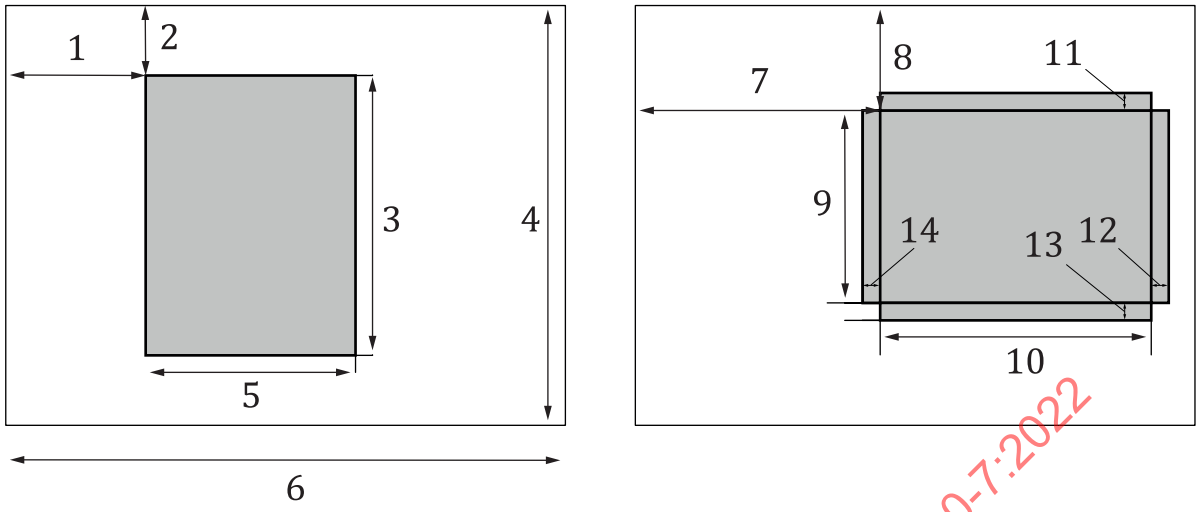
The content of the rectangular region packing structure `RectRegionPacking(i)` is informatively summarized below, while the normative semantics follow subsequently in this clause:

- `proj_reg_width[i]`, `proj_reg_height[i]`, `proj_reg_top[i]`, and `proj_reg_left[i]` specify the width, height, top offset, and left offset, respectively, of the i-th projected region.
- `transform_type[i]` specifies the rotation and mirroring, if any, that are applied to the i-th packed region to remap it to the i-th projected region.
- `packed_reg_width[i]`, `packed_reg_height[i]`, `packed_reg_top[i]`, and `packed_reg_left[i]` specify the width, height, the top offset, and the left offset, respectively, of the i-th packed region.

The content of the guard band structure `GuardBand(i)` is informatively summarized below, while the normative semantics follow subsequently in this clause:

- `left_gb_width[i]`, `right_gb_width[i]`, `top_gb_height[i]`, or `bottom_gb_height[i]` specify the guard band size on the left side of, the right side of, above, or below, respectively, the i-th packed region.
- `gb_not_used_for_pred_flag[i]` indicates if the encoding was constrained in a manner that guards bands are not used as a reference in the inter prediction process.
- `gb_type[i][j]` specifies the type of the guard bands for the i-th packed region.

Figure 6.4 illustrates an example of the position and size of a projected region within a projected picture (on the left side) as well as that of a packed region within a packed picture with guard bands (on the right side). This example applies when the value of `constituent_picture_matching_flag` is equal to 0.



Key

1 proj_reg_left[i]
2 proj_reg_top[i]
3 proj_reg_height[i]
4 proj_picture_height
5 proj_reg_width[i]
6 proj_picture_width
7 packed_reg_left[i]
8 packed_reg_top[i]
9 packed_reg_height[i]
10 packed_reg_width[i]
11 top_gb_height[i]
12 right_gb_width[i]
13 bottom_gb_height[i]
14 left_gb_width[i]

Figure 6.4 — Projected region and the corresponding packed region with guard bands

This clause is organized as follows:

— The syntax and semantics of the rectangular region packing structure are specified in 6.5.2.2 and 6.6.2.2, respectively.

— The syntax and semantics of the guard band structure are specified in 6.5.2.3 and 6.6.2.3, respectively.

— The syntax and semantics of the region-wise packing structure are specified in 6.5.2.4 and 6.6.2.4, respectively.

— 6.6.2.5 derives variables from syntax element values of the rectangular region packing, guard band, region-wise packing structures. 6.6.2.5 also uses the variables to specify constraints for the syntax element values. The variables are also used in other clauses.

6.5.2.2 Syntax of the rectangular region packing structure

```
aligned(8) class RectRegionPacking(i) {
   unsigned int(32) proj_reg_width[i];
   unsigned int(32) proj_reg_height[i];
   unsigned int(32) proj_reg_top[i];
   unsigned int(32) proj_reg_left[i];
```

```
   unsigned int(3)  transform_type[i];
   bit(5) reserved = 0;
   unsigned int(16) packed_reg_width[i];
   unsigned int(16) packed_reg_height[i];
   unsigned int(16) packed_reg_top[i];
   unsigned int(16) packed_reg_left[i];
}
```

6.5.2.3 Syntax of the guard band structure

```
aligned(8) class GuardBand(i) {
   unsigned int(8) left_gb_width[i];
   unsigned int(8) right_gb_width[i];
   unsigned int(8) top_gb_height[i];
   unsigned int(8) bottom_gb_height[i];
   unsigned int(1) gb_not_used_for_pred_flag[i];
   for (j = 0; j < 4; j++)
      unsigned int(3) gb_type[i][j];
   bit(3) reserved = 0;
}
```

6.5.2.4 Syntax of the region-wise packing structure

```
aligned(8) class RegionWisePackingStruct() {
   unsigned int(1) constituent_picture_matching_flag;
   bit(7) reserved = 0;
   unsigned int(8) num_regions;
   unsigned int(32) proj_picture_width;
   unsigned int(32) proj_picture_height;
   unsigned int(16) packed_picture_width;
   unsigned int(16) packed_picture_height;
   for (i = 0; i < num_regions; i++) {
      bit(3) reserved = 0;
      unsigned int(1) guard_band_flag[i];
      unsigned int(4) packing_type[i];
      if (packing_type[i] == 0) {
         RectRegionPacking(i);
         if (guard_band_flag[i])
            GuardBand(i);
      }
   }
}
```

6.5.3 Fisheye omnidirectional video structure

6.5.3.1 Syntax of the fisheye video essential information structure

```
aligned(8) class FisheyeVideoEssentialInfoStruct() {
   unsigned int(3) view_dimension_idc;
   bit(21) reserved = 0;
   unsigned int(8) num_circular_images;
   for (i=0; i< num_circular_images; i++) {
      unsigned int(32) circular_image_centre_x;
      unsigned int(32) circular_image_centre_y;
      unsigned int(32) rect_region_top;
      unsigned int(32) rect_region_left;
      unsigned int(32) rect_region_width;
      unsigned int(32) rect_region_height;
      unsigned int(32) circular_image_radius;
      unsigned int(32) scene_radius;
      signed int(32) camera_centre_azimuth;
      signed int(32) camera_centre_elevation;
      signed int(32) camera_centre_tilt;
      unsigned int(32) camera_centre_offset_x;
      unsigned int(32) camera_centre_offset_y;
      unsigned int(32) camera_centre_offset_z;
      unsigned int(32) field_of_view;
      bit(16) reserved = 0;
      unsigned int(16) num_polynomial_coeffs;
      for (j=0; j< num_polynomial_coeffs; j++)
          signed int(32) polynomial_coeff;
```

```
    }
}
```

6.5.3.2 Syntax of the fisheye video supplemental information structure

```
aligned(8) class FisheyeVideoSupplementalInfoStruct(container_box_version) {
    unsigned int(8) num_circular_images;
    bit(19) reserved = 0;
    unsigned int(1) entrance_pupil_flag;
    unsigned int(1) flip_info_flag;
    unsigned int(1) camera_intrinsic_flag;
    unsigned int(1) local_fov_flag;
    unsigned int(1) deadzone_flag;
    for (i=0; i<num_circular_images; i++) {
       if (container_box_version > 0) {
           unsigned int(4) lens_projection_type;
           bit(4) reserved = 0;
       }
       if (entrance_pupil_flag == 1 ) {
           unsigned int(16) num_ep_coeffs;
           for (j=0; j<num_ep_coeffs; j++)
               signed int(32) ep_coeff;
       }
       if (flip_info_flag == 1) {
           bit(30) reserved = 0;
           unsigned int(2) image_flip;
       }
       if (camera_intrinsic_flag == 1) {
           unsigned int(32) image_scale_axis_angle;
           unsigned int(32) image_scale_x;
           unsigned int(32) image_scale_y;
           bit(16) reserved = 0;
           unsigned int(16) num_polynomial_coefs_lsc;
           for (j=0; j<num_polynomial_coefs_lsc; j++) {
               signed int (32) polynomial_coef_k_lsc_r;
               signed int (32) polynomial_coef_k_lsc_g;
               signed int (32) polynomial_coef_k_lsc_b;
           }
       }
       if (local_fov_flag == 1) {
           unsigned int (16) num_angle_for_displaying_fov;
           bit(16) reserved = 0;
           for (j=0; j<num_angle_for_displaying_fov; j++) {
               unsigned int(32) displayed_fov;
               unsigned int(32) overlapped_fov;
           }
           bit(16) reserved = 0;
           unsigned int (16) num_local_fov_region;
           for (j=0; j<num_local_fov_region; j++) {
               unsigned int(32) start_radius;
               unsigned int(32) end_radius;
               signed int(32) start_angle;
               signed int(32) end_angle;
               unsigned int(32) radius_delta;
               signed int(32) angle_delta;
               for (rad=start_radius; rad<=end_radius; rad+=radius_delta)
                   for (ang=start_angle; ang<= end_angle; ang+=angle_delta)
                      unsigned int(32) local_fov_weight;
           }
       }
    }
    if (deadzone_flag == 1) {
       bit(24) reserved = 0;
       unsigned int(8) num_deadzones;
       for (j=0; j<num_deadzones; j++) {
           unsigned int(16) deadzone_left_horizontal_offset;
           unsigned int(16) deadzone_top_vertical_offset;
           unsigned int(16) deadzone_width;
           unsigned int(16) deadzone_height;
       }
```

```
    }
}
```

6.6 Video and image metadata semantics

6.6.1 Projection format structure - Semantics

`projection_type` indicates the type of the mapping of the projected picture onto the spherical coordinate system as specified in 5.1. The values of `projection_type` and their semantics are specified in Table 6.1.

6.6.2 Region-wise packing structure

6.6.2.1 General

This subclause specifies information about region-wise packing structure.

6.6.2.2 Semantics of the rectangular region packing structure

`proj_reg_width[i]`, `proj_reg_height[i]`, `proj_reg_top[i]`, and `proj_reg_left[i]` specify the width, height, top offset, and left offset, respectively, of the i-th projected region, either within the projected picture (when `constituent_picture_matching_flag` is equal to 0) or within the constituent picture of the projected picture (when `constituent_picture_matching_flag` is equal to 1). `proj_reg_width[i]`, `proj_reg_height[i]`, `proj_reg_top[i]` and `proj_reg_left[i]` are indicated in relative projected picture sample units.

Two projected regions may partially or entirely overlap with each other. When there is an indication of quality difference, for example, by a region-wise quality ranking indication, then for the overlapping area of any two overlapping projected regions, the packed region corresponding to the projected region that is indicated to have higher quality should be used for rendering.

`transform_type[i]` specifies the rotation and mirroring that is applied to the i-th packed region to remap it to the i-th projected region. When `transform_type[i]` specifies both rotation and mirroring, rotation is applied before mirroring for converting sample locations of a packed region to sample locations of a projected region. The following values are specified:

0: no transform

1: mirroring horizontally

2: rotation by 180 degrees (counter-clockwise)

3: rotation by 180 degrees (counter-clockwise) before mirroring horizontally

4: rotation by 90 degrees (counter-clockwise) before mirroring horizontally

5: rotation by 90 degrees (counter-clockwise)

6: rotation by 270 degrees (counter-clockwise) before mirroring horizontally

7: rotation by 270 degrees (counter-clockwise)

NOTE 1 6.2.2 specifies the semantics of `transform_type[i]` for converting a sample location of a packed region in a packed picture to a sample location of a projected region in a projected picture.

`packed_reg_width[i]`, `packed_reg_height[i]`, `packed_reg_top[i]`, and `packed_reg_left[i]` specify the width, height, the offset, and the left offset, respectively, of the i-th packed region, either within the packed picture (when `constituent_picture_matching_flag` is equal to 0) or within each constituent picture of the packed picture (when `constituent_picture_matching_flag` is equal to 1). `packed_reg_width[i]`, `packed_reg_height[i]`, `packed_reg_top[i]`, and `packed_reg_left[i]` are indicated in relative packed picture sample units. `packed_reg_width[i]`, `packed_reg_height[i]`,

packed_reg_top[i], and packed_reg_left[i] shall represent integer horizontal and vertical coordinates of luma sample units within the decoded pictures.

NOTE 2 Two packed regions can partially or entirely overlap with each other.

6.6.2.3 Semantics of the guard band structure

left_gb_width[i] specifies the width of the guard band on the left side of the i-th packed region in relative packed picture sample units. When the decoded picture has 4:2:0 or 4:2:2 chroma format, left_gb_width[i] shall correspond to an even number of luma samples within the decoded picture.

right_gb_width[i] specifies the width of the guard band on the right side of the i-th packed region in relative packed picture sample units. When the decoded picture has 4:2:0 or 4:2:2 chroma format, right_gb_width[i] shall correspond to an even number of luma samples within the decoded picture.

top_gb_height[i] specifies the height of the guard band above the i-th packed region in relative packed picture sample units. When the decoded picture has 4:2:0 chroma format, top_gb_height[i] shall correspond to an even number of luma samples within the decoded picture.

bottom_gb_height[i] specifies the height of the guard band below the i-th packed region in relative packed picture sample units. When the decoded picture has 4:2:0 chroma format, bottom_gb_height[i] shall correspond to an even number of luma samples within the decoded picture.

When GuardBand(i) is present, at least one of left_gb_width[i], right_gb_width[i], top_gb_height[i], or bottom_gb_height[i] shall be greater than 0.

gb_not_used_for_pred_flag[i] equal to 0 specifies that the guard bands may or may not be used in the inter prediction process. gb_not_used_for_pred_flag[i] equal to 1 specifies that the sample values of the guard bands are not used in the inter prediction process.

NOTE 1 When gb_not_used_for_pred_flag[i] is equal to 1, the sample values within guard bands in decoded pictures can be rewritten even if the decoded pictures were used as references for inter prediction of subsequent pictures to be decoded. For example, the content of a packed region can be seamlessly expanded to its guard band with decoded and re-projected samples of another packed region.

gb_type[i][j] specifies the type of the guard bands for the i-th packed region as follows, with j equal to 0, 1, 2, or 3 indicating that the semantics below apply to the left, right, top, or bottom edge, respectively, of the packed region:

— gb_type[i][j] equal to 0 specifies that the content of the guard bands in relation to the content of the packed regions is unspecified. When gb_not_used_for_pred_flag[i] is equal to 0, gb_type[i][j] shall not be equal to 0.

— gb_type[i][j] equal to 1 specifies that the content of the guard bands suffices for interpolation of sub-pixel values within the packed region and less than one pixel outside of the boundary of the packed region.

NOTE 2 gb_type[i][j] equal to 1 can be used when the boundary samples of a packed region have been copied horizontally or vertically to the guard band.

— gb_type[i][j] equal to 2 specifies that the content of the guard bands represents actual picture content that is spherically adjacent to the content in the packed region and is on the surface of the packed region at quality that gradually changes from the picture quality of the packed region to that of the spherically adjacent packed region.

— gb_type[i][j] equal to 3 specifies that the content of the guard bands represents actual picture content that is spherically adjacent to the content in the packed region and is on the surface of the packed region at the picture quality of the packed region.

— gb_type[i][j] values greater than 3 are reserved.

6.6.2.4 Semantics of the region-wise packing structure

`constituent_picture_matching_flag` equal to 1 specifies that the projected region information, packed region information, and guard band region information in this syntax structure apply individually to each constituent picture and that the packed picture and the projected picture have the same stereoscopic frame packing format. `constituent_picture_matching_flag` equal to 0 specifies that the projected region information, packed region information, and guard band region information in this syntax structure apply to the projected picture. When SpatiallyPackedStereoFlag is equal to 0, `constituent_picture_matching_flag` shall be equal to 0.

NOTE 1 For the stereoscopic content that uses equivalent region-wise packing for the constituent pictures, setting this flag equal to 1 allows more compact signalling of region-wise packing information.

`num_regions` specifies the number of packed regions when `constituent_picture_matching_flag` is equal to 0. Value 0 is reserved. When `constituent_picture_matching_flag` is equal to 1, the total number of packed regions is equal to 2 * `num_regions` and the information in `RectRegionPacking(i)` and `GuardBand(i)` applies to each constituent picture of the projected picture and the packed picture.

`proj_picture_width` and `proj_picture_height` specify the width and height, respectively, of the projected picture, in relative projected picture sample units. `proj_picture_width` and `proj_picture_height` shall both be greater than 0.

NOTE 2 The same sampling grid, width, and height are used for the luma sample array and the chroma sample arrays of the projected picture.

`packed_picture_width` and `packed_picture_height` specify the width and height, respectively, of the packed picture, in relative packed picture sample units. `packed_picture_width` and `packed_picture_height` shall both be greater than 0.

`guard_band_flag[i]` equal to 0 specifies that the i-th packed region has no guard bands. `guard_band_flag[i]` equal to 1 specifies that the i-th packed region has at least one guard band.

`packing_type[i]` specifies the type of region-wise packing. The values of `packing_type[i]` and their semantics are specified in Table 6.2.

`RectRegionPacking(i)` specifies the region-wise packing between the i-th packed region and the i-th projected region. The syntax and semantics of `RectRegionPacking(i)` are specified in 6.5.2.2 and 6.6.2.2, respectively.

`GuardBand(i)` specifies the guard bands for the i-th packed region. The syntax and semantics of `GuardBand(i)` are specified in 6.5.2.3 and 6.6.2.3, respectively.

6.6.2.5 Derivation of region-wise packing variables and constraints for the syntax elements of the region-wise packing structure

When the i-th packed region as specified by this `RegionWisePackingStruct` overlaps with the j-th packed region specified by the same `RegionWisePackingStruct`, the i-th and j-th projected regions shall reside in different constituent pictures for any values of i and j that are not equal to each other. The i-th packed region as specified by this `RegionWisePackingStruct` shall not overlap with any guard band specified by the same `RegionWisePackingStruct`.

The guard bands associated with the i-th packed region, if any, as specified by this `RegionWisePackingStruct` shall not overlap with any packed region specified by the same `RegionWisePackingStruct` or any other guard bands specified by the same `RegionWisePackingStruct`.

Projected regions may overlap. When projected regions overlap and a quality difference is indicated between the projected regions, for example, by a region-wise quality ranking indication, the packed region that is indicated to have the highest quality among the packed regions corresponding to the projected regions that overlap should be used for rendering the overlapping area.

The variables NumRegions, PackedRegLeft[n], PackedRegTop[n], PackedRegWidth[n], PackedRegHeight[n], ProjRegLeft[n], ProjRegTop[n], ProjRegWidth[n], ProjRegHeight[n], TrasnformType[n], PackingType[n] are derived as follows:

- For n in the range of 0 to `num_regions` − 1, inclusive, the following applies:
  - PackedRegLeft[n] is set equal to `packed_reg_left[n]`.
  - PackedRegTop[n] is set equal to `packed_reg_top[n]`.
  - PackedRegWidth[n] is set equal to `packed_reg_width[n]`.
  - PackedRegHeight[n] is set equal to `packed_reg_height[n]`.
  - ProjRegLeft[n] is set equal to `proj_reg_left[n]`.
  - ProjRegTop[n] is set equal to `proj_reg_top[n]`.
  - ProjRegWidth[n] is set equal to `proj_reg_width[n]`.
  - ProjRegHeight[n] is set equal to `proj_reg_height[n]`.
  - TransformType[n] is set equal to `transform_type[n]`.
  - PackingType[n] is set equal to `packing_type[n]`.
- If `constituent_picture_matching_flag` is equal to 0, the following applies:
  - NumRegions is set equal to `num_regions`.
- Otherwise (`constituent_picture_matching_flag` is equal to 1), the following applies:
  - NumRegions is set equal to 2 * `num_regions`.
  - When TopBottomFlag is equal to 1, the following applies:
    - projLeftOffset and packedLeftOffset are both set equal to 0.
    - projTopOffset is set equal to `proj_picture_height` / 2 and packedTopOffset is set equal to `packed_picture_height` / 2.
  - When SideBySideFlag is equal to 1, the following applies:
    - projLeftOffset is set equal to `proj_picture_width` / 2 and packedLeftOffset is set equal to `packed_picture_width` / 2.
    - projTopOffset and packedTopOffset are both set equal to 0.
  - For n in the range of NumRegions / 2 to NumRegions − 1, inclusive, the following applies:
    - nIdx is set equal to n − NumRegions / 2.
    - PackedRegLeft[n] is set equal to `packed_reg_left[nIdx]` + packedLeftOffset.
    - PackedRegTop[n] is set equal to `packed_reg_top[nIdx]` + packedTopOffset.
    - PackedRegWidth[n] is set equal to `packed_reg_width[nIdx]`.
    - PackedRegHeight[n] is set equal to `packed_reg_height[nIdx]`.
    - ProjRegLeft[n] is set equal to `proj_reg_left[nIdx]` + projLeftOffset.
    - ProjRegTop[n] is set equal to `proj_reg_top[nIdx]` + projTopOffset.
    - ProjRegWidth[n] is set equal to `proj_reg_width[nIdx]`.