**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 11**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11/N16699**

**Geneva, CH – January 2017**

**Title: Algorithm descriptions of projection format conversion and video quality metrics in 360Lib**

**Source: JVET**

**Status: Approved**

|  |  |
| --- | --- |
| **Joint Video Exploration Team (JVET)**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11**  5th Meeting: Geneva, CH, 12–20 January 2017 | Document: JVET-E1003 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Algorithm descriptions of projection format conversion and video quality metrics in 360Lib** | | |
| *Status:* | Output Document of JVET | | |
| *Purpose:* | 360Lib algorithm description | | |
| *Author(s) or Contact(s):* | Yan Ye  InterDigital Communications  Elena Alshina  Samsung Electronics  Jill Boyce  Intel | Tel: Email: | [yan.ye@interdigital.com](mailto:yan.ye@interdigital.com)  [elena\_a.alshina@samsung.com](mailto:elena_a.alshina@samsung.com)  [jill.boyce@intel.com](mailto:jill.boyce@intel.com) |
| *Source:* | Editors | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

The Joint Video Exploration Team (JVET) of ITU-T VCEG (Q6/16) and ISO/IEC MPEG (JTC 1/SC 29/WG 11) is studying the potential need to include 360-degree video coding technologies in the future video coding standard. The JVET has established a 360Lib software package that can perform projection format conversion between various projection formats as a standalone conversion tool, or in combination with encoding and decoding using HM or JEM reference software. Among the projection formats 360Lib supports, viewport generation using rectilinear projection is also included. Further, 360Lib supports a number of objective 360-degree video quality metrics. This document describes the algorithms used for projection format conversion and video quality metrics in 360Lib.

[Abstract 1](#_Toc473143440)

[1 Introduction 2](#_Toc473143441)

[2 Conversion of 2D and 3D coordinates for each projection format 2](#_Toc473143442)

[2.1 Equirectangular projection format 4](#_Toc473143443)

[2.2 Cubemap projection format 4](#_Toc473143444)

[2.3 Equal-area projection format 5](#_Toc473143445)

[2.4 Octahedron projection format 5](#_Toc473143446)

[2.5 Icosahedron projection format 8](#_Toc473143447)

[2.6 Truncated Square Pyramid 9](#_Toc473143448)

[2.7 Segmented sphere projection format 11](#_Toc473143449)

[3 Conversion between two projection formats 12](#_Toc473143450)

[4 Spherical objective quality metrics 12](#_Toc473143451)

[4.1 WS-PSNR 13](#_Toc473143452)

[5 Viewport generation 13](#_Toc473143453)

[6 References 14](#_Toc473143454)

# Overview

The Joint Video Exploration Team (JVET) of ITU-T VCEG (Q6/16) and ISO/IEC MPEG (JTC 1/SC 29/WG 11) is working on 360-degree video coding as part of the explorations being conducted for developing coding technologies for future video coding standards. The JVET has established a 360Lib software package for 360-degree video coding and processing [1], and defined the common test condition (CTC) for 360-degree video coding [2]. This document describes the algorithms implemented in 360Lib for projection format conversion and 360-degree video quality evaluation.

The 360Lib software is written in C++, following similar programming styles used by the HM and JEM reference software. 360Lib can be used as standalone application to perform projection format conversion and quality metric calculations. It is also integrated with the latest versions of HM and JEM, and can be used in combination with HM or JEM to perform projection format conversion (before and/or after coding), compression, and quality metric calculation altogether, without the need to store intermediate YUV sequences at intermediate steps. Example configuration files according to the common test conditions for 360 video [1], and a software usage manual are included in the 360Lib package. 360Lib reference software was agreed to be established at the 5th JVET meeting in Oct 2016. The first version, 360Lib v1.0, was officially released in Dec 2016, and 360Lib v2.0 is on schedule to be released in Feb 2017.

The projection formats supported in 360Lib, as well as their corresponding indices, are listed in Table 1. 360Lib supports projection format conversion between any pair of these projection formats, except for indices 4 and 6, which are used for viewport generation and CPP-PSNR calculation, respectively. More details about each of the projection formats will be presented in Section 2, followed by the detailed discussion of the conversion process in Section 3.

**Table 1. Projection formats supported in 360Lib**

|  |  |
| --- | --- |
| **Index** | **Projection format** |
| 0 | Equirectangular [3](ERP) |
| 1 | Cubemap [4] (CMP) |
| 2 | Equal-area [5] (EAP) |
| 3 | Octahedron [6] (OHP) |
| 4 | Viewport generation using rectilinear projection [7] |
| 5 | Icosahedron [8] (ISP) |
| 6 | Crasters Parabolic Projection for CPP-PSNR calculation [9] |
| 7 | Truncated Square Pyramid [10] (TSP) |
| 8 | Segmented Sphere Projection [11] (SSP) |

Depending on the specific projection format used to represent the spherical video, different areas of the sphere are sampled at different densities on the 2D plane. For example, the commonly used ERP format oversamples the sphere at the poles, resulting in over-stretched top and bottom areas on the ERP picture. When using CMP format, spherical positions corresponding to the center of a CMP face are sampled more sparsely compared to those corresponding to the sides of the face. Because projecting a spherical 360-degree video onto 2D planes is a non-linear process, the conventional PSNR, which weighs the sample errors at each 2D position equally, is not a suitable quality metric for 360-degree video. 360Lib supports a number of 360-degree video quality metrics to address the uneven sampling problem in the 2D projected video. Further details about these metrics are provided in Section 4.

The rest of this document is organized as follows. Section 2 provides detailed description of each projection format supported in 360Lib. The viewport generation process using rectilinear projection is also discussed. Section 3 describes the conversion process between any two projection formats. Section 4 describes the objective quality metrics supported in 360Lib, as well as how objective quality metrics are measured in the 360 video CTC.

# Detailed descriptions of projection formats supported in 360Lib

The 3D XYZ coordinate system as shown in Figure 1 is used in 360Lib to describe the 3D geometry of each projection format representation. Starting from the center of the sphere, X axis points toward the front of the sphere, Y axis points toward the top of the sphere, and Z axis points toward the right of the sphere.



**Figure 1.** 3D XYZ coordinate definition used in 360Lib, A3 is the equator

The sphere can be sampled with longitude (ϕ) and latitude (θ). In aviation, the longitude ϕ in the range [-π, π] is known as yaw, and latitude θ in the range [-π/2, π/2] is known as pitch, where π is the ratio of a circle's circumference to its diameter. In 360Lib, the longitude ϕ is defined by the angle starting from X axis in counter-clockwise direction as shown in Figure 1. The latitude θ is defined by the angle from the equator toward Y axis as shown in Figure 1. The (X, Y, Z) coordinates on the unit sphere can be evaluated from (ϕ, θ) using (1) (2) (3).

|  |  |
| --- | --- |
| X = cos(θ)cos(ϕ) | (1) |
| Y = sin(θ) | (2) |
| Z = -cos(θ)sin(ϕ) | (3) |

Inversely, the longitude and latitude (ϕ, θ) can be evaluated from (X, Y, Z) coordinates using (4)(5).

|  |  |
| --- | --- |
| ϕ = tan-1(-Z/X) | (4) |
| θ = sin-1(Y/(X2+Y2+Z2)1/2) | (5) |

A 2D plane coordinate system is defined for each face in the 2D projection plane. Whereas some of the projection formats in 360Lib have only one face (such as ERP and EAP), other projection formats have multiple faces. In order to generalize the 2D coordinate system, a face index is defined for each face in the 2D projection plane. Each face is mapped to a 2D plane, referred as the uv plane, associated with one face index. The 2D image sampling grid is defined in the uv plane. We refer to the sampling point position as (m, n), where m and n are the column and row coordinates of the sampling position. Figure2 shows an example for the sampling coordinates defined in the uv plane for ERP projection. The orange circles are the sampling points (m, n). In 360Lib, in order to arrange all sampling points in a symmetric manner in both directions, there is a shift between the origin of (u, v) coordinates and the origin of (m, n) coordinates, as shown in Figure2.



**Figure 2.** Sampling coordinate definition in (u, v) plane in 360Lib

Finally, we assume W and H are the width and height of a face, respectively. With these notations and coordinate systems, this section describes the conversion between sampling point (f, m, n) (f is the face index) and 3D point position (X, Y, Z) for each projection format supported in 360Lib. In the 360Lib software package, the conversion from (f, m, n) to (X, Y, Z) is implemented by the function map2Dto3D(), and the conversion from (X, Y, Z) to (f, m, n) is implemented by the function map3Dto2D().

## Equi-rectangular projection format (ERP)

The ERP projection format is the most widely used projection format for representing 360-degree video on a 2D plane. It is the default projection format in 360Lib.

ERP has only one face. Therefore, the face index f for ERP is always set to 0. In the uv plane, u and v are in the range [0, 1].

For 2D-to-3D coordinate conversion, we first start from a given sampling position (m, n), and calculate (u, v) using (6)(7).

|  |  |
| --- | --- |
| u = (m + 0.5)/W, 0≤ m <W | (6) |
| v = (n + 0.5)/H, 0 ≤ n < H | (7) |

Then, the longitude and latitude (ϕ, θ) in the sphere can be calculated from (u, v) using (8)(9):

|  |  |
| --- | --- |
| ϕ = (u - 0.5)\*(2\* π) | (8) |
| θ = (0.5 - v)\*π | (9) |

Finally, (X, Y, Z) can be calculated from Equation (1)(2)(3).

For 3D-to-2D coordinate conversion starting from (X, Y, Z), (ϕ, θ) is first calculated using (4)(5). Then, (u, v) is calculated by solving Equations (8)(9). Finally, (m, n) is calculated by solving Equation (6)(7).

## Cubemap projection format (CMP)

The CMP projection has 6 square faces in total, labelled as PX, PY, PZ, NX, NY, NZ (with “P” standing for “positive” and “N” standing for “negative”), in Figure 3. Table 2 specifies the face index values corresponding to each of the six CMP faces in 360Lib.

The uv plane definition for each CMP face in 360Lib is also shown in Figure 3. Each face in the uv plane is a 2x2 square, with u and v being defined in the range of [-1, 1].



**Figure 3. Coordinates definition for CMP**

**Table 2. Face index of CMP**

|  |  |  |
| --- | --- | --- |
| **Face index** | **Face label** | **Notes** |
| 0 | PX | Front face with positive X axis value |
| 1 | NX | Back face with negative X axis value |
| 2 | PY | Top face with positive Y axis value |
| 3 | NY | Bottom face with negative Y axis value |
| 4 | PZ | Right face with positive Z axis value |
| 5 | NZ | Left face with negative Z axis value |

For 2D-to-3D coordinate conversion, given the position (m, n) on a given face f, (u, v) is first calculated as:

|  |  |
| --- | --- |
| u = (m + 0.5)\*2/W - 1, 0≤ m <W | (10) |
| v = (n + 0.5)\*2/H - 1, 0 ≤ n < H | (11) |

Then, the 3D coordinates (X, Y, Z) are derived using Table 3 given the position (u, v) and the face index f.

**Table 3. (X, Y, Z) derivation given (u, v) and the face index f**

|  |  |  |  |
| --- | --- | --- | --- |
| **f** | **X** | **Y** | **Z** |
| 0 | 1.0 | -v | -u |
| 1 | -1.0 | -v | u |
| 2 | u | 1.0 | v |
| 3 | u | -1.0 | -v |
| 4 | u | -v | 1.0 |
| 5 | -u | -v | -1.0 |

For 3D-to-2D coordinate conversion, given (X, Y, Z), the (u, v) and face index f is calculated according to Table 4. Then, (m, n) on the face f is calculated by solving the Equation (10)(11).

**Table 4. Derivation of (u, v) and the face index f given (X, Y, Z)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Condition** | **f** | **u** | **v** |
| |X| ≥ |Y| and |X| ≥ |Z| and X >0 | 0 | -Z/|X| | -Y/|X| |
| |X| ≥ |Y| and |X| ≥ |Z| and X <0 | 1 | Z/|X| | -Y/|X| |
| |Y| ≥ |X| and |Y| ≥ |Z| and Y >0 | 2 | X/|Y| | Z/|Y| |
| |Y| ≥ |X| and |Y| ≥ |Z| and Y <0 | 3 | X/|Y| | -Z/|Y| |
| |Z| ≥ |X| and |Z| ≥ |X| and Z >0 | 4 | X/|Z| | -Y/|Z| |
| |Z| ≥ |X| and |Z| ≥ |Y| and Z <0 | 5 | -X/|Z| | -Y/|Z| |

For projection formats that have multiple faces, there are different ways to arrange the faces onto one 2D picture. 360Lib allows the faces to be arranged in a flexible manner. Parameters are provided in the configuration files to allow the user to specify whether to rotate a given face by 0, 90, 180 or 270 degrees, and where to place that given face on the 2D picture. For further details on how to use the configuration parameters, readers are referred to the user guide in 360Lib at [1].

## Equal-area projection format (EAP)

Same as ERP, there is only one face for EAP, and the face index for EAP is always set to 0. In the uv plane, u and v are in the range [0, 1].

For 2D-to-3D coordinate conversion, given the sampling position (m, n), we can first calculate (u, v) with Equations (6)(7). Then, the longitude and latitude (ϕ, θ) on the sphere can be calculated from (u, v) as:

|  |  |
| --- | --- |
| ϕ = (u - 0.5)\*(2\* π) | (12) |
| θ = sin-1(1.0 – 2\*v) | (13) |

Finally, (X, Y, Z) can be calculated using Equations (1)(2)(3).

For 3D-to-2D coordinate conversion, given (X, Y, Z), first (ϕ, θ) is calculated using Equations (4)(5). Then, (u, v) is calculated by solving Equations (12)(13). Finally, (m, n) is calculated by solving Equations (6)(7).

## Octahedron projection format (OHP)

The OHP projection format has 8 triangle faces and 6 vertices, labelled as {F0, F1, F2, F3, F4, F5, F6, F7} and {V0, V1, V2, V3, V4, V5}, respectively, as shown in Figure 4. The XYZ coordinates of the 6 vertices are defined in Table 5 for both non-compact and compact OHP.



**Figure 4. Definitions of faces and vertices for OHP in 360Lib**

**Table 5. Definition of the XYZ coordinates of OHP’s vertices**

|  |  |  |
| --- | --- | --- |
| **Vertex** | **(X, Y, Z) definition in non-compact OHP** | **(X, Y, Z) definition in compact OHP** |
| V0 | (0, 20.5, 0) | (0, 0, 20.5) |
| V1 | (1, 0, 1) | (1, -1, 0) |
| V2 | (1, 0, -1) | (1, 1, 0) |
| V3 | (0, -20.5, 0) | (0, 0, -20.5) |
| V4 | (-1, 0, -1) | (-1, 1, 0) |
| V5 | (-1, 0, 1) | (-1, -1, 0) |

Unlike the CMP projection, the OHP projection (and ISP in the next section) contains triangle faces. The triangle faces need to be packed carefully, in order to minimize discontinuity between neighboring faces and to improve coding efficiency. Further, for better coding efficiency, it is also desirable to minimize the number of inactive samples in the packed 2D picture. “Inactive samples” are defined as samples on the 2D picture that do not correspond to any sample positions on the sphere; they are filled with default gray color in 360Lib. 360Lib supports two type of packing for OHP: the non-compact OHP packing and the compact OHP packing [12]. Table 6 shows how the 6 vertices are used to define the 8 triangle faces in the non-compact OHP and in the compact OHP in 360Lib.

**Table 6. Definition of the vertices of each of the triangle faces for non-compact OHP and compact OHP in 360Lib**

|  |  |  |
| --- | --- | --- |
| **Face index** | **Vertex definition in non-compact OHP** | **Vertex definition in compact OHP** |
| 0 | {V0, V1, V2} | {V0, V1, V2} |
| 1 | {V3, V2, V1} | {V3, V2, V1} |
| 2 | {V0, V4, V5} | {V0, V4, V5} |
| 3 | {V3, V5, V4} | {V3, V5, V4} |
| 4 | {V0, V5, V1} | {V1, V0, V5} |
| 5 | {V3, V1, V5} | {V1, V5, V3} |
| 6 | {V0, V2, V4} | {V2, V4, V0} |
| 7 | {V3, V4, V2} | {V2, V3, V4} |



**Figure 5. Projection of a point to one face of octahedron**

Before we define the 2D-to-3D and 3D-to-2D coordinate mapping processes, we define a few notations in the 3D geometry. Figure 5 shows a side view of a triangle face in OHP in 3D space. The top-left point of the rectangle containing the triangle is denoted as the point R. Denote the XYZ coordinates of R as (XR, YR, ZR), and denote R’s position in vector form as . Denote the vertices of the triangle face as , , and (the positions of , , and on a given face are provided in Table 6). is derived according to Equation (14).

|  |  |
| --- | --- |
|  | (14) |

Two base vectors , are defined for the two orthogonal directions on the face, and are derived according to Equations (15)(16).

|  |  |
| --- | --- |
|  | (15) |
|  | (16) |

The norm vector of the face is derived as:

|  |  |
| --- | --- |
|  | (17) |

With these notations, the following steps are applied in 2D-to-3D coordinate mapping. First, given the sampling positon (m, n), (u, v) is calculated as:

|  |  |
| --- | --- |
| u = (m+0.5)\*2/W | (18) |
| v = (n+0.5)\*/H | (19) |

Then, the 3D coordinates (X, Y, Z) are calculated with Equations (20) (21) (22), where ()X, ()Y and ()Z denote the X, Y, Z coordinates of vector , respectively.

|  |  |
| --- | --- |
| X = XR + (u\*)X +(v\*)X | (20) |
| Y = YR + (u\*)Y +(v\*)Y | (21) |
| Z = ZR + (u\*)Z +(v\*)Z | (22) |

For 3D-to-2D coordinate mapping, the following is applied. Denote a point P with coordinates (X, Y, Z) in 3D space as in the vector form, we first determine the face index to which it belongs using Equation (23).

|  |  |
| --- | --- |
|  | (23) |

Then, the projection point Q on the face f is calculated as follows, where is the norm vector of the face f:

|  |  |
| --- | --- |
|  | (24) |

Then, the 2D point (u, v) is calculated as follows using the base vectors , of the face f (face index f is ignored in the base vector notations):

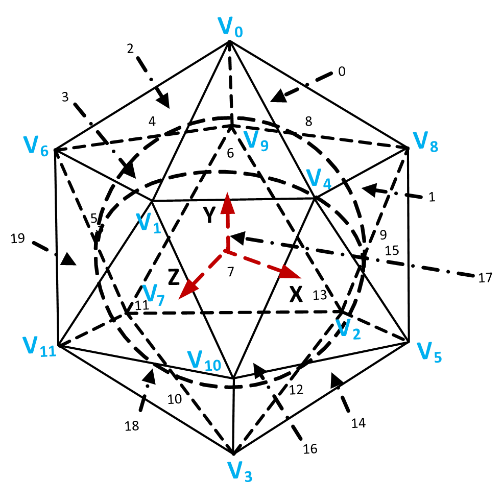
|  |  |
| --- | --- |
| u = | (25) |
| v = | (26) |

Finally, the position (m, n) in the sampling grid is calculated as:

|  |  |
| --- | --- |
| m = u\*(W/2) -0.5 | (27) |
| n = v\*(H/) -0.5 | (28) |

## Icosahedron projection format (ISP)

The ISP projection format has 12 vertices and 20 faces. Each face is a triangle in the same shape as the face of an octahedron. 360Lib also supports two types of packing for ISP: the non-compact packing and the compact packing. Figure 6 shows the definitions of those vertices and faces for non-compact ISP (left) and compact ISP (right). Table 7 defines the XYZ coordinates of those vertices, and Table 8 defines the vertices of each of the 20 faces in 360Lib, for both non-compact and compact ISP.



**Figure 6. Definition of ISP vertices in 360Lib. Left: vertices used in non-compact ISP; right: vertices used in compact ISP**

**Table 7. Coordinates definition of ISP vertices,**

|  |  |  |
| --- | --- | --- |
| **Vertex** | **Vertices for non-compact ISP** | **Vertices for compact ISP** |
| V0 | (1, c, 0) | (0,1.902, 0) |
| V1 | (-1, c, 0) | (0.526,0.851, 1.618) |
| V2 | (1, -c, 0} | (-0.526, -0.851, -1.618) |
| V3 | (-1, -c, 0) | (0, -1.902, 0) |
| V4 | (0, 1, c) | (1.701, 0.851, 0) |
| V5 | (0, -1, c) | (1.376, -0.851, -1) |
| V6 | (0, 1, -c) | (-1.376, 0.851, 1) |
| V7 | (0, -1, -c) | (-1.701, -0.851, 0) |
| V8 | (c, 0, 1} | (0.526, 0.851, -1.618) |
| V9 | (c, 0, -1) | (-1.376, 0.851, -1) |
| V10 | (-c, 0, 1) | (1.376, -0.851, 1) |
| V11 | (-c, 0, -1) | (-0.526, 0.851, 1.618) |

**Table 8. Definition of triangle faces of icosahedron**

|  |  |  |
| --- | --- | --- |
| **Face index** | **F for non-compact ISP** | **F for compact ISP** |
| 0 | {V0, V8, V9} | {V8, V9, V0} |
| 1 | {V2, V9, V8} | {V2, V9, V8} |
| 2 | {V0, V9, V6} | {V0, V9, V6} |
| 3 | {V7, V6, V9} | {V7, V6, V9} |
| 4 | {V0, V6, V1} | {V6, V1, V0} |
| 5 | {V11, V1, V6} | {V11, V1, V6} |
| 6 | {V0, V1, V4} | {V4, V0, V1} |
| 7 | {V10, V4, V1} | {V10, V4, V1} |
| 8 | {V0, V4, V8} | {V0, V4, V8} |
| 9 | {V5, V8, V4} | {V5, V8, V4} |
| 10 | {V3, V10, V11} | {V3, V10, V11} |
| 11 | {V1, V11, V10} | {V1, V11, V10} |
| 12 | {V3, V5, V10} | {V10, V3, V5} |
| 13 | {V4, V10, V5} | {V4, V10, V5} |
| 14 | {V3, V2, V5} | {V2, V5, V3} |
| 15 | {V8, V5, V2} | {V8, V5, V2} |
| 16 | {V3, V7, V2} | {V3, V7, V2} |
| 17 | {V9, V2, V7} | {V9, V2, V7} |
| 18 | {V3, V11, V7} | {V11, V7, V3} |
| 19 | {V6, V7, V11} | {V6, V7, V11} |

For non-compact ISP, the same geometry conversion functions used for OHP are shared by ISP. Non-compact ISP simply replaces the definition for vertex coordinates with those defined in Table 8. For non-compact ISP, the 2D-to-3D and 3D-to-2D coordinate mapping processes are exactly the same as those already described for OHP in section 2.4.

For compact ISP (CISP), all 20 triangle faces are compacted to the rectangular frame as shown on Figure 7. While compacting rectangular frame some triangles are split into the 2 parts vertically, some are flipped vertically or horizontally. 4 samples margin horizontal per boundary between 2 triangles are introduced in order to ensure always usage of proper chroma sample for each luma sample in non 4:4:4 formats and keeping resulting frame size multiple of 8. Those extra samples are not used in CISP to sphere (or viewport) projection, so they are just padded using nearest sample from triangle face.

If two triangle faces are next to each other both on icosahedron and on rectangular coding projection then there is no discontinuity. But if two triangle faces are not neighboring on icosahedron but put next to each other on rectangular frame then margin is introduced in order to resolve discontinuity, simplify encoding and reduce artifacts which might appear on triangles boundaries during encoding. Again extra samples from this margin are not used in CISP to sphere (or viewport) projection; so they are just filled using bilinear combination of nearest available samples from opposite edges of adjacent faces (or just copied if only one side neighbor is available). Size of the margin resolving discontinuity is 64 samples in horizontal and 32 samples in vertical directions correspondently.

As shown on Figure 7 (c), CISP has three continuous sets of polygons representing different parts of an icosahedron. Their triangle indexes are:

Set A: 2; 4-1; 6-2; 8; 13-2; 9; 15; 1; 17; 3-2.

Set B: 3-1; 19; 5; 11; 7; 13-1; 18; 10; 12; 14; 16.

Set C: 6-1; 4-2.

There is a vertical discontinuity between faces: 19 and 2; 18 and 0; 16 and 1; 13-1 and 6-1

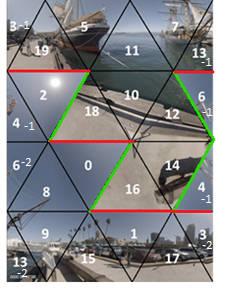
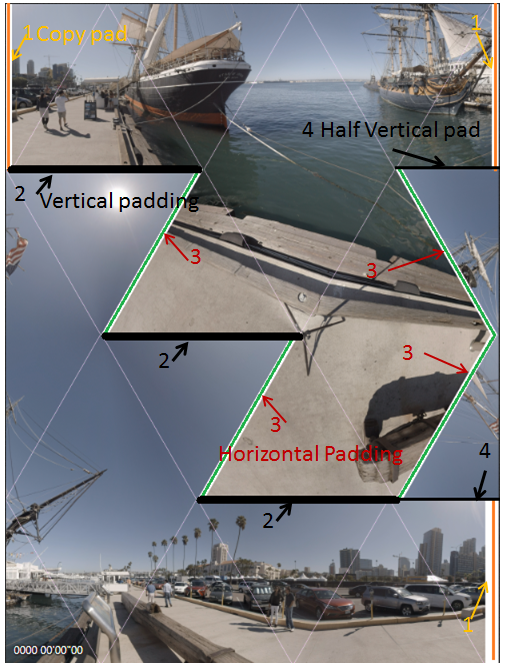
There is a horizontal discontinuity between faces: 1 and 18; 0 and 16; 14 and 4-2; 12 and 6-1

Displacement for set of triangle faces are summarized in Table 9.

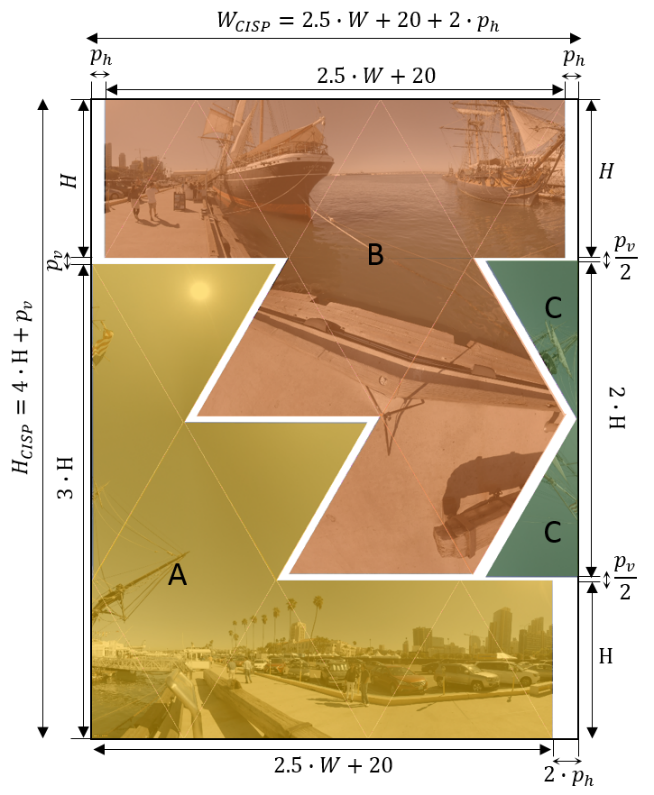
**Table 9 Displacement for faces set in CISP**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Set A | Set B | Set C |
| Vertical Shift | 32 | 0 | 16 |
| Horizontal Shift | 0 | 64 | 128 |

Figure 7.b specifies methods of padding on each boundary:  
 1. 64 pixels in top row and 128 pixels in bottom row; copy padding method is used;  
 2. 32 pixels are padded with bilinear filter;  
 3. 64 pixels are padded with bilinear filter;  
 4. 16 pixels are padded with bilinear filter.

a) b)



c)

**Figure 7. Compact icosahedral projection layout: a – face indexes; b – padding description; c – face sets and shift description.**

The size of CISP rectangular frame WCISP×HCISP is calculated based on width (W) and height (H) of rectangular faces and horizontal (and vertical (padding sizes as follows:

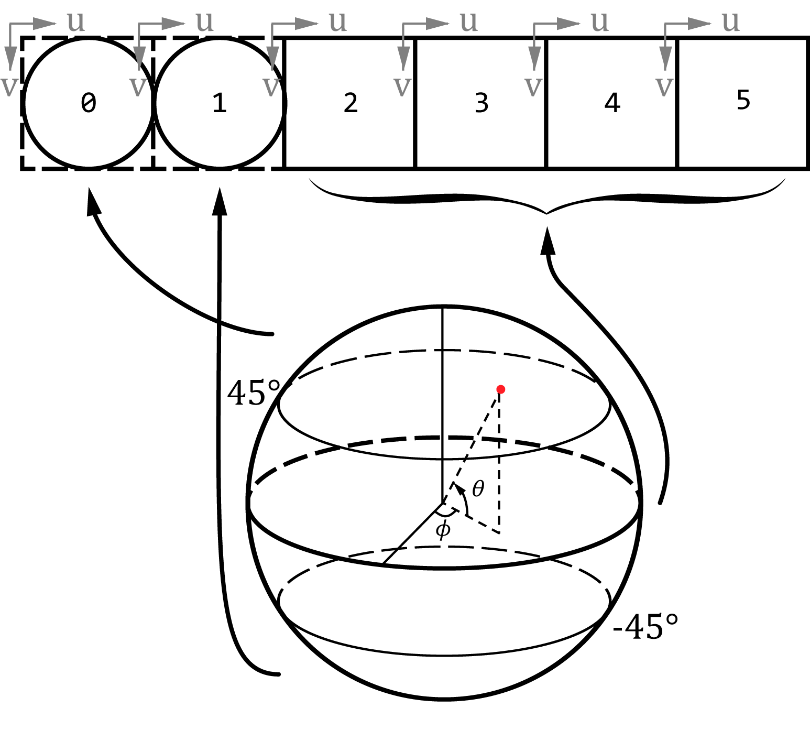
For W calculation there is multiple 5 because CISP has 5 triangles boundaries in each horizontal line. For H calculation there is multiple 4 because CISP has 4 triangles boundaries in each vertical line.

## Segmented sphere projection format (SSP)

SSP segments the sphere into 3 segments: the north pole, the equator, and the south pole. The boundaries of the 3 segments are at 45°N and 45°S. The north and south poles are mapped into 2 circles, and the projection of the equatorial segment is the same as ERP. The diameter of the circle is equal to the height of the equatorial segments because both pole segments and equatorial segment have a 90° latitude span.

The equatorial segment is split into 4 squares in order to get “faces” of same size. The frame packing structure is depicted in Figure 8[Ed. (YY): to be updated to vertical arrangement]. The corners of poles are inactive samples and filled with the default gray color. Points on the sphere are mapped as the following Equation (33)-(38):

|  |  |  |  |
| --- | --- | --- | --- |
|  | | (29) | |
|  | (30) | |
|  | (31) | |
|  | (32) | |
|  | (33) | |
|  | (34) | |



**Figure 8. Segmented sphere projection faces specification.**

## Truncated Square Pyramid (TSP)

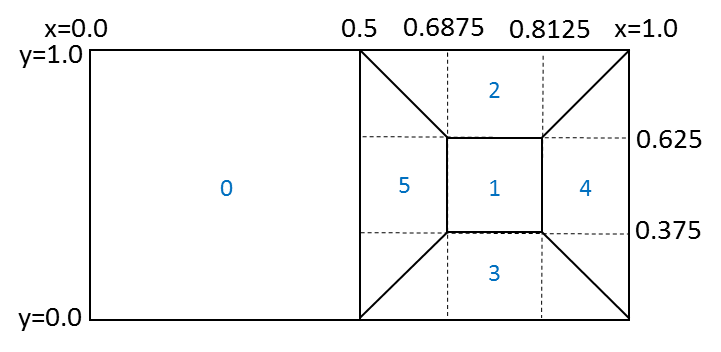
360Lib includes the truncated square pyramid (TSP) projection, which utilizes the cube geometry and warps the six cube faces into a compact frame. Figure 9 defines the (x, y) coordinates inside the packed TSP frame. (x, y) have normalized ranges (0.0, 1.0). Given the sampling position (m, n) on cube face f, (x, y) are calculated with following equations:

|  |  |
| --- | --- |
| x = 0.5\*(m + 0.5)/W + 0.5, 0≤ m <W | (35) |
| y = (n + 0.5)/H, 0 ≤ n < H | (36) |

In this implementation, the front TSP face corresponds with the front cube face, the back cube face is subsampled by 4 horizontally and vertically, while the side cube faces are warped to trapezoidal regions. Table 10 specifies the forward and inverse mapping equations. The coordinates (u, v) can be obtained from (x’, y’) as follows:

|  |  |
| --- | --- |
| u = 2x’ – 1.0 | (37) |
| v = 2y’ – 1.0 | (38) |

where (x’, y’) is defined in Table 10, and (x’, y’) are in the range (0.0, 1.0). Subsequently, (X, Y, Z) are derived with Table 3 given the position (u, v) on cube face f.



**Figure 9. Truncated-square-pyramid frame packing format**

**Table 10 Forward and inverse equations between cube faces and TSP faces.**

|  |  |
| --- | --- |
| **Forward equations (TSP to cube faces)** | **Inverse equations (cube faces to TSP)** |
| Right TSP trapezoid from right cube face:  x’ = (x – 0.5) / 0.1875  y’ = (y – 2.0x + 1.0) / (3.0 - 4.0x) | Right cube face from right TSP trapezoid:  x = 0.1875x’ + 0.5  y = 0.375x’ – 0.75x’y’ + y’ |
| Left TSP trapezoid from left cube face:  x' = (x – 0.8125) / 0.1875  y' = (y + 2.0x – 2.0) / (4.0x – 3.0) | Left cube face from left TSP trapezoid:  x = 0.1875x’ + 0.8125  y = 0.25y’ + 0.75x’y’ – 0.375x’ + 0.375 |
| Bottom TSP trapezoid from bottom cube face:  x' = (1.0 – x – 0.5y) / (0.5 – y)  y’ = (0.375 – y) / 0.375 | Bottom cube face from bottom TSP trapezoid:  x = 0.1875y’ – 0.375x’y’ – 0.125x’ + 0.8125  y = 0.375 – 0.375y’ |
| Top TSP trapezoid from top cube face:  x' = (0.5 – x + 0.5y) / (y – 0.5)  y’ = (1.0 – y) / 0.375 | Top cube face from top TSP trapezoid:  x = 1.0 – 0.1875y’ – 0.5x’ + 0.375x’y’  y = 1.0 – 0.375y’ |
| Back TSP face from back cube face:  x' = (x – 0.6875) / 0.125  y’ = (y – 0.375) / 0.25 | Back cube face from back TSP face:  x = 0.125x’ + 0.6875  y = 0.25y’ + 0.375 |

## Viewport generation with rectilinear projection

In 360Lib, a viewport is generated by rectilinear projection, as illustrated in Figure 10. Viewport generation in 360Lib is performed assuming that the viewing angle is along the Z axis. When the viewport specified by the user is not along the Z axis, the sphere is rotated first to align the viewport with the Z axis, and then viewport generation is performed. Denote the center of the viewport to be at (ϕ, θ), the rotation matrix R is defined as:

|  |  |
| --- | --- |
|  | (39) |



**Figure 10. Viewport generation with rectilinear projection**

Viewport generation starts from a sample position on the projected viewport, first finds the corresponding 3D (X, Y, Z) coordinates, then finds the corresponding 2D coordinates in the source projection plane, and finally takes the corresponding sample value at the corresponding position on the source 2D projection plane. In this section, we only describe the first step, that is, how to map a sample position to the 3D (X, Y, Z) coordinates based on rectilinear projection. The remaining steps are the same as the projection format conversion process to be described in the Section 3.

Denote the viewport picture ABCD’s width as WVP and its height as HVP. Denote the size of field of view (FOV) of the viewport as (Fh x Fv), where Fh is the horizontal FOV angle and Fv is the vertical FOV angle. Given a sampling point (m, n) in the viewport picture ABCD shown in Figure 9, the (u, v) coordinates are calculated as:

|  |  |
| --- | --- |
| u = (m+0.5)\*2\*tan(Fh/2)/WVP | (40) |
| v = (n+0.5)\*2\*tan(Fv/2)/HVP | (41) |

Then, the 3D coordinates (x, y, z) are calculated as:

|  |  |  |
| --- | --- | --- |
| x = u - tan(Fh/2) |  | (42) |
| y = -v + tan(Fv/2) |  | (43) |
| z = 1.0 |  | (44) |

Projecting the point (x, y, z) onto the point (x', y', z') on the unit sphere, we have:

|  |  |
| --- | --- |
| x' = x/ | (45) |
| y' = y/ | (46) |
| z' = 1.0 / | (47) |

Finally, taking the rotation R into account, the 3D coordinates (X, Y, Z) on the sphere are calculated as:

# Conversion between two projection formats

In 360Lib, the picture of a given projection format is stored face by face. The conversion from the source projection format to the destination projection format is also performed face by face.

Denote (fd, md, nd) as a point (md, nd) on face fd in the destination projection format, and (fs, ms, ns) as a point (ms, ns) on face fs in the source projection format. Denote (X, Y, Z) as the corresponding coordinates in the 3D XYZ space. The conversion process starts from each sample position (fd, md, nd) on the destination projection plane, maps it to the corresponding (X, Y, Z) in 3D coordinate system, finds the corresponding sample position (fs, ms, ns) on the source projection plane, and sets the sample value at (fd, md, nd) based on the sample value at (fs, ms, ns). Additionally, instead of mandating the XYZ axes of the source and destination projections to be aligned, 360Lib allows the user to use a set of 3D rotation parameters (rx, ry, rz) in the configuration file to specify the relative rotation between the source and destination 3D coordinates.

The projection format conversion process from source format to destination format is performed in the following steps:

1. Map the destination 2D sampling point (fd, md, nd) to 3D space coordinates (X, Y, Z) based on the destination projection format;
2. If needed, rotate the 3D point (X, Y, Z) along the three axes according to the angles given three rotation angles rx, ry, rz along X, Y, Z axes to (X’, Y’, Z’) using the rotation matrix RXYZ defined in Equation (48):

|  |
| --- |
| (48) |

1. Map (X’, Y’, Z’) to 2D sampling point (fs, ms, ns) based to the source projection format;
2. Calculate the sample value at (fs, ms, ns) by interpolating from neighboring samples at integer positions on face fs, and the interpolated sample value is placed at (fd, md, nd) in the destination projection format.

The above steps are repeated until all sample positions (fd, md, nd) in the destination projection format are filled. In 360Lib software, (Step 1) is performed using the function map2Dto3D() defined in the class of destination projection geometry, and (Step 3) is performed using function map3Dto2D() defined in the class of source projection geometry. (Step 2) may be skipped, if the user has not specified non-zero (rx, ry, rz) parameters. Note that (Step 1), (Step 2) and (Step 3) can be pre-calculated at the sequence level and stored as a lookup table, and only (Step 4) needs to be performed per sample position for each picture in order to render the sample values.

## Interpolation filters

When the source sample position (fs, ms, ns) is at factional sample position, interpolation needs to be performed to calculate the sample values at (fs, ms, ns) from the neighboring sample values at integer positions. The set of interpolation filters supported by 360Lib is listed in Table 11. All of the interpolation filters are specified at 1/100-th sample precision in 360Lib.

For projection format conversion, the user can specify interpolation filters for the luma and chroma components separately. By default, Lanczos-3 is used for luma and Lanczos-2 is used for chorma in projection format conversion. For viewport generation, bilinear filters are used. When 360Lib is used in combination with the HM or JEM codec, only bilinear filters are supported for viewport generation. When 360Lib is used as a stand-alone application, the filters used for viewport generation are configurable.

**Table 11. Interpolation methods supported**

|  |  |  |
| --- | --- | --- |
| **Index** | **Interpolation method** | **Notes** |
| 0 | Reserved |  |
| 1 | Nearest neighbor |  |
| 2 | Bilinear | Used for viewport generation \* |
| 3 | Bicubic |  |
| 4 | Lanczos2 | Default filter for the chroma components |
| 5 | Lanczos3 | Default filter for the luma component |

Note - Viewport generation inside encoder supports only bilinear filters, whereas viewport generation in standalone 360Lib application supports all filters listed.

## Chroma format support

360Lib allows the user to specify an internal chroma format, in which the projection format conversion process is performed. The internal chroma format can be different from the input and/or output chroma formats. For example, if the input video is in 4:2:0 chroma format, the user may specify the internal chroma format to be 4:2:0 or 4:4:4. When the internal chroma format is not the same as the input chroma format, chroma upsampling or downsampling is performed on the input video before projection format conversion is performed. When the internal chroma format is not the same as the output chroma format, chroma upsampling or downsampling is performed after projection format conversion. The HEVC motion compensation filters are used to perform chroma upsampling or downsampling.

When the internal chroma format is in 4:4:4, the luma and chroma components have the same sampling grid. Therefore, the projection format conversion process in Section 3 can be directly applied to luma and chroma, as the luma and chroma coordinates will remain the same before and after projection conversion, maintaining the 4:4:4 chroma format without any issue.

However, when the internal chroma format is not 4:4:4, the luma and the chroma sampling grids are not alway exactly aligned. For example, it is well known that in chroma format 4:2:0, by default the chroma sample location is shifted by a ½ luma sample compared to the corresponding luma location in the vertical direction. Because the projection format conversion process is non-linear, care needs to be given in order to ensure that luma and chroma sampling grids are properly aligned after projection format conversion. In other words, for default 4:2:0 chroma location type, the ½ luma sample shift in the vertical direction needs to be maintained after projection format conversion. 360Lib uses a simple method to deal with this issue: because the luma and chroma sampling grids are aligned (except for a factor of 2 in terms of resolution) in the chroma sample location type 2, 360Lib uses the type-2 chroma sample location when internal chroma is set to 4:2:0. Then, when performing projection format conversion for the chroma components, the chroma sampling grid is first scaled up by a factor of 2 to align with the luma sampling grid before conversion, and is scaled down by a factor of 2 after conversion.

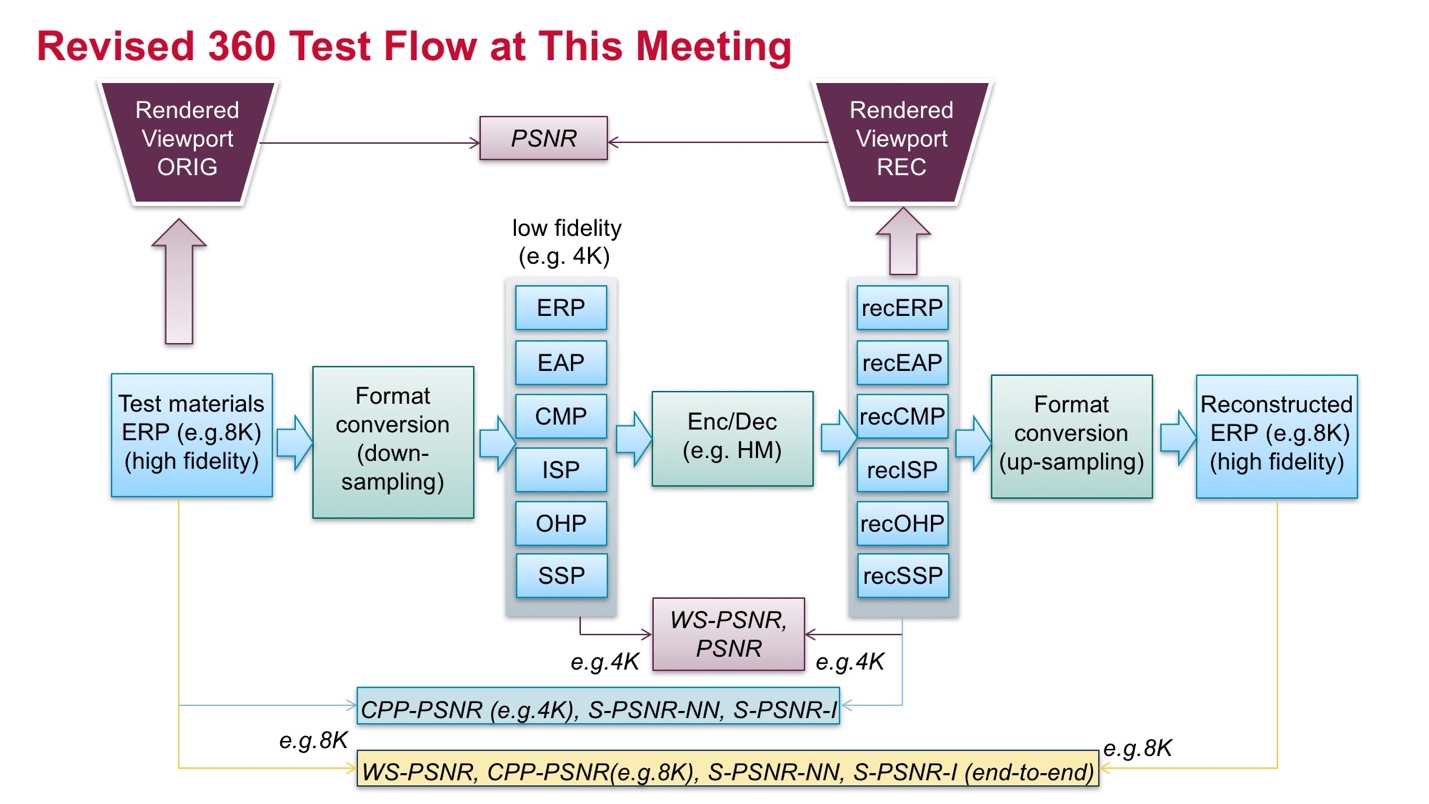
# Spherical objective quality metrics

Four spherical quality metrics are implemented in 360Lib for 360 video quality evaluation: weighted to spherically uniform PSNR (WS-PSNR) [13], spherical PSNR without interpolation (S-PSNR-NN), spherical PSNR with interpolation (S-PSNR-I), PSNR in Crasters Parabolic Projection format (CPP-PSNR). S-PSNR was originally proposed in [9]. In order to evaluate viewport quality, viewport based PSNR is also supported in 360Lib.

**Table 12. Quality metrics supported in 360Lib**

|  |  |
| --- | --- |
| **Metric name** | **Notes** |
| PSNR | Conventional PSNR calculation with equal weight for all samples |
| Weighted to Spherically uniform PSNR (WS-PSNR) | The distortion at each sample position is weighted by the area on the sphere covered by the given sample position. All samples on the 2D projection plane are used in WS-PSNR calculation. The two inputs to the metric calculation must have the same resolution and projection format. |
| Spherical PSNR w/o interpolation  (S-PSNR-NN) | Calculate PSNR based on a set of points uniformly sampled on the sphere. To find the sample value at the corresponding position on the projection plane, nearest neighbor rounding is applied. The two inputs to the metric calculation can have different resolution and/or projection format. |
| Spherical PSNR with interpolation  (S-PSNR-I) | Calculate PSNR based on a set of points uniformly sampled on the sphere. To find the sample value at the corresponding position on the projection plane, bicubic interpolation is applied. The two inputs to the metric calculation can have different resolution and/or projection format. |
| CPP-PSNR | Apply another projection format conversion to convert the two inputs into the Crasters Parabolic Projection (CPP) domain, and calculate PSNR in CPP domain. The two inputs to the metric calculation can have different resolution and/or projection format. |

The JVET has established common test conditions for 360-degree video [2]. Figure 11 depicts the processing and coding pipeline. For a given input 360-degree video, projection format conversion is applied first to convert source projection format into coding projection format. In the CTC for 360-degree video, the coding projection format is in lower resolution than that of the source projection format. For example, 8K source video is coded in 4K resolution. After coding, the reconstructed signal in coding projection format is back converted to the source projection format at the source resolution. In the CTC, original video sequences are all provided in the ERP format in either 8K or 4K.



**Figure 11. 360 video testing procedure**

Calculation of 360-video objective quality metrics is performed at different stages in the CTC coding pipeline, summarized as the following 3 categories:

1. End-to-end distortion measurement: WS-PSNR, CPP-PSNR, S-PSNR-I and S-PSNR-NN are calculated between the original signal in source projection format and the reconstructed signal in source projection format. The end-to-end distortion considers both projection format conversion errors (including forward and backward projection format conversion) and coding errors;
2. Cross-format distortion measurement: CPP-PSNR, S-PSNR-I and S-PSNR-NN are measured between the original signal in source projection format and the reconstructed signal in coding projection format (hence the name “cross-format”). Partial (only forward) projection format conversion errors and coding errors are measured;
3. Coding distortion measurement: WS-PSNR and PSNR are measured between the input to the codec and the output of the codec. Only coding errors are measured, and projection format conversion errors are not measured;

Additionally, the CTC includes viewport quality evaluation using viewports generated from the original signal in the source projection format and the reconstructed signal in the coding projection format. For more details on the CTC, readers are referred to [2].

## S-PSNR

S-PSNR was original proposed in [9]. It uses a set of uniformly sampled positions on the sphere to measure 360-degree video quality. S-PSNR is calculated using the following steps:

1. For a point “s” on the sphere, apply 3D-to-2D coordinate mapping to find the corresponding positions “r” and “t” in the two inputs to metric calculation (that is, reference signal and test signal in Figure 12);
2. Calculate the distortion between the samples values at “r” and “t”;
3. Calculate the overall distortion by accumulating the distortion at each “s” in the set of points, and calculate PSNR from distortion

360Lib uses the same set of 655,362 points as provided in [9].



**Figure 12. S-PSNR calculation**

Two variants of the S-PSNR metric are supported in 360Lib. The first one uses nearest neighbor rounding when “r” and/or “t” are at fractional sample positions in order to avoid introducing impact due to interpolation filters when calculating distortion. It is referred to as S-PSNR-NN. The second variant uses interpolation filters to calculate sample values at fractional positions. It is referred to as S-PSNR-I.

## WS-PSNR

Weighted to Spherically uniform PSNR (WS-PSNR) calculates PSNR based on all samples. The distortion is weighted by sample area on corresponding spherical surface. WS-PSNR for omnidirectional video is defined via the WMSE.

|  |  |
| --- | --- |
|  | (49) |

where is the maximum possible intensity of the image.

|  |  |
| --- | --- |
|  | (50) |

where and are intensities in location of the test and reference images, and  is the corresponding mapped area on spherical surface for each sample .

[Ed. (YY): need to add more details about the weights used each projection format]

## CPP-PSNR

Craster parabolic projection PNSR [9]is a generalization of omnidirectional image with a uniform samples distribution close to the distribution on the sample on a surface of unit sphere.

No additional weighting is applied to transformed samples.

Spherical coordinates are mapped on the 2D plane (Figure 10) using following equations:

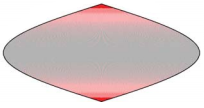
|  |  |
| --- | --- |
|  | (51) |

where m and n are horizontal and vertical integer luma sample position in CPP projection respectively; and are longitude and latitude in spherical coordinate system (Figure 2); sphere has . Corresponding sampling position in origin projection may be derived using map3Dto2D() method of corresponding projection class.

CPP dimensions are derived from input projection parameters in a manner that no additional samples are introduced. Thus CPP width () and CPP height ():

|  |  |
| --- | --- |
|  | (52) |

where is coding ERP width for corss projection metrics and hi-fidelity ERP width for end-to-end metrics.



**Figure 13. Craster parabolic projection image**

Color coding on Figure 13 represents a number of samples in ERP line used to represent single point on the sphere surface.

# References

1. https://jvet.hhi.fraunhofer.de/svn/svn\_360Lib/trunk
2. E. Alshina, J. Boyce, A. Abbas, Y. Ye, “JVET common test conditions and evaluation procedures for 360° video, JVET-E1030,” Jan. 2016, Geneva, Switzerland.
3. Equirectangular projection, https://en.wikipedia.org/wiki/Equirectangular\_projection
4. Cubemap, https://en.wikipedia.org/wiki/Cube\_mapping
5. Lambert cylindrical equal-area projection, <https://en.wikipedia.org/wiki/Lambert_cylindrical_equal-area_projection>
6. Octahedron, <https://en.wikipedia.org/wiki/Octahedron>
7. Rectilinear projection, <http://wiki.panotools.org/Rectilinear_Projection>
8. V. Zakharchenko, E. Alshina, K. P. Choi, A. Singh, A. Dsouza, AhG8: Icosahedral projection for 360-degree video content, Joint Video Exploration Team of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, JVET-D0028, Oct. 2016, Chengdu, China.
9. V. Zakharchenko, E. Alshina, A. Singh, A. Dsouza, AhG8: Suggested testing procedure for 360-degree video, Joint Video Exploration Team of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, JVET-D0027, Oct. 2016, Chengdu, China.
10. G. Van der Auwera, M. Coban, H. Fnu, M. Karczewicz, “AHG8: Truncated Square Pyramid Projection (TSP) For 360 Video,” Joint Video Exploration Team of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, JVET-D0071, Oct. 2016.

1. [C. Zhang](mailto:chuanyi.zhang@owlreality.com), [Y. Lu](mailto:yao.lu@owlreality.com), [J. Li](mailto:jisheng.li@owlreality.com), Z. Wen, “AhG8: segmented sphere projection (SSP) for 360-degree video content,” Joint Video Exploration Team of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, JVET-D0030, Oct. 2016.
2. H.-C. Lin, C.-Y. Li, J.-L. Lin, S.-K. Chang, C.-C. Ju, “AHG8: An efficient compact layout for octahedron format,” Joint Video Exploration Team of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, [JVET-D0142](http://phenix.int-evry.fr/jvet/doc_end_user/current_document.php?id=2723), Oct. 2016.
3. M. Yu, H. Lakshman, B. Girod, “A Framework to Evaluate Omnidirectional Video Coding Schemes”, IEEE International Symposium on Mixed and Augmented Reality, 2015.
4. Y. Sun, A. Lu, L. Yu, “AHG8: WS-PSNR for 360 video objective quality evaluation,” Joint Video Exploration Team of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, JVET-D0040, Oct. 2016, Chengdu, China.