ISO/IEC JTC 1/SC 29/WG 11
Coding of moving pictures and audio
Convenorship: UNI (Italy)

Document type:	**Approved WG 11 document**
Title:	**Test Model 5 for Immersive Video**
Status:	**Output document**
Date of document:	**2020-05-16**
Source:	**Video**
Expected action:	
No. of pages:	**35**
Email of convenor:	**leonardo@chiariglione.org**
Committee URL:	**mpeg.chiariglione.org**

Editors' integration notes

TMIV 5 – N19213:

- m52994-v3: Proposed simplifications of MIV (Philips).
- m52953: Restructuring of this document (AHG).
- m53506: CE1-related: HEVC Multiplexing (Philips).
- m53701: CE3-related: Additional patch dilation in temporal patch redundancy removal (ETRI).

TMIV 4 – N19002:

- m51604: Poznan response to CE3: Spatio-temporal patch redundancy removal.
- m52320: ZJU response to CE2: Culling for viewport rendering.
- m52350: Philips response to CE1: MIV HLS bitstream codec.
- m52365: Depth-map scaling.
- m52413: InterDigital response to CE2: Synthesizer.
- m52414: InterDigital response to CE2: Graph-based pruning.
- m52475: Object-based implementation.

TMIV 3 – N18795:

- m49958: Intel response to CE1: Grouping.
- m49962: Philips response to CE2: Pruning.
- m50949: Object-based immersive coding.
- m51439: Depth occupancy coding.
- m51487: Viewing space.

TMIV 2 – N18577:

- No tool adoption.
- Inclusion of operating modes.

TMIV 1 – N18470:

- Document established based on CfP responses reviewed during MPEG 126th meeting.

INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 11 N19213

April 2020, Alpbach (AU) Virtual

Source:	Video
Title:	Test Model 5 for Immersive Video
Editors:	Basel Salahieh, Bart Kroon, Joel Jung, Adrian Dziembowski

Abstract

The Video sub-group has established the fifth working draft and the fifth test model for immersive video during the 130th MPEG meeting (April 2020) after evaluating the core experiment results and related contributions. The test model consists of this document and the reference software, providing an encoder and decoder/renderer in alignment with the specification. This document serves as a source of general tutorial information on the MPEG Immersive Video (MIV) design. It defines terminology used, process and data flow, operating modes, and description of algorithmic components adopted by the video group for the test model.

1 Introduction

The MPEG-I project (ISO/IEC 23090) on *coded representation of immersive media* includes Part 2 *Omnidirectional MediA Format* (OMAF) version 1 published in 2018 that supports 3 Degrees of Freedom (3DoF), where a user's position is static but its head can yaw, pitch and roll. However, rendering flat 360° video, *i.e.* supporting head rotations only, may generate visual discomfort especially when objects close to the viewer are rendered. 6DoF enables translation movements in horizontal, vertical, and depth directions in addition to 3DoF orientations. The translation support enables interactive motion parallax providing viewers with natural cues to their visual system and resulting in an enhanced perception of volume around them. At the 125th MPEG meeting, a call for proposals [1] was issued to enable head-scale movements within a limited space. This has resulted in new MPEG-I Part 12 *Immersive Video* (MIV).

At the 129th MPEG meeting the fourth working draft of MIV has been realigned to use MPEG-I Part 5 *Video-based Point Cloud Compression* (V-PCC) as a normative reference for terms, defintions, syntax, semantics and decoding processes. At the 130th MPEG meeting this alignment has been completed by restructuring Part 5 in a common specification *Video-based*

Volumetric Coding (V3C) and annex H *Video-based Point Cloud Compression* (V-PCC). V3C provides extension mechanisms for V-PCC and MIV. The terminology in this document reflects that of V3C and MIV.

2 Scope

The normative decoding process for MPEG Immersive Video (MIV) is specified in working draft 5 of immersive video (WD) [2]. The TMIV reference software (Annex A) provides a reference implementation of non-normative encoding and rendering techniques and the normative decoding process for the MIV standard.

This document provides an algorithmic description for the TMIV encoder and decoder/renderer. The purpose of this document is to promote a common understanding of the coding features, in order to facilitate the assessment of the technical impact of new technologies during the standardization process. *Common test Conditions for Immersive Video* [3] provides test conditions including TMIV-based anchors.

3 Terms and definitions

For the purpose of this document, the following definitions apply in addition to the definitions in MIV specification [2] clause 3.

Table 1: Terminology definitions used for TMIV.

Term	**Definition**
Additional view	A *source view* that is to be pruned and packed in multiple *patches.*
Basic view	A *source view* that is packed in an atlas as a single *patch*.
Clustering	Combining pixels in a pruning mask to form *patches*.
Culling	Discarding part of a rendering input based on target viewport visibility tests.
Entity	An abstract concept to be defined in another standard. For example, entities may either represent different physical objects, or a segmentation of the scene based on aspects such as reflectance properties, or material definitions.
Entity component	A multi-level map indicating the *entity* of each pixel in a corresponding *view representation*.
Entity layer	A *view representation* of which all samples are either part of a single *entity* or non-occupied.
Entity separation	Extracting an *entity layer* per a *view representation* that

	includes the desired *entity component.*
Geometry scaling	Scaling of the geometry data prior to encoding, and reconstructing the nominal resolution geometry data at the decoder side.
Inpainting	Filling missing pixels with matching values prior to outputting a requested *target view*.
Mask aggregation	Combination of *pruning masks* over a number of frames, resulting in an aggregated pruning mask.
Metadata merging	Combining parameters of encoded atlas groups.
Omnidirectional view	A *view representation* that enables rendering according to the user's *viewing orientation*, if consumed with a head-mounted device, or according to user's desired *viewport* otherwise, as if the user was in the spot where and when the view was captured.
Patch packing	Placing *patches* into an *atlas* without overlap of the occupied regions, resulting in patch parameters.
Pose trace	A navigation path of a virtual camera or an active viewer navigating the immersive content over time. It sets the *view parameters* per frame.
Pruning	Measuring the interview redundancy in *additional views* resulting in *pruning masks.*
Pruning mask	A mask on a *view representation* that indicates which pixels should be preserved. All other pixels may be *pruned*.
Source splitting	Partitioning views into multiple spatial groups to produce separable *atlases*.
Source view	Indicates source video material before encoding that corresponds to the format of a *view representation*, which may have been acquired by capture of a *3D scene* by a real camera or by *projection* by a virtual camera onto a surface using source *view parameters*.
Target view	Indicates either *perspective viewport* or *omnidirectional view* at the desired *viewing position* and *orientation*.
View labeling	Classifying the source views as *basic views* or *additional views*.

4 Description of encoder processes

4.1 Introduction

The TMIV encoder has a "group-based" encoder, described in Figure 1, at higher level which invokes for each group a "single-group" encoder described in Figure 2. The group-based encoder has the following stages:

1. Preparation of source material by:
    - Assessing the geometry (depth map) quality for each source view.
    - Splitting source views in groups.
    - Labeling source views as basic view or additional view.
2. Encoding of each group separately (using the associated subset of splitted source views).
3. Formatting of the bitstream (includes a merging sub stage to combine sub bitstreams of same type produced by each single-grup encoder together) which is V3C sample stream with MIV extensions and related SEI messages.
4. HEVC encoding of video sub bitstreams (each separately) using HM16.16.
5. Multiplexing to combine the formatted bitstream with the video sub bitstream into a single MIV-complied bitstream.

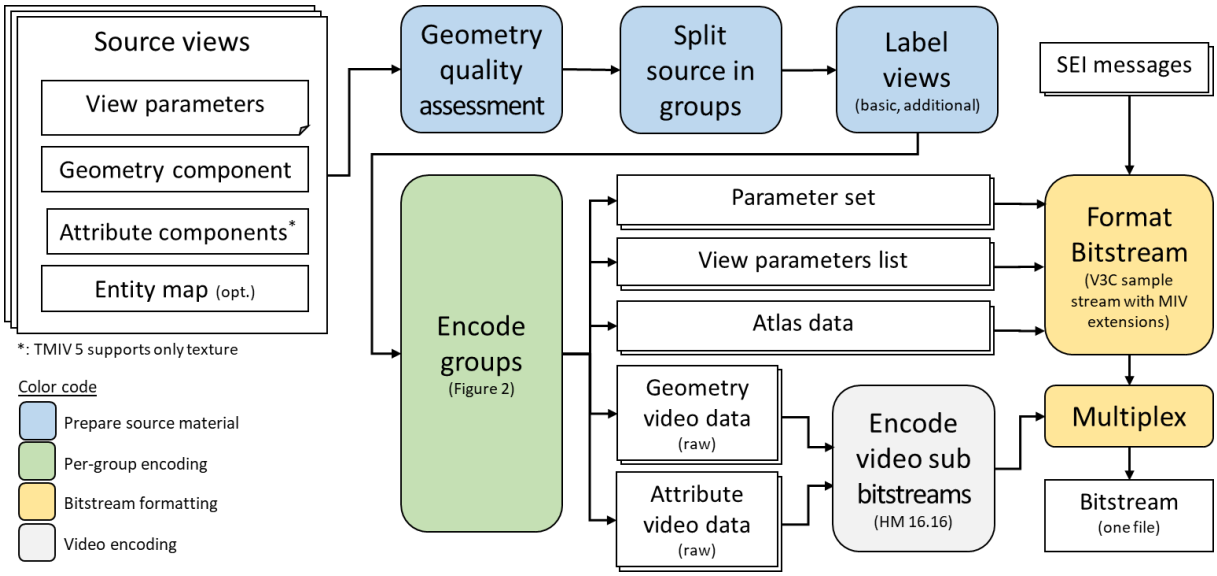

Figure 1: Top-level diagram of the TMIV group-based encoder.

The single-group encoder acts on the selected source views for a given group and has the following stages:

1. Automatic parameter selection to set the atlas parameters (*i.e.* number of atlases, and the frame size of each of the atlases).
2. Separation of views into entity layers (optional stage).
3. Pruning of redundant information, aggregating the pruned masks over an intra-period. and clustering of preserved pixels for each group and entity.
4. Packing of patches and generation of video data per group (Figure 3).
5. Quantization and scaling of geometry video data per atlas.

The remainder of this section explains the encoder input, output and each of the encoder processes in more detail.

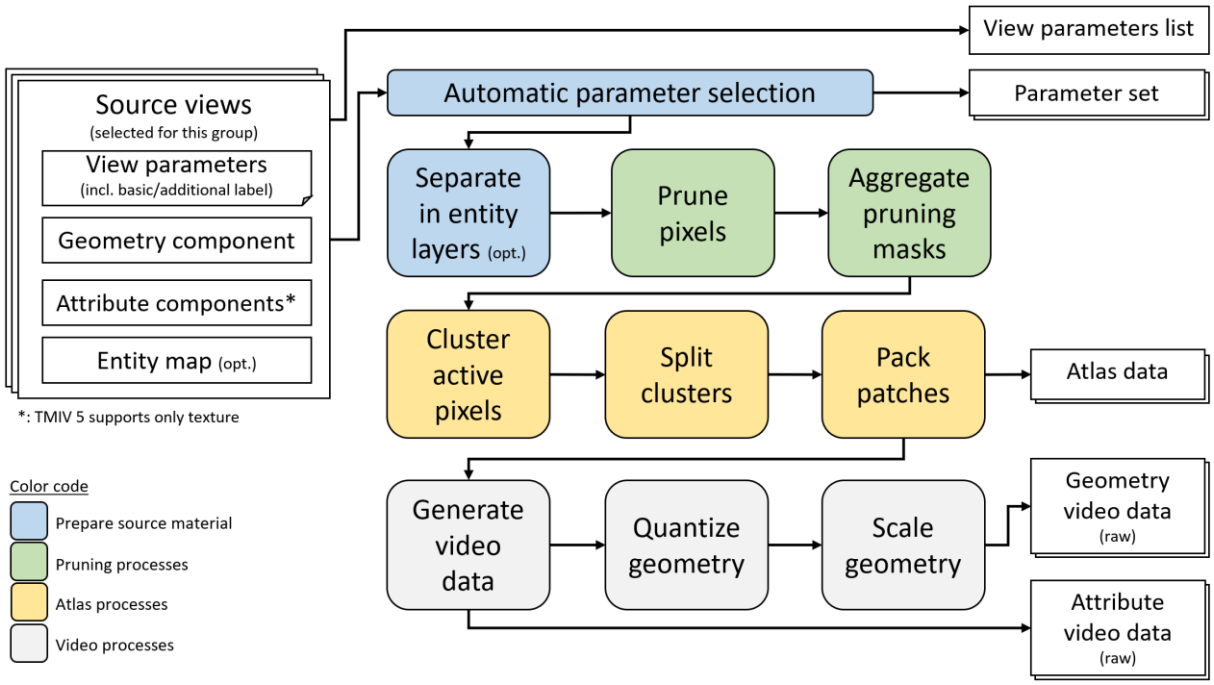

Figure 2: Top-level diagram of the TMIV single-group encoder.

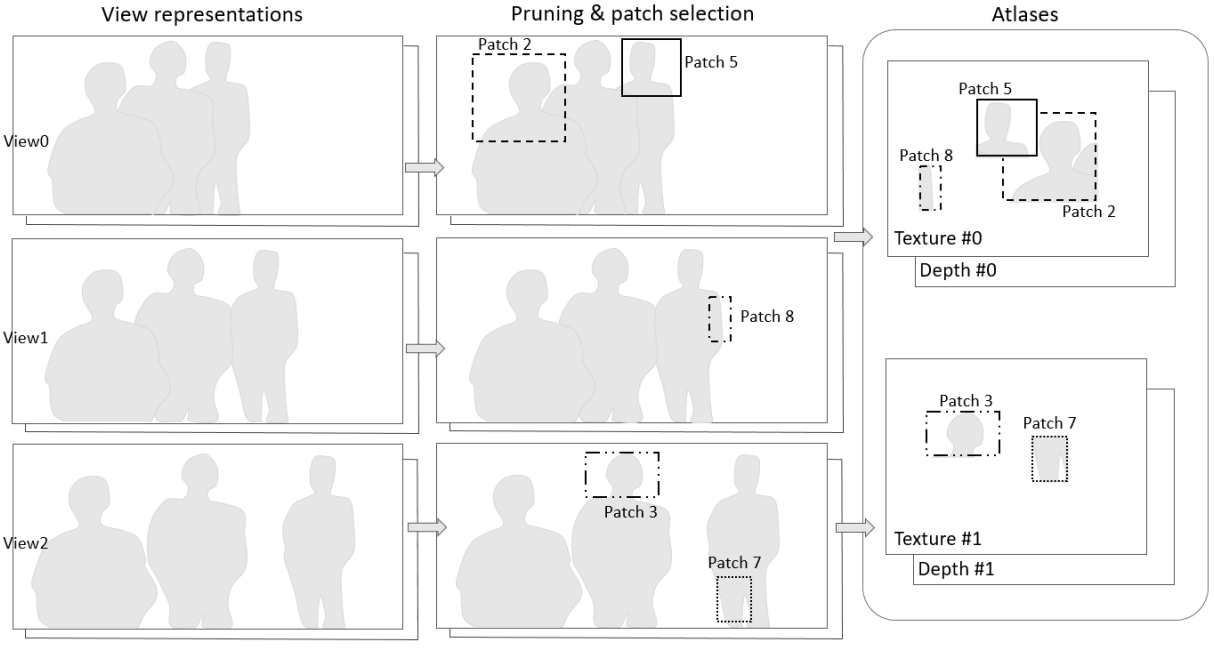

Figure 3: Representing source views using patch atlases.

4.2 Encoder inputs

The input to the TMIV encoder consists of a list of source views (Figure 4). The source views represent projections of a 3D real or virtual scene. The source views can be in equirectangular, perspective, or orthographic projection. Each source view should at least have view parameters

(camera intrinsics, camera extrinsics, geometry quantization, etc.), and a geometry component in the form of 8-16 bits raw video with range/invalid sample values. A source view may have attribute components but for this version of TMIV only YC_BC_R 4:2:0 10 bits texture is supported. A source view may also comprise an entity component. The set of components has to be the same for all source views.

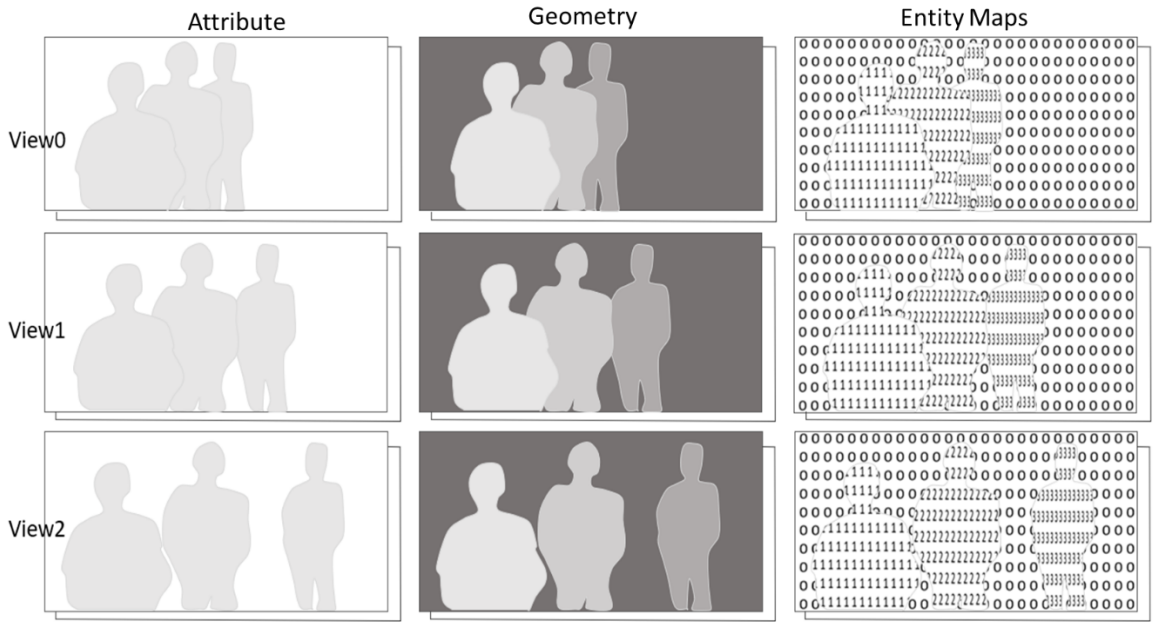

Figure 4: Input source views composed of attribute and geometry components, and entity maps.

4.3 Encoder outputs

The output of the TMIV encoder is a single file according to the V3C sample stream format containing a single V3C sequence. Most parameter sets have MIV extensions enabled and the special atlas is present. The view parameter list is sent once and, while the standard has support, is never updated in this version of the test model. For each of the regular atlases, there are sub bitstreams with patch data, geometry video data, and attribute video data (if present). Atlas and patch parameters include groups and entity ID's respectively[1].

4.4 Distribution of source views in groups

Source views can be divided into multiple groups. The grouping helps outputing local coherent projections of important regions (*e.g.* belonging to foreground objects or occluded regions) in the atlases per group as oppose to having fewer samples of those regions when processing all source views as a single group. An automatic process is implemented to select views per group, based on the view parameters list and the number of groups to obtain. The source views are being distributed accordingly in multiple branches, and each group is encoded independently of each other.

[1] There may be only one group and/or entity in which case group-based and/or entity-based coding is effectively disabled.

Source splitting operates as follows: a views pool including all available source views is formed and the number of views per group is set (by dividing the number of source views by the number of groups). The view parameters list is used to identify the range the views are spanning in Cartesian scene coordinates. The dominant coordinate axis is selected as a basis to set key positions. Key positions are located at the maximum view positions of the dominant axis across view in the views pool. Distances of views to these key positions are computed. Based on the number of views for the group, the closest views to the first key position are selected and removed from the views pool. Then a second key position is identified and the process is repeated covering the distribution of all source views across the chosen number of groups.

4.5 View labeling

This process labels the source views as basic views or additional views. This process can be skipped to encode only complete views (*i.e.* in this case all views are regarded as basic views and not pruned).

The process includes two steps:

- Determination of the number of basic views, considering direction deviation, field of view, distance and overlap between views.
- Selection of the basic views, considering the distance to a central view position and some overlap.

The input of the process is the source views and the source view parameters list. The output of the process is a list of basic views and a list of additional views.

4.5.1 Determination of the number of basic views

First, a pair of views (view m, view n) that has the largest direction deviation according to the equation $(m, n) = \text{argmax}\{\theta(i, j)\}$ is found, where i and j are the indices the source views $0,1, \ldots, \text{N} - 1$, with $\text{i} \neq \text{j}$ as shown in Figure 5.

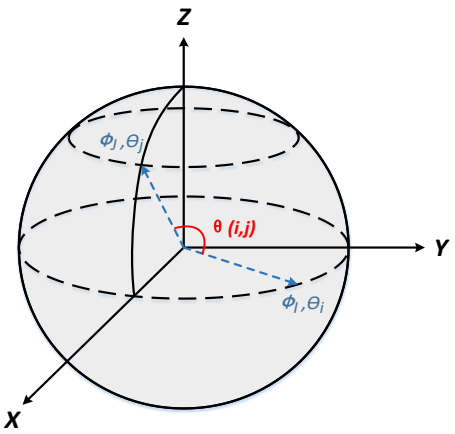

Figure 5: Explanation of the direction's deviation.

When two pairs provide the same maximum direction deviation, the pair which has the largest sum of field of views (FOVs) is selected: $(m, n) = \text{argmax}\{FOV_i + FOV_j\}$.

When two pairs provide the same maximum sum of FOVs, the pair which has the largest distance between each other is selected:

$$(m,n) = \text{argmax}\left\{\sqrt{(Tx_i - Tx_j)^2 + (Ty_i - Ty_j)^2 + (Tz_i - Tz_j)^2}\right\}$$

Second, the overlap between the two views is computed, as illustrated in Figure 6.

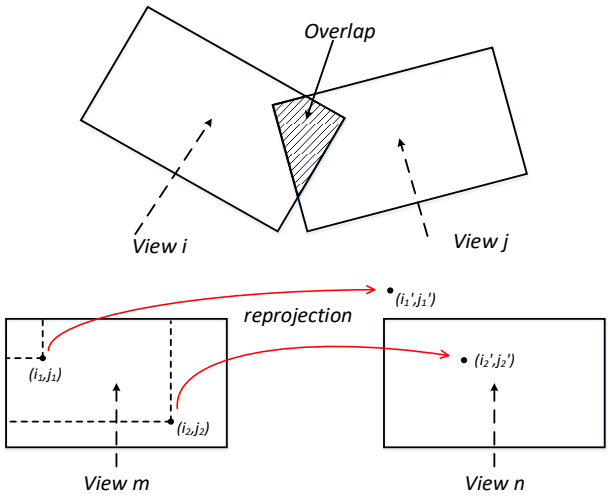

Figure 6: Illustration of the overlap and its calculation.

Each pixel position (i,j) of the view m is projected on the view n in position (i',j'). The weighted sum of overlapped pixels whose new position (i',j') is in the FOV of view n is computed as in the equation below, with $isoverlap(i,j) = 1$ meaning that (i,j) is visible by both m and n views:

$$\text{overlap} = \frac{\sum weight(i,j) \cdot isoverlap(i,j)}{\sum weight(i,j)} \cdot FOV_m$$

where $weight(i,j)$ is the spherical weight of each pixel position (i,j). FOV is in Steradian unit.

Finally, the number of basic views is determined:

- If overlap $\geq 0.5 \cdot \min(FOV_i, FOV_j)$, only one basic view is selected.
- If overlap $< 0.5 \cdot \min(FOV_i, FOV_j)$, multiple basic views including view m and view n are selected.

The number of basic views cannot exceed two when a single group is used, and cannot exceed one per group, when several groups are used.

4.5.2 Selection of the basic views

When the number of basic views is one, the following applies:

The source view that has the largest FOV is selected as the basic view $j = \text{argmax}\{FOV_i\}$. If several views have the same largest FOV then the following applies:

- Calculate the central camera position of the source capturing system given the source camera parameters list.

$$x_{center} = \frac{1}{N}\sum_{i=0}^{N-1} Tx_i \, , y_{center} = \frac{1}{N}\sum_{i=0}^{N-1} Ty_i \, , z_{center} = \frac{1}{N}\sum_{i=0}^{N-1} Tz_i$$

- Select as the basic view the source view which camera position is the closest to the central camera position:

$$l = \text{argmin}\left\{\sqrt{(Tx_k - x_{center})^2 + (Ty_k - y_{center})^2 + (Tz_k - z_{center})^2}\right\}$$

When several basic views are needed, the following applies:

The views *m* and *n* found are selected as basic views. The view *k* which has the largest direction deviation with view *m* and view *n* is determined:

$$k = \text{argmax}\{\min\left(\theta(k,m), \theta(k,n)\right)\}$$

If the view *k* has less than 50% FOV overlap with the already selected basic views *m* and *n*, then view *k* is selected as a basic view, and the same process is repeated to find the next basic view. Otherwise the process stops. All other non-selected source views are labeled as additional views.

4.6 Automatic parameter selection

Some of the parameters of the TMIV encoder are automatically calculated based on the camera configuration or at most the first frame of the source views. This section describes these processes.

4.6.1 Geometry quality assessment

The quality of the geometry is assessed automatically based on the first frame of the geometry component. Each input view is reprojected to the position of all remaining input views. Then, for every reprojected pixel it is checked if reprojected geometry value is higher than 97% (by default, in general: 1.0 – "*blendingFactor*") of geometry value of collocated pixel or any of its neighbors in the target view (in a 3×3 neighborhood). If this condition is not fulfilled, the pixel is counted as inconsistent. If the number of inconsistent pixels between any pair of input views is higher than a threshold (default: 0.1% of total area of the input view) the quality of the geometry is supposed to be low.

4.6.2 Atlas frame size calculation

In V3C, each atlas has a frame size (asps_frame_width × asps_frame_height) to which all components (atlas data, occupancy video data, geometry video data, and attribute video data) are scaled up as part of the reconstruction. The block to patch map is scaled down by the block size (often 8 or 16) with patch positions and sizes aligned by this amount. In MIV, there is no occupancy video data, the attribute video data is always at nominal resolution, and the geometry video data is scaled down by an integer factor $N \geq 1$.

The encoder calculates the number of atlases per group and atlas frame size automatically. This computation is related to constraints on the maximum size of a picture (considering the luma only), the maximum sample rate (in Hz) of the luma, and a total number of allowed decoder instantiations.

Taking into account the MIV restrictions, and assuming one attribute, the following applies:

- number of atlases = number of atlases per group · number of groups,
- luma picture size = atlas frame width · atlas frame height,
- luma sample rate = $(1 + 1/N^2)$ luma picture size · frame rate · number of atlases,
- number of decoder instantiations = 2 · number of atlases.

To meet the constraints, the following algorithm is applied:

1. The atlas frame width is set to the widest source view,
2. The number of atlases per group is set high enough to reach or exceed the maximum luma sample rate, but within the maximum number of atlases,
3. The atlas frame height is set as large as possible within the constraints.

The calculations are aligned on the block size.

Without those constraints, there is one atlas per source view and the nominal atlas resolution of each atlas is set equal to the resolution of the corresponding source view. This enables complete (unconstrained) transmission of all source views.

4.7 Separation in entity layers

TMIV has the ability to operate in entity mode when entity maps are provided for the source views (and *maxEntities* > 1 in the TMIV config). In this mode, the patches extracted and packed within the atlases have active pixels that belong to single entity per patch, thus it is possible to tag each patch with its associated entity ID. This enables selective encoding and/or decoding of entities separately if desired (set by *EntityEncodeRange* and/or *EntityDecodeRange* respectively in TMIV config) resulting in savings in utilized bandwidth and improved quality. If entity coding mode is chosen, then the source views (attribute and geometry components) including the basic ones are sliced into multiple layers such that each layer includes content belong to one entity at a time. Then following encoding stages are invoked on entity bases such that the layers across all views that blong to the same selected entity are prunned, aggregated, and clustered together.

4.8 Pixel pruning

A multiview representation of a scene inherently has interview redundancy. The pruner selects which areas of the views may be safely pruned. The pruner operates on a per-frame basis, receiving multiple views with geometry component and camera parameters, and outputting masks per view and frame of the same size. For additional views, mask values are either 'pruned' (0) or 'preserved' (255). For basic views, all pixels are 'preserved'.

The method has been devised with the following goals in mind:

- Remove redundancy between all pairs of views,
- Prefer fewer larger patches,
- Maintain a realistic complexity,
- Consider temporal consistency.

4.8.1 Pruning graph

In order to determine interview redundancy, the pruner performs data projection between input views. To achieve the first two goals, the pruner creates a pruning graph, which defines hierarchy of view pruning (Figure 7). The pruning graph is created in a greedy fashion, which allows to achieve the third goal.

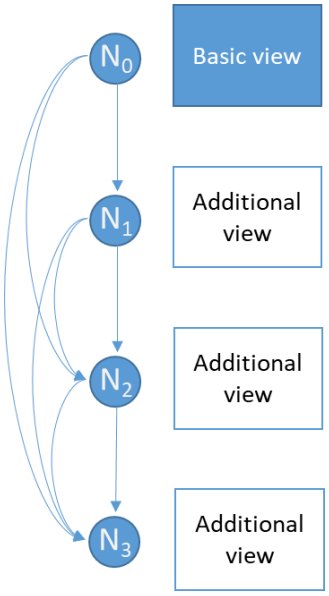

Figure 7: Pruning graph for one basic and three additional views. Basic view is assigned to a root node (node id: N_0), each additional view is assigned to a node N_i, which is a child node of all nodes N_j where $j < i$.

Pruning graph creation:

1. Insert basic views into the pruning graph (as root nodes).
2. Project all pixels of all basic views to each additional view.
3. Create the *pruning mask* for each additional view (cf. section 4.8.2).
4. Select the additional view with maximum number of preserved pixels (to prefer larger patches).
5. Insert selected additional view into the pruning graph (as a child node of all nodes already in graph) and stop if all the views are assigned to nodes in the pruning graph.
6. Project all preserved pixels of selected view to remaining additional views.
7. Update the *pruning mask* for each remaining additional view.
8. Go to 4.

The temporal consistency is maintained due to the preservation of the view hierarchy over time. The pruning graph can change only if view parameter list changes (only at the first frame with

current CTCs).

The pruning graph is transmitted as part of the view parameters.

4.8.2 Pruning mask creation

The pruner uses two criteria to determine if a pixel may be pruned:

- The pixel should be synthesized from the views higher up in the hierarchy (it should be preserved in view assigned to parent node and pruned in view assigned to child node).
- The difference between synthesized and source geometry should be less than a threshold.

A mask typically has holes and irregularities which are cleaned up by a classical iterative erosion and dilation method on a 3×3 structuring element:

- For the erosion, a pixel that has at least one empty neighbor is discarded (pixel = 0).
- For the dilation, a pixel that has at least one non-empty neighbor is filled (pixel =1).

In the case of entity coding mode, no patch can have pixels belong to more than one entity at a time. Thus, the basic masks are updated based on the entity maps such that pixels are turned on only for pixels belong to the processed entity. The other pruning masks are refined as well to accurately represent the entity.

4.9 Pruning mask aggregation

An accumulation of the pruning masks is done at the pixel level, across the different frames of the intra period by implementing the logical operation OR as follows:

```
aggregatedMask[i]@current_frame =
    max(Mask[i]@current_frame, aggregatedMask[i]@previous_frame)
```

The mask is reset at the beginning of each intra period. The process is completed at the end of the intra period by outputting the last accumulation result. Figure 8 illustrates for a pruned view at frame i, the accumulation of non-null samples (drawn in white) between the frame i and frame i+k within an intra period; it can be seen that contours are getting thicker on the changing parts of the geometrycomponent accounting for the motion within the scene.

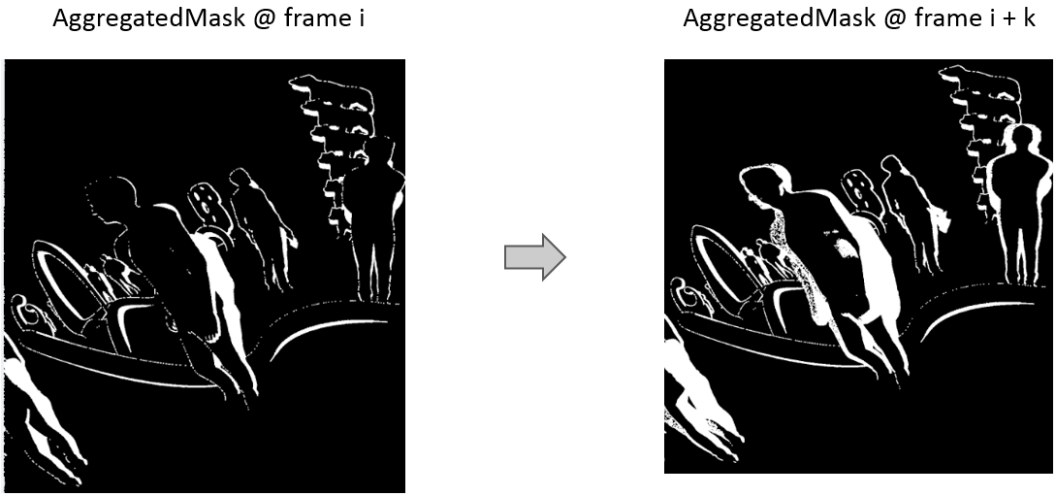

Figure 8. AggregatedMask evolution within an intra period.

In the case of entity coding mode, entity's updated pruning masks are aggregated across frames (per entity) to account for entity motion within an intra-period and stored so they can be used later for the entity clustering stage.

4.10 Clustering active pixels

This block is in charge of identifying what is called "clusters". A cluster is a set of connected mask pixels of 1s value obtained by the mask aggregator. The connection criteria of one pixel is the presence of at least one other pixel among the 8 neighbors.

Figure 9: 8-pixel neighborhood for defining the connectivity criteria for region growing.

An example of the clustering is illustrated in Figure 10 where each cluster of an already pruned view is represented by a specific false color. The cluster are then sorted by a decreasing size order. The parameters associated to each cluster are:

- x and y positions of the top left corner of the bounding box.
- Width and height of the bounding box.

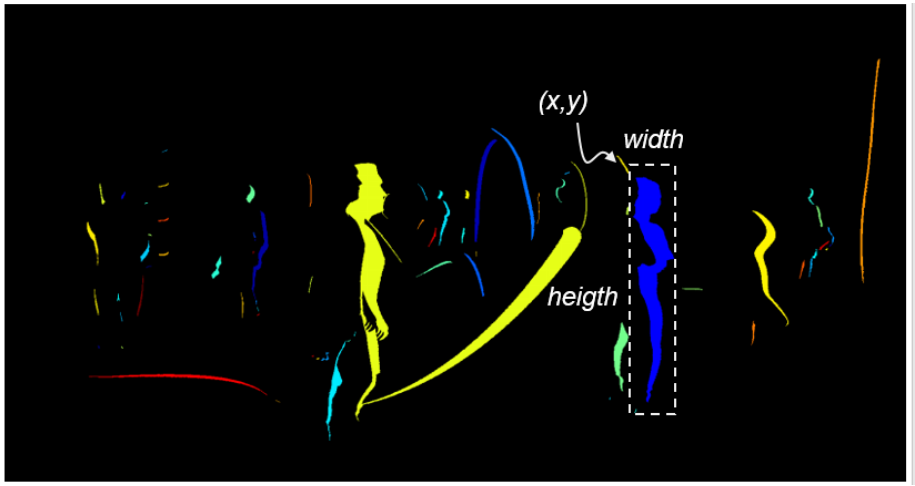

Figure 10: Clusters represented in false color on a pruned view.

In case of entity coding mode, the aggregated entity masks are passed to the clusterer to perform clustering for each entity separately and the resulted clusters are tagged with the related entity ID and queued together.

4.11 Cluster splitting

In order to reduce spatial redundancy of data in the atlas, irregularly-shaped clusters (*e.g.* large yellow cluster in Figure 10) are split. Each cluster is split into two smaller clusters if the total area of bounding boxes of two new resulting clusters is smaller than the area of bounding box of the initial cluster by at least 10%. In order to decide how to split a cluster, the total area of bounding boxes of two sub clusters is minimized. The split is done along a line that is parallel to the shorter side of the cluster's bounding box. This approach allows to divide an L-shaped cluster.

For other cluster shapes (*e.g.* C-shape), this approach does not split the cluster. Therefore, an additional cluster splitting is performed. Within the entire bounding box of the cluster, the number of *Alignment*×*Alignment* (cf. section 4.12) blocks that contain pixels belonging to the cluster is calculated. This number is divided by the total number of blocks within the analyzed bounding box. If that ratio is smaller than 0.3, the cluster is split in half. Splitting of C-shaped cluster usually results in two L-shaped clusters.

The cluster splitting of irregularly-shaped cluster is a recursive method, as depicted in Figure 11.

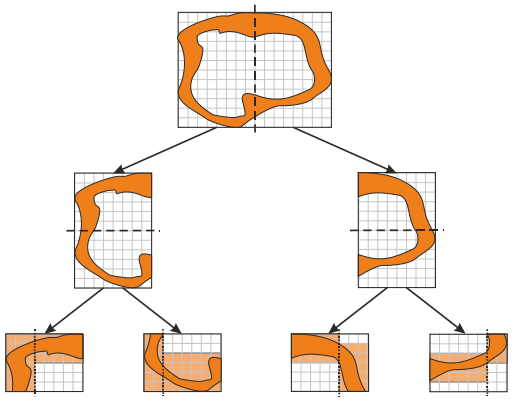

Figure 11: Recursive splitting of the cluster; dashed lines: C-splitting, dotted lines: L-splitting.

4.12 Patch packing

The packing process sequentially packs each cluster into the atlases. The input parameters are the following:

- *"Alignment"*: the patch size and the patch position are multiple of the alignment (number of pixels). Default value is 8.
- *"MinPatchSize"* is the number of pixels of the smallest border of the patch, below which the patch is discarded. Default value is 8.
- *"Overlap"* is the number of pixels which will be added to a frontier of a newly split patch; it prevents seam artefacts. Default value is 1.
- *"PiP"* is a flag enabling the Patch-in-Patch feature when equal to 1. It allows the insertion of patches into other patches. Default value is 1.

The packing process is based on a version of MaxRect algorithm [6]. It considers the available "Used Space" first, by examining the space which is effectively occupied. In a second time, "Free space" is considered. It is made of intricated loops which are described by the following pseudo-code:

```
For each cluster:
    For each atlas:
        Push the cluster in "Used Space" (0° rotation first, 90° otherwise)
        If the push failed:
            Push the cluster into "Free Space" (0° rotation first, 90° otherwise)
            If the push failed:
                Split the cluster into 2 parts by its largest border
                For each resulting 2 parts:
                    If  smaller than MinPatchSize:
                        Discard the patch
                    Else:
                        Put the part in the cluster priority list
```

The output is a patch list for each atlas with all information necessary to recover the patches at the decoder side:

- The location in the atlas (patch_pos_in_atlas_x, patch_pos_in_atlas_y) along with the atlas id (AtlasId).
- The location in the original view representation (patch_pos_in_view_x, patch_pos_in_view_y), and its dimensions (patch_width_in_view, patch_height_in_view).
- The view id (ViewId), which itself refers to the de-projection parameters for that view in the decoder.
- The entity id (entityId) in case *maxEntities* > 1.
- A possible rotation by i*90° where i = 0 or 1 (2 and 3 are supported by the standard but not implemented in TMIV).
- A possible vertical flip.

The packing operation from view representation to atlas is done with rotation (first) then vertical flipping (second). Only two rotations are tested by the TMIV (among eight configurations supported by the standard, considering combinations of rotations and flipping).

Note that at the encoding side, the rotation of 90° is here meant to be from view representation to atlas and is counter-clockwise, *i.e.* rotates the Y-Axis on the X axis, as illustrated in Figure 12.

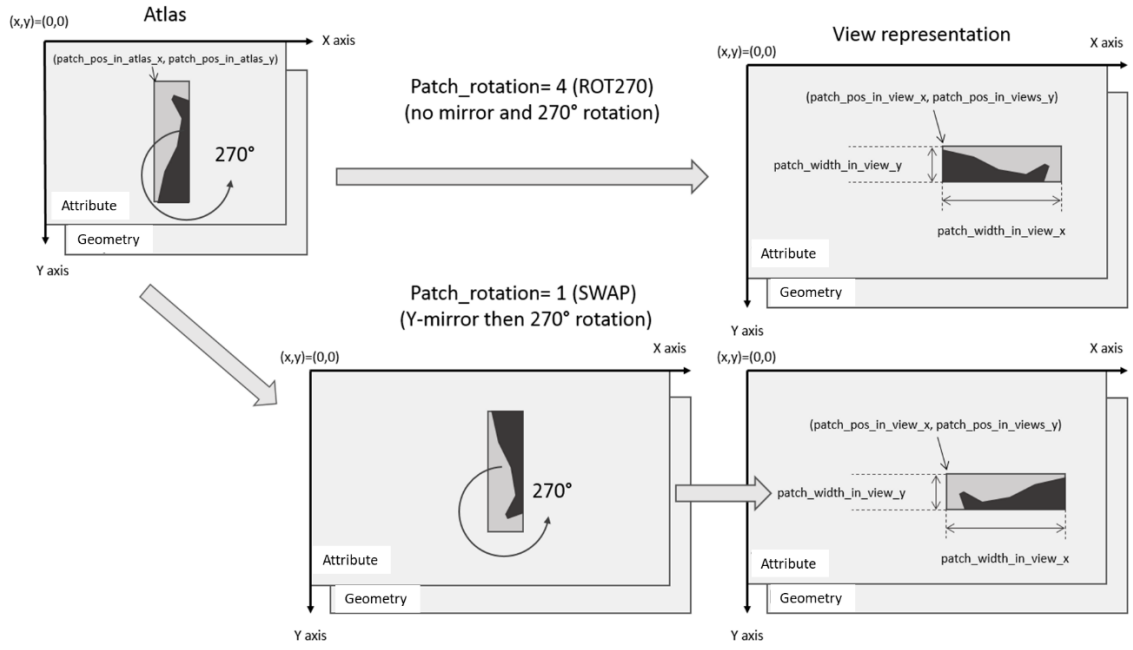

Figure 12: Meaning of patch related parameters.

4.13 Video data generation

The final operation within the single-group encoder is writing the patches in the buffer allocated to the atlas (both the geometry and the attribute components). Note that for the enity coding mode, the content of a given patch is extracted from the associated entity view generated by an

entity separator based on the patch's entity ID. This assures having the right entity content (attribute and geometry) being written to the patches within the formed atlases.

Figure 13 illustrates the generation of an atlas, with the successive write of patch 2, 5 and 8. While the packing algorithm is using the information of samples that are mandatory and are non-pruned (represented by area inside the perimeters in dash), the copy of the patch is rectangular, resulting in a heap of possibly overlapping rectangles.

The occupancy of these rectangles is set separately for each frame by analyzing non-aggregated pruning mask. At first, the mask is dilated iteratively 2×*Alignment* times using 3×3 structuring element. A pixel of a patch is copied to the atlas if there is any non-zero value in a collocated *Alignment*×*Alignment* (cf. section 4.11) block of dilated pruning mask. Otherwise, it is filled using neutral attribute and its gemoetry is set to 0, expressing the invalidity of a sample.

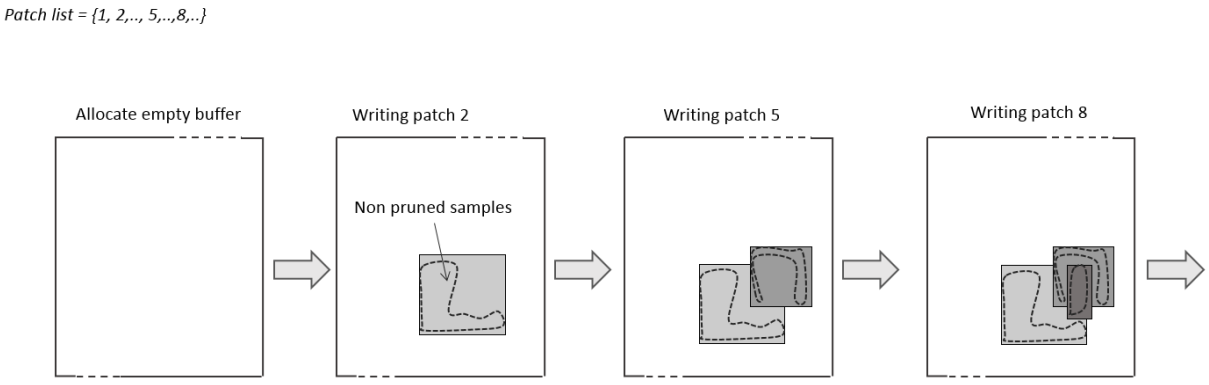

Figure 13: Successive writing of patches into an atlas.

4.14 Geometry coding

An atlas value is either "invalid/non-occupied" or a geometry value expressed in meters, with maximum geometry value set to 1 km. The Working Draft [2] specifies how to encode occupancy information within geometry atlases. The decoding is based on a normalized disparity range, a geometry threshold and an optional clamping start value. These values are signalled per view or even per patch. Assuming 10 bits full range geometry atlases, the transformation is described in pseudo-code as:

```
valid := x ≥ depthOccMapThreshold
if (valid) {
    normDisp := max(kilometer^-1,
        normDisp0 + (normDisp1023 - normDisp0) * (max(depthStart, x) ÷ 1023))
    depth := 1 / normDisp
}
```

Line 1 is part of the block to patch map decoder (Section 8.5.5 of [2]), lines 3...5 are part of the Synthesizer (Section 5.4) and lines 2 and 6 are implicit in the TMIV decoder.

The single-group encoder outputs rectangular patches with full occupancy so the occupancy coding capability of the Working Draft is not fully utilized by the TMIV encoder. Because of this, the geometry coder implements a simple method that recognizes two situations as depicted in Figure 14 and Figure 15:

- When a source view has only valid geometry values, depthOccMapThreshold is set to zero. This effectively encodes full occupancy (Figure 14).

- When a source view has invalid geometry values, depthOccMapThreshold is set to a configured value (T) and the normalized disparity range is adjusted such that the value 2T corresponds to the far geometry (Figure 15).

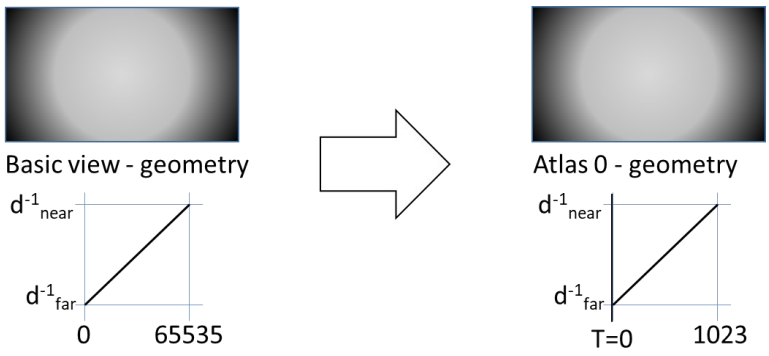

Figure 14: When the source material has only valid geometry values, the geometry coder only performs u(16) to u(10) scaling and the geometry threshold is set to zero to signal full occupancy.

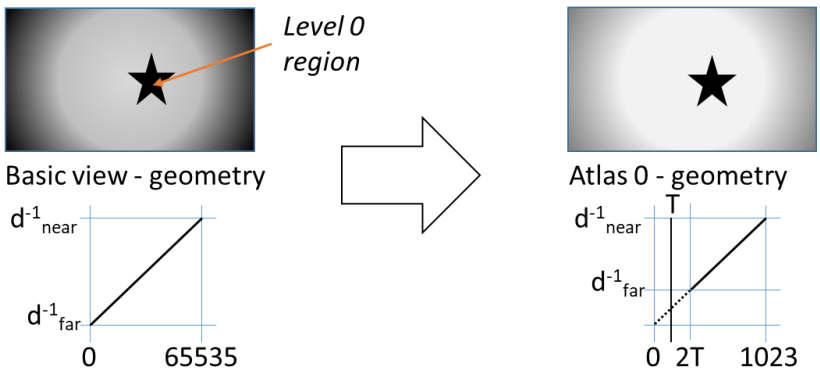

Figure 15: When the source material has invalid geometry values, the geometry coder not only performs u(16) to u(10) scaling, but it also sets the geometry threshold to a configurated value (T) and the normalized disparity range is modified such that value 2T corresponds to the far geometry.

4.15 Geometry downscaling

When the parameter 'depthDownScaleFlag' is asserted in the configuration, the geometry atlases are scaled-down by a factor of 2×2. The downscaling yields a lower overall pixel-rate and a higher geometry encoding quality for a given bitrate. For downscaling the geometry a 'max_pooling 2x2' filter is used. The assumption is made that foreground objects are encoded as high (bright) levels. The max_pooling filter does not produce 'in-between' geometry levels and the downscaled output has a known bias as foreground objects are slightly dilated. Such bias can be reverted on the decoder side.

4.16 Video encoding

The HEVC encoder, Main10 profile with Random Access configuration, is used to provide the attribute and geometry atlases encoded video in separate layers (4:2:0 10 bits format).

4.17 SEI messages

TMIV has a full parser and formatter that can handle various supplemental enhanced information (SEI) messages as part of the bitstream. Although some of these messages are not fully imlemented yet (as how they are being actually generated at the encoder side or being acted on the decoder and rendering side), the TMIV software does include unit tests to prove that their information can be encoded to and decoded from the bitstream.

A list of the SEI messages currently supported in TMIV includes:

- Decoded atlas data hash SEI: provides the hash information for the block to patch map per atlas to check if the ones being generated at the decoder side matches those at the encoder side.
- Enabled views per atlas SEI: informs the decoder and renderer of what views the patches included in the atlases are extracted from. This information is available at atlas level hence it provides direct access to speed up the processing.
- Objects in atlas SEI: informs the renderer of what objects are included per atlas in case of entity-coding mode is used.
- Recommended viewport SEI: informs the renderer of navigation paths (recommended by the content generator for instance) to synthesize viewports at and display to the viewer alternatively to the viewing position and orientation usually fed per viewer's motion.
- Viewing space SEI: informs the renderer of the viewing space supported by the decoded content that the viewer can move within and still observes decent-quality viewports.
- Viewing space handling SEI: informs the renderer of how to output the viewport in case the viewer is stepping outside the supported viewing space.

4.18 Bitstream formation and multiplexing

The output of the TMIV encoder is a V3C sample stream with MIV extensions (Figure 16). The V3C sample stream consists of a V3C parameter set, special atlas data, atlas data, geometry video data, and optionally attribute video data. The atlas data is a NAL sample stream which includes also the SEI messages. The special atlas data sub bitstream contains the view parameters list and the regular atlas data sub bitstreams which contain patch data. This patch data is sent only for intra frames and a frame order count NAL unit is used to skip all inter frames at once.

A restriction of MIV on V3C is that V3C units have to be grouped in chunks of frames, with all V3C units in a chunk corresponding to the same frame range. This restriction has the purpose of improving buffering. The current version of TMIV addresses the restriction in a trivial way by having only one V3C sequence with one V3C unit per type and atlas. This choice makes it

possible to use the HEVC test model (HM) encoder as an external tool to encode entire video sub bitstreams at once. The formatting and multiplexing is thus performed as follows:

1. Atlases of multiple groups are concatenated with renumbering of atlas ID's. Also, parameter set and atlas data of all groups are merged together into one parameter set and one atlas data units, respectively.
2. An intermediate bitstream is formatted that includes no video sub bitstreams.
3. All geometry and attribute video data is output as raw video files.
4. The raw video is encoded using the HEVC test model (HM) resulting in separate video sub bitstreams.
5. The intermediate bitstream plus all sub bitstreams are concatenated with insertion of suitable headers, to form the output bitstream.

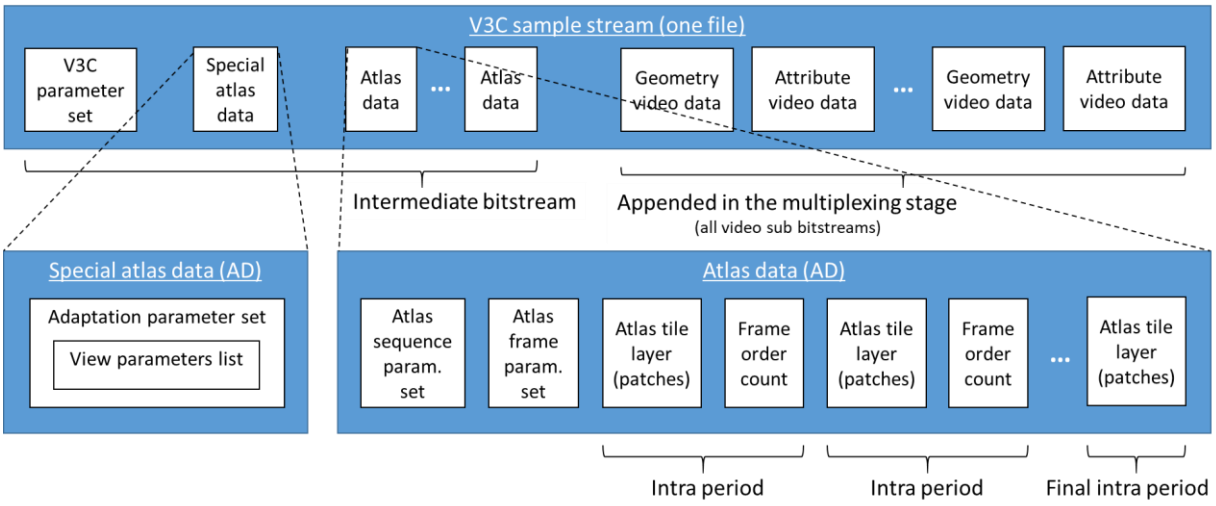

Figure 16: V3C sample stream with MIV extensions.

5 Description of the rendering processes

The TMIV decoder follows the MIV decoding process described in the working draft [2] including the demultiplexing & decoding order, bitstream parsing, video decoding, block to patch map decoding, and resulting conformance points. This section describes the non-normative renderer (Figure 17) starting from the conformance points. This includes the following stages:

- Entity filtering (optional stage),
- Patch culling to speed up the rendering,
- Pruned view reconstruction,
- View synthesis,
- Inpainting,
- Viewing space handling.

The output of the TMIV decoder is a *perspective viewport* or an *omnidirectional view* according to a desired viewing pose, enabling motion parallax cues within a limited space. The rendered

output is provided in luma and chroma 4:2:0 format with 10 bits for attribute component and 16 bits for geometry component. It can in principle be displayed on either head mounted display (HMD) or on regular 2D monitor with tracking system feeding the updated *viewing position* and *orientation* back to the renderer for the next target view. More details on the coordinate systems, projections, and camera extrinsics can be found in Annex B.

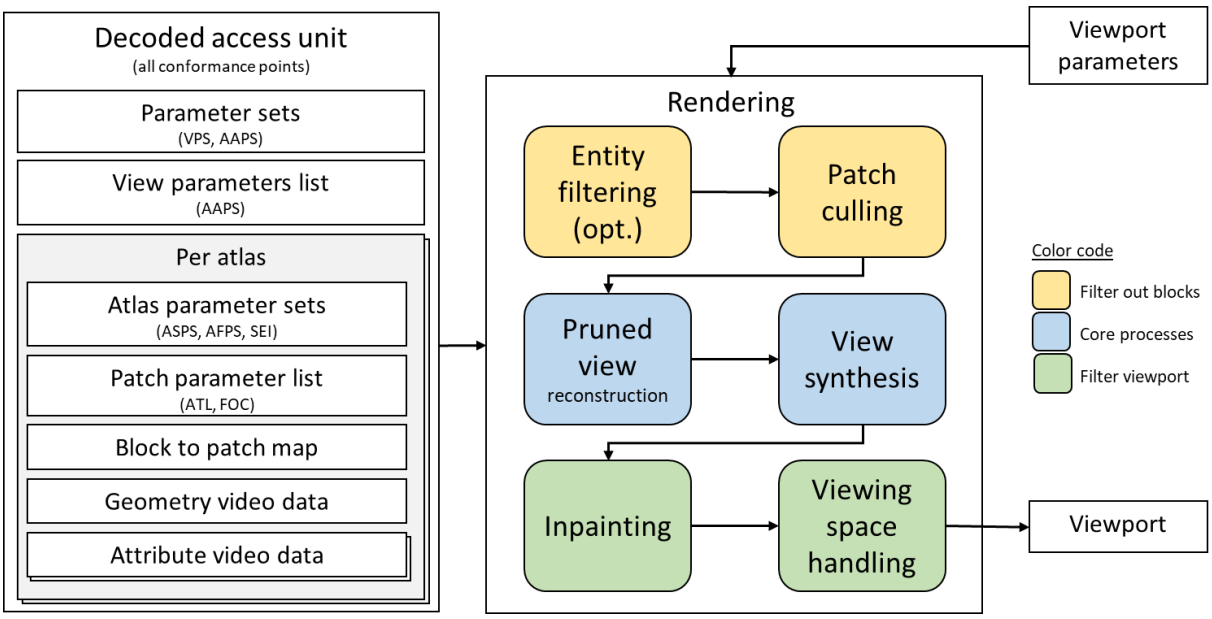

Figure 17: Process flow for the TMIV renderer.

5.1 Entity filtering

The entity filtering is an optional stage that is invoked to select a subset of entities for rendering, by filtering out blocks in the block to patch map that correspond to other entities. A possible usecase is an application that chooses to render foreground objects only, and thus all patches that belong to background objects are excluded. The block to patch map per atlas is filtered as specified in Annex G of the specification.

5.2 Patch culling

The patch culler filters out blocks from the block to patch map to *cull* patches which have no overlap with the target view based on the viewing position and the orientation. The purpose is to reduce the computational cost of the view synthesis. The culling operation follows the same order as the patch creation to be able to filter the block to patch map patch-by-patch.

For each patch, the four corners of the patch are reprojected to the target view by using both minimum and maximum geometry values of view which the patch belongs to. When area enclosed by the eight reprojected points ($P_i'(x, y), i = 0, 1, \dots, 7$) has no overlap with the target viewport, the patch is culled.

The patch map is updated as illustrated in Figure 18. If the patch is culled, the corresponding atlas's samples are set to unusedPatchId n (n=65535 for 16 bits occupancy map).

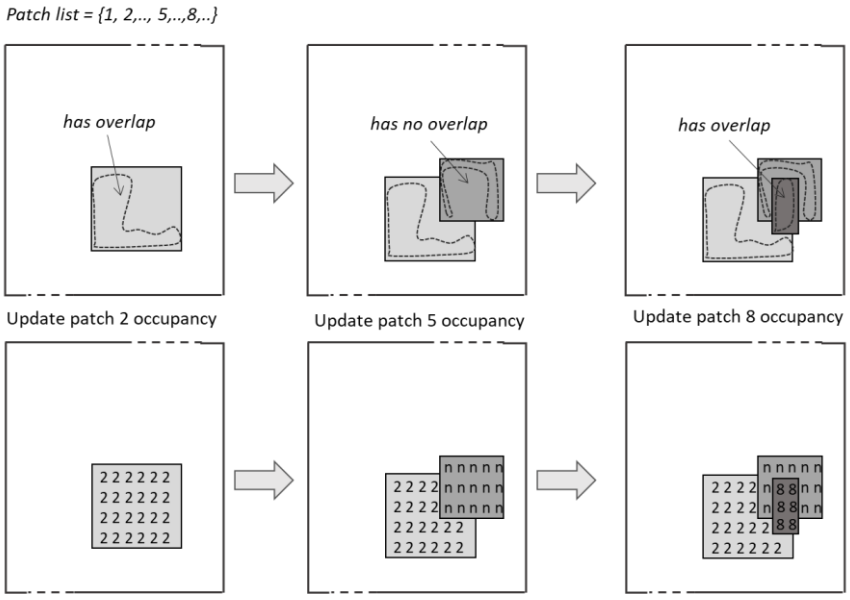

Figure 18. Occupancy map update in an ordered manner.

5.3 Pruned view reconstruction

The reconstruction of the pruned views is an operation opposite to video data generation performed in the encoder (section 4.13). All the (non-culled) patches from atlases (both geometry and attributes) are copied to images which correspond to each source view.

Pruned view reconstruction is presented in Figure 19: patches 2, 3, 5, 7 and 8 are copied to proper position in proper views, based on their position (patch_pos_in_view_x, patch_pos_in_view_y) and the view they belong to (ViewId). Except for basic views, significant part of most reconstructed views is empty.

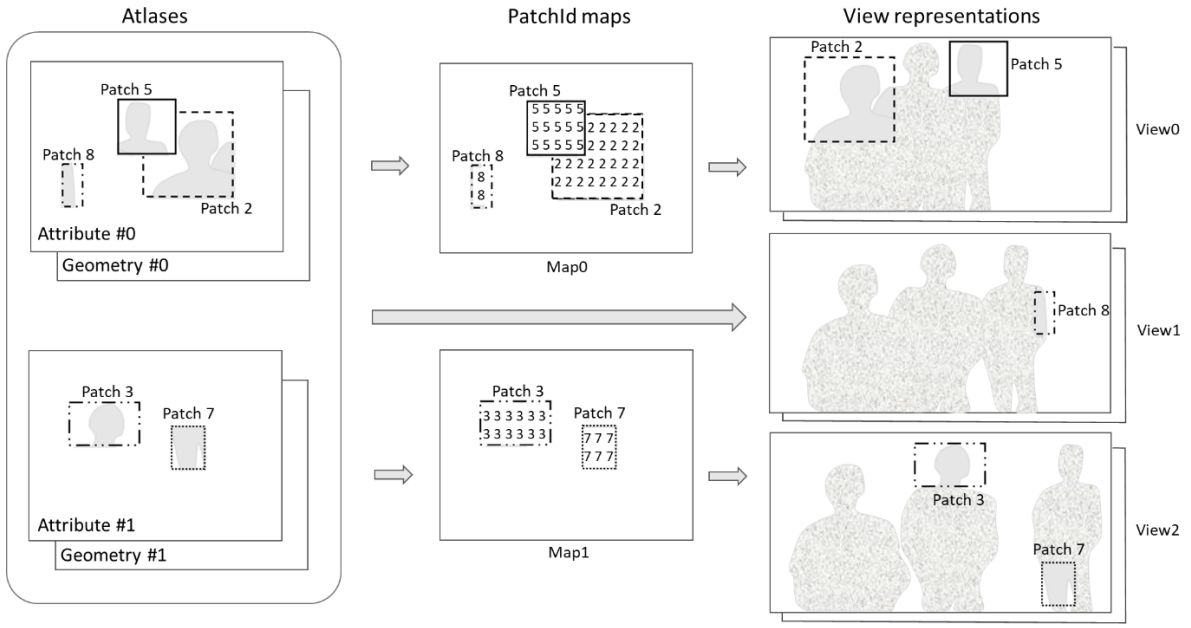

Figure 19: Pruned view reconstruction.

5.4 View synthesis

The TMIV proposes two alternatives for the synthesis. The first one is RVS-based [4], described in section 5.4.1, the second one is the View Weighting Synthesizer (VWS), described in section 5.4.2.

5.4.1 RVS-based synthesizer

5.4.1.1 Overview

1. Generic reprojection of image points,
    a. Unprojection image to scene coordinates (using intrinsics source camera parameters),
    b. Changing the frame of reference from the source to the target camera by a combined rotation and translation (using extrinsics camera parameters),
    c. Projecting the scene coordinates to image coordinates (using target intrinsics camera parameters).
2. Rasterizing triangles,
    a. Discarding inverted triangles,
    b. Creating a clipped bounding box,
    c. Barycentric interpolation of attribute and geometry values,
3. Blending views/pixels.

While RVS was designed to render full views, the Synthesizer works with arbitrary vertex descriptor lists, vertex attribute lists, and triangle descriptor lists (which is very much like OpenGL). The view blending is per pixel and independent of the rendering order. It is thus possible to render any triangle from any patch in any order.

The RVS-based synthesizer may synthesize directly from atlases, thus for this synthesizer there is no necessity of pruned view reconstruction (section 5.3).

5.4.1.2 Rendering from atlases

As part of the decoder (primary purpose) the renderer takes as input:

- Multiple 10 bits attribute atlases and 10 bits geometry atlases (normalized disparities),
- Block to patch map per atlas,
- Parameters including an atlas parameters list and a camera parameters list,
- Target camera parameters for a *perspective viewport* or an *omnidirectional view*.

The output of the renderer is a single view (viewport or omnidirectional) with 10 bits attribute and 10 bits geometry components.

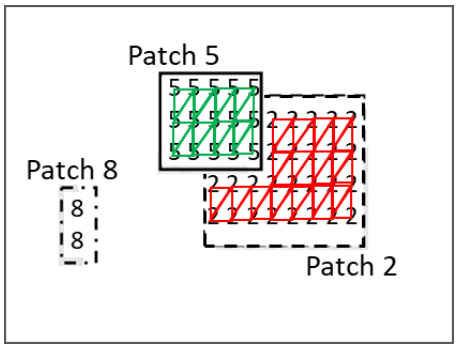

Figure 20: Creating a mesh from an atlas. Triangles between pixels from Patch 5 and 2 are omitted. Note that Patch 8 is not drawn because no triangle can be formed. Unused pixels are skipped too.

The process is to build a mesh (Figure 20) from each of the atlases:

- The **vertex descriptor list** is formed pixel-by-pixel:
  - Skip or write dummy values for unoccupied pixels (occupancy value 0xFFFF),
  - Looking up the atlas parameters list using the PatchId in the occupancy map,
  - Looking up the camera parameters list using the ViewId in atlas parameters list,
  - Calculating the position of the vertex in the view.
  - Reprojecting from the source view to the target view.
- The **vertex attribute list** is simply the texture values converted to YC_BC_R 4:4:4.
- The **triangle descriptor list** is formed by:
  - For each pixel consider two triangles [/]
  - Add the triangle when all vertices have the same PatchId.

This mesh is then rasterized using barycentric interpolation of attribute and geometry. Multiple atlases will be utilized to render from directly in order to have an efficient pipeline for mesh generation and rasterization operations.

5.4.1.3 Pixel blending

The blended value of a pixel component is the weighted sum over all pixel contributions. This choice enables pixel blending in arbitrary order. The weight of a contributing pixel is determined by multiplying three exponential functions with configurable parameters (Table 2).

$$I_{blend} = \sum_i w(\gamma_i, d_i, s_i) I_i$$
$$w\colon (\gamma, d, s) \rightarrow e^{-c_\gamma \gamma + c_d d - c_s s}$$

The weighted sums are normalized by the geometry weight to reduce the required internal precision. All three inputs (ray angle, depth and stretching) are computed in the reprojection process.

Table 2: Description of the blending process.

Input	**Description**	**Purpose**
RayAngle γ	The angle [rad] between the ray from the input camera and the ray from the	Prefer nearby views over views

	target camera.	further away (soft view selection).
Reciprocal geometry d	The reciprocal of the geometry value in the target view [diopter].	Prefer foreground over background (geometry ordering).
Stretching s	The unclipped area of the triangle in the target view relative to the source view.	Penalize triangles that stretch between foreground and background objects.

5.4.2 View weighting synthesizer

5.4.2.1 Overview

The view weighting synthesizer (VWS) relies on the following pipeline:

- Visibility: this step aims at generating a geometry map for the target viewport. First a warped geometry map is generated for each input view, by unprojecting/reprojecting pixels from this view towards the target view. It uses splat-based rasterization [7] instead of triangulation. From the warped geometry maps, a single geometry map is generated, namely the visibility map. This selection process is based on a pixel-wise majority voting process which takes into account the weight of each view, described in section 5.4.2.2. Finally, the visibility map is cleaned out using a post median filtering to remove outliers.

- Shading: this step aims at computing the target viewport color. Each input view's pixel is blended into the target viewport with a contribution/weight taking into account its consistency with the visibility map and the weight of the view it belongs to. Input contours are detected and discarded from the shading stage to avoid ghosting.

5.4.2.2 Weighting strategy

The visibility and shading steps rely on the notion of view weighting. For each input view a weight is computed as:

- A function of the distance between the view position and the target viewport position in the case of tridimensional rigs,

- A function of the distance between the target viewport position and the view forward axis for linear or planar rigs.

To check for the tridimensionality, a test on the singularity of the covariance matrix of the view positions is performed. The contribution of each pixel in the visibility and shading pass is thus weighted by the contribution of its associated view.

However, when dealing with pruned input views, this information is incomplete and an additional step which makes use of the pruning information as defined in section 4.8.1 is performed to recover proper view weight information.

The weight of each non-pruned pixel is updated at the synthesis stage to take into account that it could "represent" other pruned pixels in the descendant hierarchy of the pruning graph (cf. Figure 21). To correctly assess the weight of a non-pruned pixel, the following procedure is applied. Let's consider a non-pruned pixel p of a view associated to a node N of the pruning graph. Let's call $w_P = w_N$ its initial weight (which only depends on the "distance" from the view it belongs to, to the view being synthesized). Then this weight is updated as follows:

1. If the pixel p reprojects into one of the pruned pixels belonging to child views (with respect to the view p belongs to) then its weight is accumulated with the weight w_O of this "child" view (which only depends on the "distance" from this child view to the view being synthesized) by $w_P := w_P + w_O$ and the process is recursively repeated to the grandchildren.

2. If the pixel p does not reproject into one of its child views, then the previous rule is extended recursively to the grandchildren.

3. If the pixel p reprojects into one of its child views at an unpruned pixel then its weight is let unchanged and no more inspection of the graph is performed toward grandchildren.

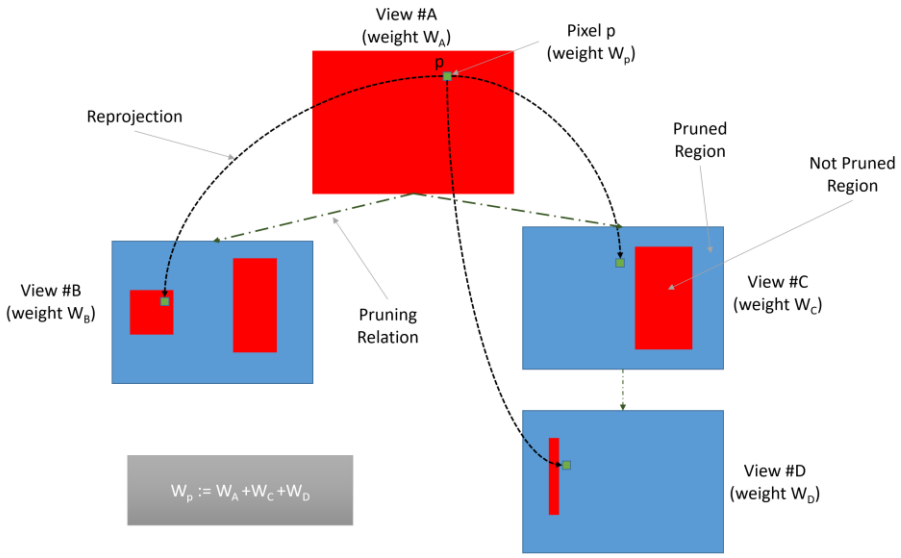

Figure 21: graph-based pruning: weight recovery procedure.

5.4.2.3 Parameters

The parameters of VWS are presented in Table 3.

Table 3: parameters of the view weighting synthesizer.

Parameter	**Type**	**Description**
angularScaling	float	Drives the splat size at the warping stage.
minimalWeight	float	Allows for splat degeneracy test at the warping stage.
stretchFactor	float	Limits the splat max size at the warping stage.

overloadFactor	float	Geometry selection parameter at the selection stage.
filteringPass	int	Number of median filtering pass to apply to the visibility map.
blendingFactor	float	Used to control the blending at the shading stage.

5.5 Inpainting

In order to fill holes in the virtual view, a 2-ways inpainter is used. For each empty pixel with no information, two neighbors are being searched: the nearest non-empty pixel at the left and at the right. The color of the inpainted pixel is a weighted average of colors of the left and the right neighbor, weighted by the distances to these pixels. In the case of significant difference between geometry value of both neighbors, the attribute of the neighbor with further geometry is copied instead of using the weighted average.

However, horizontal inpainting of the virtual view would cause appearance of unnaturally-oriented lines in the case of projecting ERP images to perspective views. Therefore, for ERP images an additional step of changing projection type is performed, and the search of the nearest points is performed within transverse ERP images (transverse equirectangular projection – the Cassini projection [5]). In equirectangular projection, a sphere is mapped onto a cylinder that is tangential to points on a sphere having the latitude equal to 0 degree (Figure 22a). In transverse projection, the cylinder on which the sphere is mapped is rotated by 90 degrees; it is tangential to points that have longitude equal to 0 degree (Figure 22b). It changes the properties of the equirectangular projection in such a way that the search for the nearest projected points can be performed only on the rows of the image.

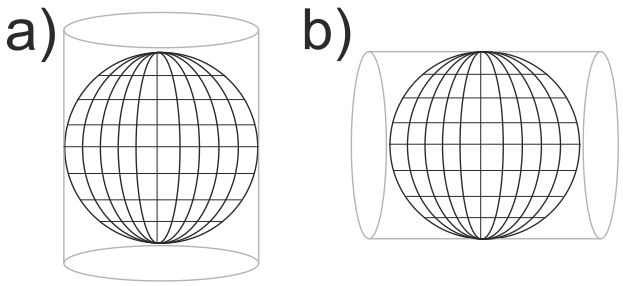

Figure 22: Cylinders used in the projection of a sphere on a flat image in a) equirectangular projection and b) transverse equirectangular projection.

A fast approximate reprojection of equirectangular image to transverse equirectangular image is used. In a first step, the length of all rows in an equirectangular image is changed to correspond to the circumference of the corresponding circle on a sphere (Figure 23a). In a second step, all columns of such image are expanded (Figure 23b), to be of the same length (Figure 23c).

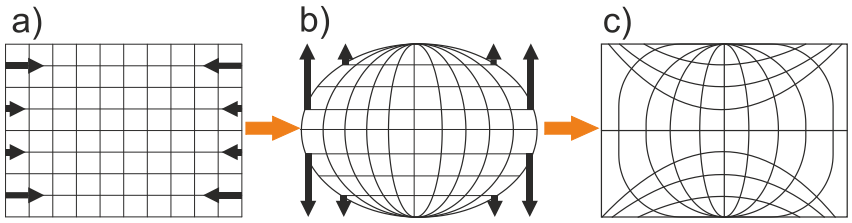

Figure 23: Fast reprojection of an equirectangular image (a) to transverse equirectangular image (c). Black arrows show direction of change of size of respective rows and columns of images.

5.6 Viewing space handling

The viewing space controller is in charge of applying to the viewport a smooth fade out to black according to an internal fading index computed in the decoder part in the viewing space controller (value 0 means no fade). This module computes this index from the viewport current position and orientation and from metadata related to the geometrical dimension of the viewing space and viewing direction constraints. The dimension of the viewing space is defined by a flag (*es_primitive_operation_flag)* through two operation alternatives which are either Constructed Solid Geometry (CSG) or interpolation. The interpolation mode makes use of metadata which lists in an ordered way the position and orientation of primitive cardinal shapes (cuboid, spheroid, half space). The CSG operation makes use of the elementary shapes which are themselves defined from primitive shapes either by CSG or interpolation. For all these modes, it is possible to compute a signed distance *SD(p)* which is zero at the frontier of the related shape, negative inside and positive outside, from which a positional fading index can be computed as follows:

```
positional fading index( p ) = clamp(
    (SD(p) + es_guard_band_size) / es_guard_band_size, 0, 1)
```

where *p* is the position of the viewport, and *es_guard_band_size* is the value of the signed distance from which the fading should start, and *clamp(a, min, max)* is the clamping function of a value a on the [min, max] interval. This first index should be combined multiplicatively by two orientational fading indexes related to the current viewport converted from quaternion to yaw and pitch respectively. For example, the direction fading index for the yaw is computed as follows:

```
yaw fading index( p ) = clamp(
    (abs(yaw - primitive_shape_viewing_direction_yaw_center) -
        primitive_shape_viewing_direction_yaw_range +
        es_guard_band_direction_size) / es_guard_band_direction_size, 0, 1)
```

where primitive_shape_viewing_direction_yaw_center is the yaw converted value from the primitive viewing direction center quaternion and yaw is the yaw value of the viewport.

The viewing direction at a given position of the viewport is obtained from the set of individual values. In Figure 24, two modes of viewing space are illustrated, as well as viewing direction with the arrows.

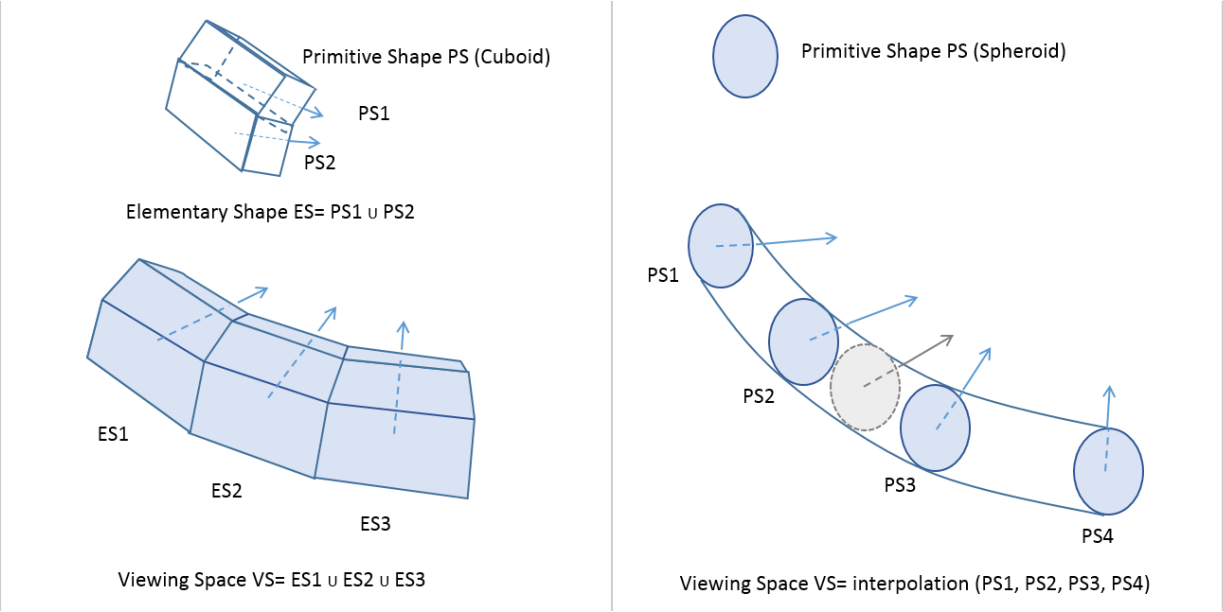

Figure 24: Illustration of VS creation with additive CSG (left) and interpolation (right).

References

[1] *Call for Proposals on 3DoF+ Visual*, ISO/IEC JTC1/SC29/WG11 MPEG/N18145, Jan. 2019, Marrakesh, Morocco.

[2] J. Boyce, R. Doré, V. Kumar Malamal Vadakital (Eds.), *Working Draft 5 of Immersive Video*, ISO/IEC JTC1/SC29/WG11 MPEG/N19212, Apr. 2020, Online / Alpbach, Austria.

[3] J. Jung, B. Kroon, J. Boyce, *Common Test Conditions for Immersive Video*, ISO/IEC JTC1/SC29/WG11 MPEG/N19214, Apr. 2020, Online / Alpbach, Austria.

[4] *Reference View Synthesizer (RVS) manual*, ISO/IEC JTC1/SC29/WG11 MPEG/N18068, Oct. 2018, Macao, China.

[5] J. Snyder, P. Voxland, *An album of map projections*, US Government Printing Office, Washington, 1989.

[6] J. Jylänki, *A thousand ways to pack the bin - a practical approach to two-dimensional rectangle bin packing*, 2010.

[7] Point-based graphics, 2007, Elsevier, edited by Markus Gross and Hanspeter Pfister.

Annex A. Reference software

A.1. Availability and use

The reference software (TMIV-SW) including manual is publicly available on the Gitlab server[2].

A.2. Software coordination

In case of any related inquiries, please contact one of the software coordinators:

- Julien Fleureau, julien.fleureau@interdigital.com
- Vinod Malamalvadakital, vinod.malamalvadakital@nokia.com
- Bart Kroon, bart.kroon@philips.com
- Bin Wang (王彬), 3130100819@zju.edu.cn

[2] https://gitlab.com/mpeg-i-visual/tmiv/

Annex B. Coordinate systems, projections, and camera extrinsics

This section summarizes the coordinate conversions of the *hypothetical view renderer* (HVR) and the conventions that are applied in TMIV.

B.1. OMAF coordinate system

Although the MIV specification is agnostic to the coordinate system of the bitstream, the TMIV world coordinate system is that of MPEG-I OMAF[3] as shown in Figure 25. Coordinate axis system VUI parameters are printed by the TMIV decoder but ignored by the TMIV renderer.

- $\hat{x}_{\text{world}}$ points forward (the reference direction for a viewer),
- $\hat{y}_{\text{world}}$ points left,
- $\hat{z}_{\text{world}}$ points up,

Hereby $\hat{x}, \hat{y}, \hat{z}$ is the notation for Cartesian unit vectors such that $\boldsymbol{x} = (x, y, z)^T = x\hat{x} + y\hat{y} + z\hat{z}$. For an untransformed camera the origin is the cardinal point.

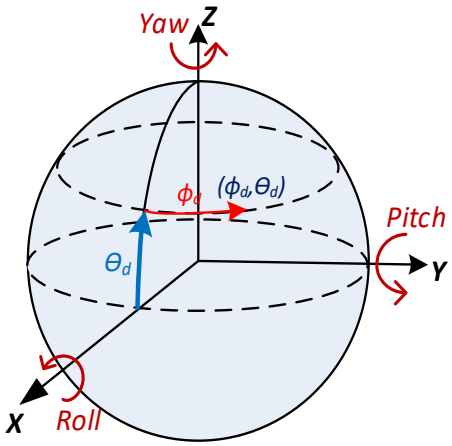

Figure 25: OMAF coordinate system illustrating the directions of positional and rotational units.

The definition of image coordinates is:

- The top-left image corner is (0, 0),
- The top-left pixel center is at (½, ½),
- $\hat{x}_{\text{image}}$ points right,
- $\hat{y}_{\text{image}}$ points down.

Image positions are notated as $\boldsymbol{u} = (u, v)^T = u\hat{x}_{\text{image}} + v\hat{y}_{\text{image}}$.

B.2. Perspective projection

Perspective projection requires an intrinsic matrix where all variables are in pixel units:

[3] https://mpeg.chiariglione.org/standards/mpeg-i/omnidirectional-media-format

$$M = \begin{bmatrix} f_x & & p_x \\ & f_y & p_y \\ & & 1 \end{bmatrix} \tag{1}$$

Projection:

Taking into account the change of coordinate system, the projection equation is

$$\boldsymbol{x}_{\text{image}} = \begin{bmatrix} x_{\text{image}} \\ y_{\text{image}} \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} - x_{\text{world}}^{-1} \begin{bmatrix} f_x y_{\text{world}} \\ f_y z_{\text{world}} \end{bmatrix}, \tag{2}$$

where $\boldsymbol{x}_{\text{image}}$ is the image position in pixel units.

Unprojection:

The matching projection equation is

$$\boldsymbol{x}_{\text{world}} = \begin{bmatrix} x_{\text{world}} \\ y_{\text{world}} \\ z_{\text{world}} \end{bmatrix} = d \begin{bmatrix} 1 \\ f_x^{-1}(p_x - x_{\text{image}}) \\ f_y^{-1}(p_y - y_{\text{image}}) \end{bmatrix}, \tag{3}$$

where d is geometry in meters and $\boldsymbol{x}_{\text{world}}$ is the world position in meters. The geometry is typically stored as normalized disparities based on a configurable geometry range, however in above equation d is a length in meters.

B.3. **Equirectangular projection**

For equirectangular projection the image is mapped on a horizontal angular range $[\phi_1, \phi_2]$ and vertical angular angle $[\theta_1, \theta_2]$ as specified in the JSON content metadata file.

Unprojection:

For an image size $w \times h$, the spherical coordinates are:

$$\phi = \phi_2 + (\phi_1 - \phi_2)\frac{x_{image}}{w}, \tag{4}$$

$$\theta = \theta_2 + (\theta_1 - \theta_2)\frac{y_{image}}{h}. \tag{5}$$

The *ray direction* is:

$$\hat{r} = \begin{bmatrix} \cos\phi \cos\theta \\ \sin\phi \cos\theta \\ \sin\theta \end{bmatrix} \tag{6}$$

and the world position is:

$$\boldsymbol{x}_{\text{world}} = r\hat{r}, \tag{7}$$

Whereby r is the *ray length* which is the equivalent of geometry d for perspective projection. Please note that also ray length is stored as normalized disparities based on a configurable ray length range, however in the above equation r is a real length.

Projection**:**

The ray length and ray direction are trivially determined as

$$r = |\boldsymbol{x}_{\text{world}}|, \tag{8}$$
$$\hat{r} = \boldsymbol{x}_{\text{world}}/r, \tag{9}$$

making use of the fact that valid ray lengths are $r > 0$.

Finally, spherical angles are then estimated from $\hat{r}$:

$$\phi = \text{atan2}(\langle \hat{r}, \hat{y} \rangle, \langle \hat{r}, \hat{x} \rangle) \tag{10}$$
$$\theta = \sin^{-1}\langle \hat{r}, \hat{z} \rangle \tag{11}$$

with atan2 the full circle extension of atan[4]. Then the image position is

$$x_{image} = w(\phi - \phi_2)/(\phi_1 - \phi_2) \tag{12}$$
$$y_{image} = h(\theta - \theta_2)/(\theta_1 - \theta_2) \tag{13}$$

The only difference between equirectangular projection and other omnidirectional projections is the mapping between spherical coordinates and image coordinates.

B.4. **Camera extrinsics**

The MIV specification as well as TMIV use position vectors ($\boldsymbol{t}$) and unit quaternions[5] (q) to represent camera extrinsics.

The sequence configuration files and *pose traces* use Euler angles which are converted directly upon loading[6]. *Pose traces* are comma-separated value files with the same six columns as the CTC tables and JSON metadata files: X, Y, Z, Yaw, Pitch, Roll.

The two rotations and two translations to transform a point ($\boldsymbol{x}$) from an input camera to a virtual (output) camera are combined into a single affine transformation (f):

$$\text{f: } \boldsymbol{x} \rightarrow q\boldsymbol{x}q^* + \boldsymbol{t} \tag{15}$$

Where $q = q^*_{output} q_{input}$ and $\boldsymbol{t} = q_{output}(\boldsymbol{t}_{input} - \boldsymbol{t}_{output})q^*_{output}$.

---

[4] https://en.wikipedia.org/wiki/Atan2
[5] https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
[6] https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles