ISO/IEC JTC 1/SC 29/WG 4 N0576

ISO/IEC JTC 1/SC 29/WG 04

MPEG Video Coding

Convenorship: CN

Document type:	Output document
Title:	Test model 22 for MPEG immersive video
Status:	Approved
Date of document:	2024-12-06
Source:	ISO/IEC JTC 1/SC 29/WG 04
Expected action:	None
Action due date:	None
No. of pages:	61 (without cover page)
Email of Convenor:	yul@zju.edu.cn
Committee URL:	https://sd.iso.org/documents/ui/#!/browse/iso/iso-iec-jtc-1/iso-iec-jtc-1-sc-29/iso-iec-jtc-1-sc-29-wg-4

Editors' integration notes

TMIV 22 – N0576:

- m69945 Implementation of MIV Simple MPI profile in TMIV
- m70068 Selective patch margin enabling for rendering quality increase

TMIV 20 – N0514:

- m67323 Background / non-background separation
- m67514 Higher bit-depth geometry coding using colorized depth representation
- m67984 Basic view selection based on valid region

TMIV 18 – N0409:

- m65411 Effective-Information-Based cluster merging and better cluster splitting
- m64709 Implementation of MIV DSDE sub-profile in TMIV

TMIV 17 – N0376:

- m64165 Patch margin signaling for low-bitrate rendering improvement
- m64510 Encoder-side filtration of patch margins
- m63750 Patch constellations

TMIV 16 – N0347:

- m63397 Chroma dynamic range modification
- m63503 Atlas flickering removal
- m63109 Renderer-side edge processing for subjective quality improvement
- m63213 Encoder-side Effective Information (ESEI) Based optimization of multi-view atlas generation

TMIV 15 – N0271:

- m59442: Geometry packing
- m60668: Inhomogeneous view selection for omnidirectional content
- m60846: Wider depth dynamic range with occupancy error correction

TMIV 14 – N0242:

- m58783: Piecewise linear scaling of geometry atlas
- m59213: Server-side inpainter with improved FOV calculation
- m59304: Prioritize packing for the server-side inpainter
- m59517: On increasing subjective quality for low bitrates

TMIV 11 – N0142:

- m58035: Second-pass pruner using uniform sampling

TMIV 10 – N0112:

- m57419: Piecewise linear scaling of geometry atlas

TMIV 9 – N0084:

- m56827: Frame packing

TMIV 8 – N0050:

- m55709: New proposed MPI format for MIV
- m55736: Server-side inpainting

TMIV 7 – N0005:

- m54874: Geometry absent
- m54891: Second-pass pruner

- m54892: Color-based patch analysis
- m54491/m55343: Frame packing
- m54893: Adaptive texture-based pruning
- m54894: Color-correction
- m55089: MPI coding

TMIV 6 – WG11 N19483:

- WG11 m53042: Views enabled per atlas
- WG11 m53348: Decoded atlas data hash SEI message
- WG11 m54145: Basic views allocation
- WG11 m54152: Occupancy coding
- WG11 m54176: Geometry scaling
- WG11 m54177: Texture pruning
- WG11 m54362: V3C_CAD
- WG11 m54391: Patch merging
- WG11 m54417: Patches with constant depth
- WG11 m54489: PTL decoder instantiations constraint
- WG11 m54491: Packed independent regions SEI message
- WG11 m54754: Basic views allocation and pruner modifications

TMIV 5 – WG11 N19213:

- WG11 m52953: Restructuring of this document
- WG11 m52994: Proposed simplifications of MIV
- WG11 m53506: HEVC Multiplexing
- WG11 m53701: Additional patch dilation in temporal patch redundancy removal

TMIV 4 – WG11 N19002:

- WG11 m51604: Spatio-temporal patch redundancy removal
- WG11 m52320: Culling for viewport rendering
- WG11 m52350: MIV HLS bitstream codec
- WG11 m52365: Depth-map scaling
- WG11 m52413: Synthesizer
- WG11 m52414: Graph-based pruning
- WG11 m52475: Object-based implementation

TMIV 3 – WG11 N18795:

- WG11 m49958: Grouping
- WG11 m49962: Pruning
- WG11 m50949: Object-based immersive coding
- WG11 m51439: Depth occupancy coding
- WG11 m51487: Viewing space

TMIV 2 – WG11 N18577:

- No tool adoption
- Inclusion of operating modes

TMIV 1 – WG11 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 04 MPEG VIDEO CODING

ISO/IEC JTC 1/SC 29/WG 04 N 0576
November 2024, Kemer, TR

Title	**Test model 22 for MPEG immersive video**
Source	**WG 04, MPEG Video Coding**
Status	**Approved**
Serial number	**24569**
Editors	**Adrian Dziembowski, Gwangsoon Lee**

Abstract

The MPEG Video Coding working group has established the 22nd test model for MPEG immersive video during the 148th MPEG meeting (October 2024) after evaluating input 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 the new part ISO/IEC 23090-12 *Immersive Video* (MIV).

At the 129th MPEG meeting the fourth working draft of MIV has been realigned to use ISO/IEC 23090-5 *Video-based Point Cloud Compression* (V-PCC) as a normative reference for terms, definitions, syntax, semantics and decoding processes. At the 130th MPEG meeting this alignment has been completed by restructuring Part 5 in a common specification *Visual Volumetric Video-based 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 the Draft International Standard on MPEG immersive video Ed. 2 [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 MPEG immersive video* [3] provide 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.
Background view	A source view that is a collection of static or rarely-changing parts (e.g., “background”) of a MIV scene.
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	Changing the dynamic range of the geometry data and 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.
Occupancy scaling	Scaling of the occupancy data prior to encoding and reconstructing the nominal resolution occupancy data at the decoder side.
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 semi-basic and 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.
Semi-basic view	A source view that is to be pruned and packed in an atlas as a single patch.
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, semi-basic views, or additional views.

4. Description of encoder processes

4.1 Introduction

4.1.1 High-level description

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:

- Preparation of source material by:
  - Assessing the geometry (depth map) quality, if present, for each source view.
  - Splitting source views in groups.
  - Synthesizing an inpainted background view covering the whole field of view of the source views within each group.
  - Labeling source views as basic views, semi-basic, or additional views.
- Encoding of each group separately (using the associated subset of split source views).
- Formatting of the bitstream (includes a merging substage to combine sub bitstreams of same type produced by each single-group encoder together) which is V3C sample stream with MIV extensions and related SEI messages.
- If packed video is enabled, then geometry video data (GVD), attribute video data (AVD), and occupancy video data (OVD) can be combined into packed video data (PVD) per atlas.
- Encoding video sub bitstreams based on their presence (*i.e.*, the presence of GVD, AVD, OVD or PVD depends on the encoder configuration):
  a. HEVC encoding of video sub bitstreams (each separately) using HM.
  b. VVC encoding of video sub bitstreams (each separately) using VVenC.
- Multiplexing to combine the formatted bitstream with the video sub bitstream into a single MIV-compliant bitstream.

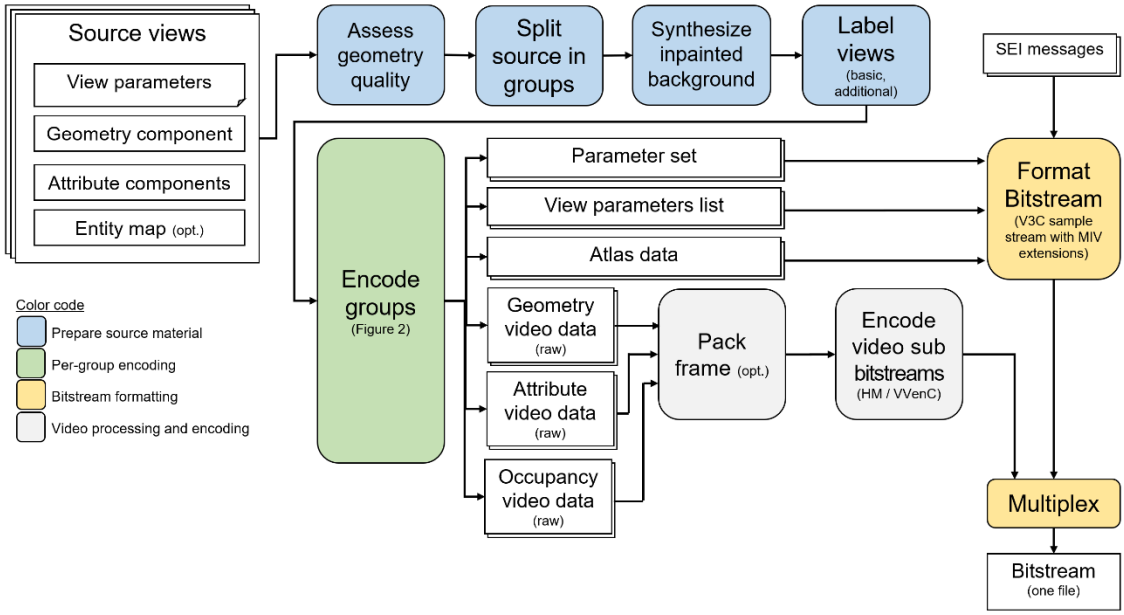

Figure 1: Top-level diagram of the TMIV group-based encoder

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, if present.
6. Scaling of occupancy video data per atlas, if present.

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

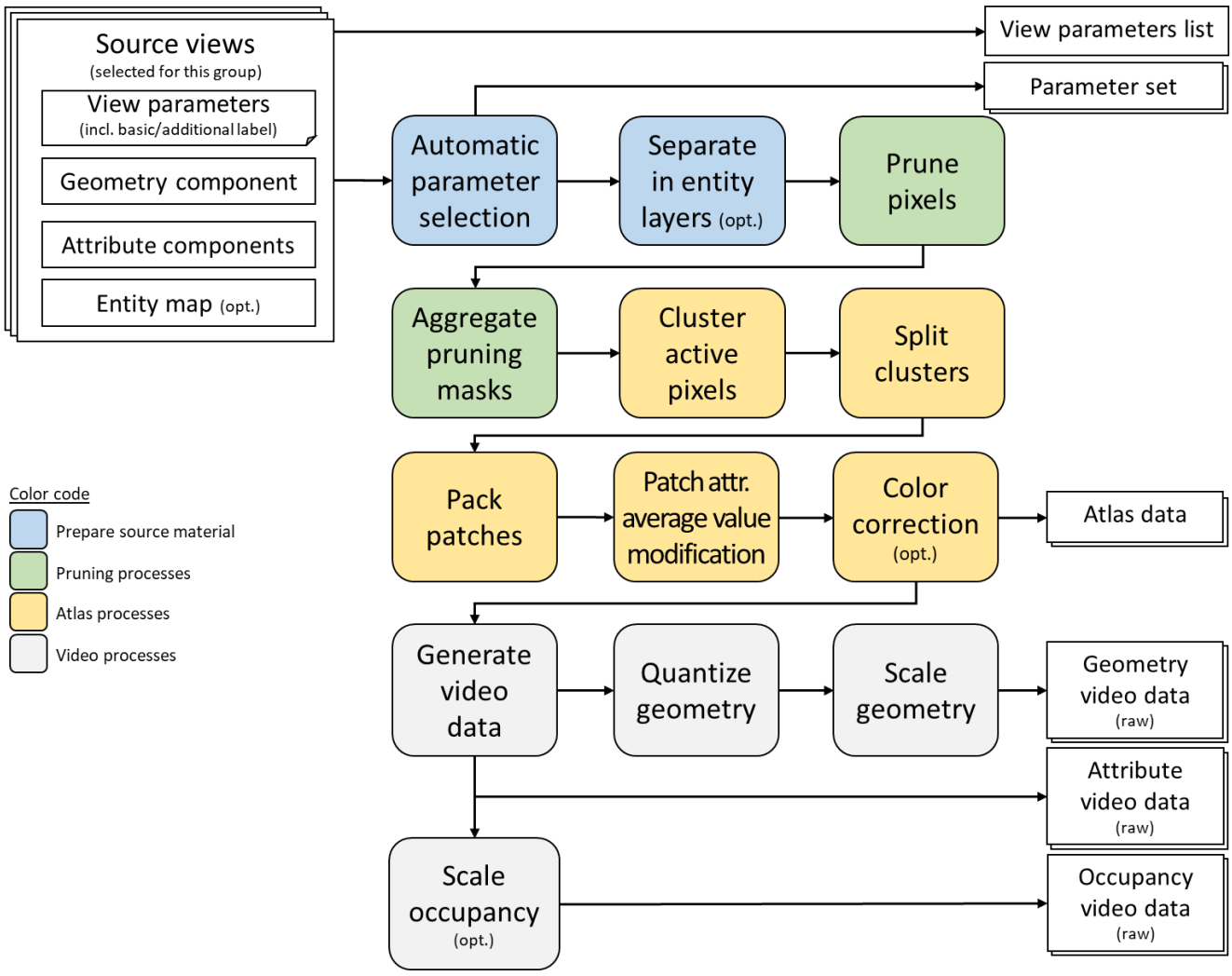

Figure 2: Top-level diagram of the TMIV single-group encoder

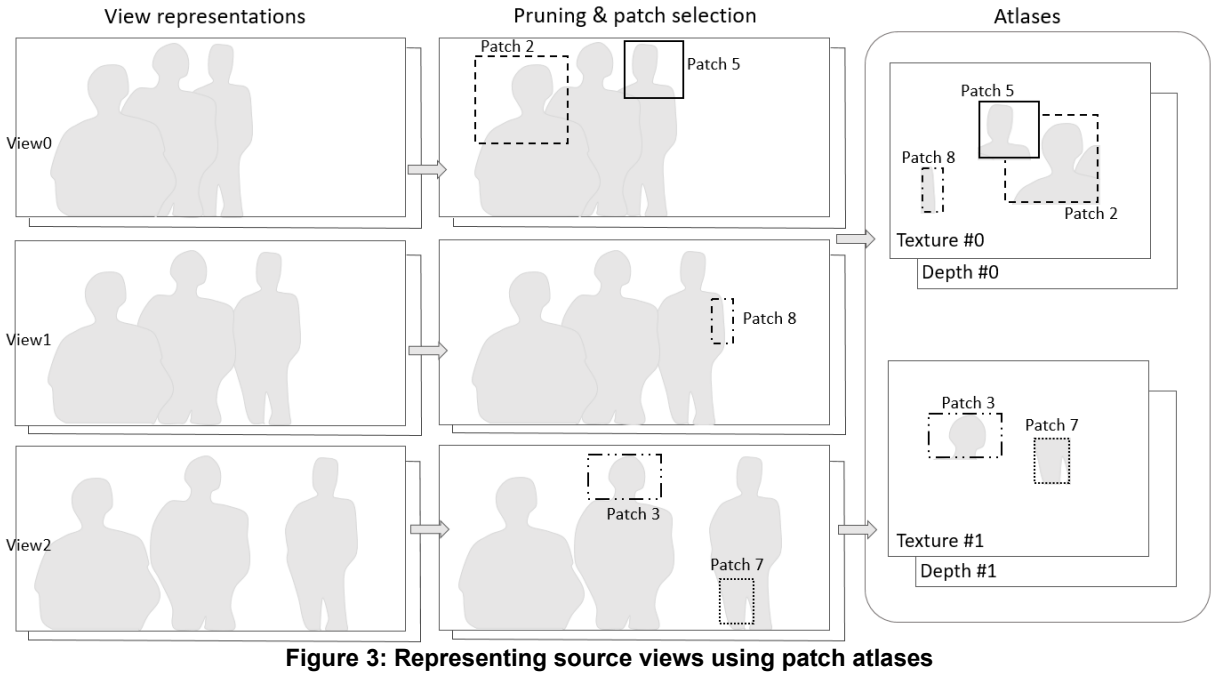

Figure 3: Representing source views using patch atlases

At the 132th MPEG meeting, Multiple Plane Images [10] have been introduced to TMIV supporting an alternative coding mechanism that uses transparency layers.

4.1.2 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.). A source view may have a geometry component with range/invalid sample values. Also, a source view may have texture attribute component. Additional optional attributes per source view are an entity map and a transparency attribute component. The set of components has to be the same for all source views.

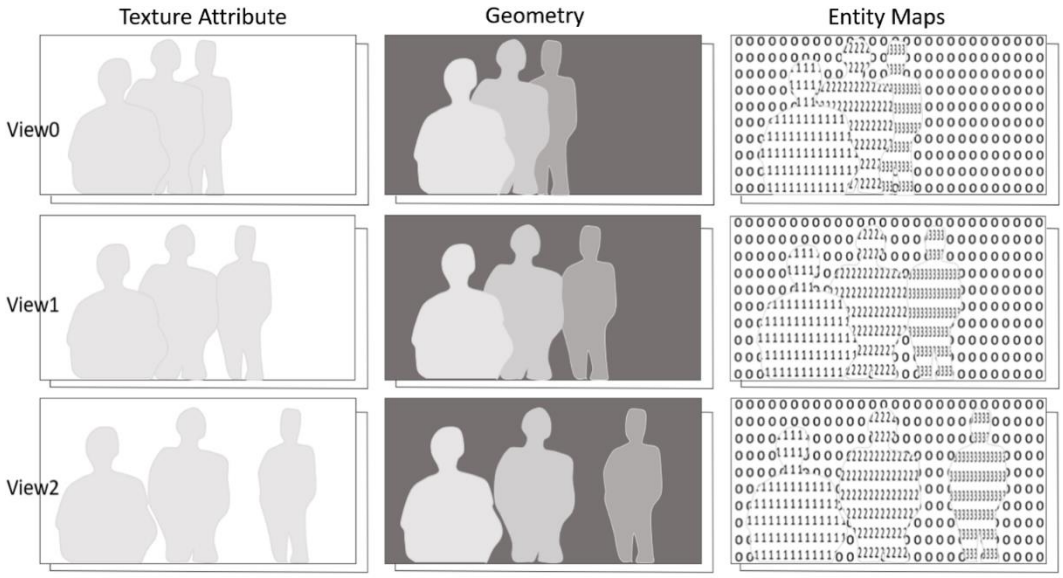

Figure 4: Input source views composed of texture attribute and geometry components, and entity maps

4.1.3 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 common atlas data is present. The view parameter list is sent once, and depth quantization parameters (if present) are updated at each intra frame. For each of the regular atlases, there are sub bitstreams with patch data, geometry video data (if present), attribute video data (if present), occupancy video data (if present), and packed video data (if present). An atlas may be composed of multiple atlas tiles. Atlas and patch parameters include groups and entity ID's respectively[1].

The structure of a V3C bitstream (Figure 5) is as follows:

- The V3C bitstream consists of a V3C unit stream (with carriage out of scope) or a V3C sample stream which is a simple container for a V3C unit stream.

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

    - At the start of the V3C unit stream, the V3C parameter set (VPS) is available in-band or out-of-band. The information in the VPS announces the presence of sub bitstreams, allowing the decoder to initialize sub decoders for all atlas and video sub bitstreams.

    - Each subsequent V3C unit has a payload that contains one or more access units of a sub bitstream. The V3C unit header identifies to which sub bitstream the payload applies.

- The geometry video data (GVD), attribute video data (AVD), and occupancy video data (OVD) V3C units contain video sub bitstreams for a specific atlas component. While the standard is video codec agnostic, for the test model the video sub bitstreams are always HEVC Annex B streams.

- TMIV also supports packed video data (PVD) V3C units that packs various video data types of multiple atlas tiles (per atlas) together.

- The atlas data (AD) V3C unit contains an atlas sub bitstream which is also a network abstraction layer (NAL) unit stream, but instead of video frames there is a NAL unit called atlas tile layer (ATL) that carries a list of patch data units (PDU). Each PDU describes the relation between a patch in an atlas and the same patch in a (hypothetical) source view. The ATL is parameterized using the atlas sequence parameter set (ASPS), atlas adaptation parameter set (AAPS), and atlas frame parameter set (AFPS).

- The common atlas data (CAD) V3C unit also contains an atlas sub bitstream, but the main NAL units are the common atlas sequence parameter set (CASPS) and the common atlas frame (CAF) that contains the view parameter list or updates thereof.

- All sub bitstreams may contain SEI messages and both the CASPS MIV extension and ASPS may contain volumetric usability information (VUI).

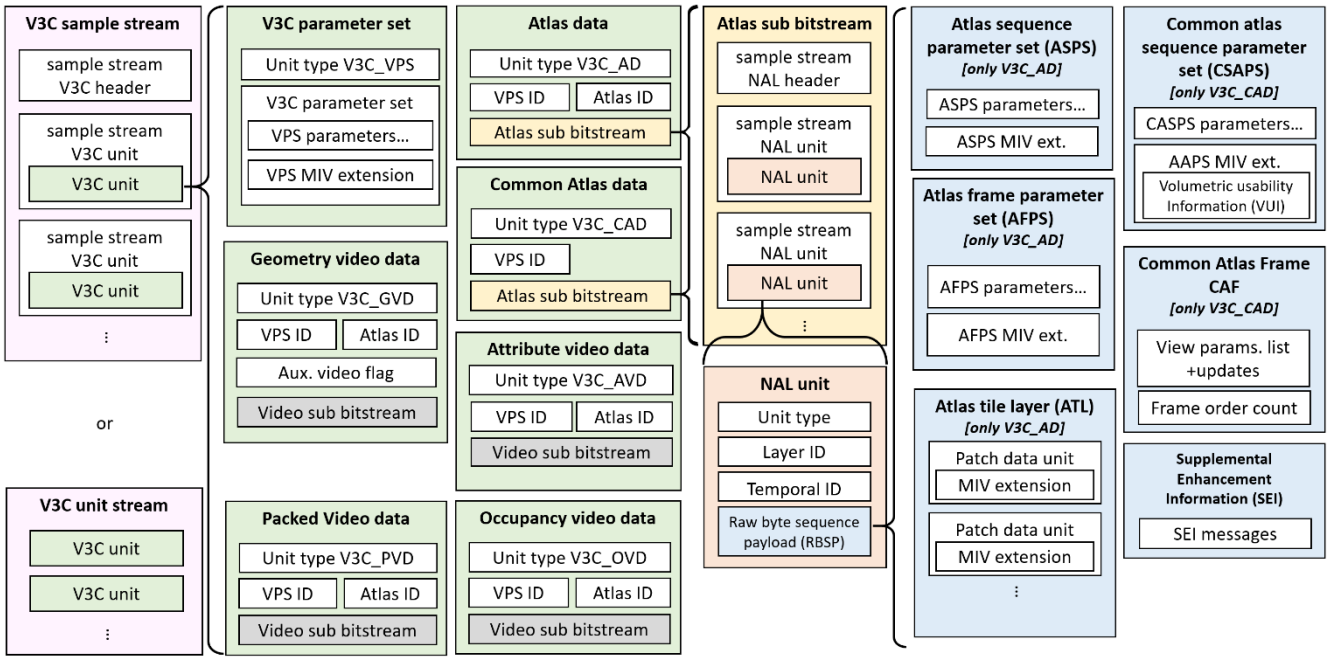

Figure 5: Structure of the V3C bitstream with MIV extensions.
Some aspects of V3C that are not relevant to MIV have been omitted for clarity

4.2 View preparation

4.2.1 Distribution of source views in groups

Source views can be divided into multiple groups. The grouping helps outputting local coherent projections of important regions (*e.g.*, belonging to foreground objects or occluded regions) in the atlases per group as opposed 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.

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.

4.2.2 Synthesis of inpainted background view

The inpainting module creates synthetic texture and geometry data that is hidden from the source views, thereby reducing the missing data problem at the decoder-side. It creates an ERP view with inpainted background data. This view has the following properties:

- It is placed in the center of the camera rig, i.e., the mean of the source camera positions.

- Its field-of-view is the union of source view field-of-views (with some additional margin set in the configuration file, by default the margin is set to 30% of combined field-of-view). This union field-of-view is calculated by projecting a set of 3d points that are at the border of each source view, for the nearest and farthest depth (znear, zfar), to the target which is the inpainted view. The parameter "*segmentsPerEdge"* determines how many points are used per border edge. The final field-of-view union is the bounding rectangle, with the applied margin (parameter "*fieldOfViewMargin"* in the TMIV encoder configuration file).
- Since the quality of the inpainted regions is lower than the original content, the resolution of the view is typically chosen lower than the resolution of the source views, hereby saving on bitrate and pixel-rate. This resolution is configurable.

The following steps are taken:

- A background view is synthesized from all available source views. The 'RVS-based synthesizer', (see Section 5.5.1) is configured to render the background (when available) over the foreground (negative "*depthParameter*"). Figure 6 illustrates this synthesis step: the left image shows synthesis with a positive "*depthParameter*" where foreground is rendered over background. The middle image shows synthesis with a negative "*depthParameter*" where the background is rendered over the foreground. It de-occludes the background region 'A' that is visible from the source views.
- Some pixels of the synthesized background view have no correspondence in the source views, hence their depth values are set to 0. Those pixels are inpainted in a later process. As an intermediate step, remaining foreground pixels are identified and removed as follows:
    - The synthesized depth frame, represented by normalized disparities, is filtered using a box blur with a configurable kernel size. This blurred depth frame is used for comparing with the actual synthesized depth frame. Depth values that are closer indicate relative foreground and depth values that are farther, indicate relative background.
    - The relative foreground pixels are identified by comparison with some configurable threshold. They identify the remaining foreground pixels that occlude the background. Their depth values are set to zero which yields a mask of missing background pixels. These are indicated by region 'B' in the right image of Figure 6.
- The masked texture and depth data are inpainted from neighboring areas. The push-pull inpainter is used for this purpose.

The inpainted background view (identified by a boolean flag) is used for filling missing pixels at the decoder side. This filling process is applied at the patch level or at the view level.

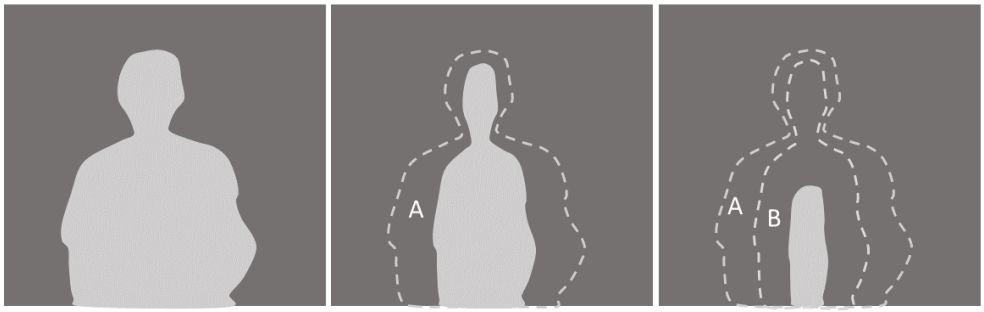

Figure 6: Steps in finding an inpaint mask for synthesizing an inpainted background view

The push-pull inpainter is similar to the method that is used to pad texture patches in V-PCC [12]. The input resolution for the push-pull inpainter is the resolution of the above-described background view that can be lower than the source views.

1. Push: starting from the input resolution, the resolution is repeatedly halved (rounding up) with linear interpolation of texture and depth (with border repeat), until the top of the pyramid is an image of 1 × 1 pixel.
2. Pull: starting with the second-smallest image, when depth is larger than zero, the texture and depth is preserved. Otherwise, texture and depth are averaged over the neighboring samples of the higher layer (with border repeat) that have non-zero depth. When none such samples exist, the zero depth is maintained.
3. The output of the algorithm is the filtered frame at input resolution.

4.2.3 View labeling

The view labeler receives source view parameters for all source views (Figure 1) and based on that each source view is labeled as basic or additional (§4.2.3). A subset of additional views may be later relabeled to semi-basic views. The decision regarding relabeling is taken during the pruning graph creation (§4.3.1.1). There are two modes for view selection, which allows to study the benefit of supplementing complete views with patches.

In the first mode, all source views are output, and they are labeled as basic, semi-basic, or additional views (Figure 7). The encoding result is one or more atlases with complete views and patches taken from semi-basic and additional views.

Figure 7: View selection behavior of the view labeler when semi-basic and additional views are enabled

In the second mode, only basic views are output (Figure 8). The encoding result is one or more atlases with only complete views.

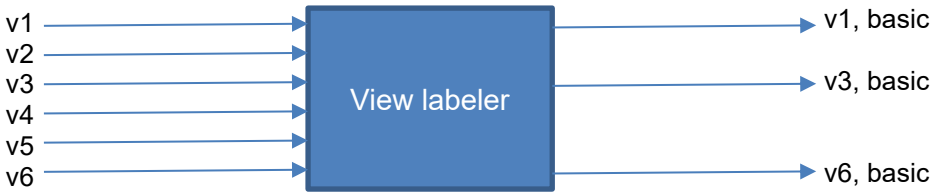

Figure 8: View selection behavior of the view labeler when semi-basic and additional views are not enabled

The labeling of basic views consists of the following steps:

1. Determine the number of basic and semi-basic views,
2. Prepare cost calculation,
3. Select initial basic views,
4. Update the view labels.

The inpainted view is labeled ‘additional’.

4.2.3.1 Determine the number of basic and semi-basic views

In the data processing flow of the test model, the atlas frame size calculation (§4.2.6) is performed after view labeling. Part of the atlas frame size calculation logic is repeated to estimate how many basic views there could be within pixel rate constraints:

1. The number of encoded atlases is assumed to be equal to the configured maximum number of atlases divided by the configured number of groups.

2. The maximum allowed number of atlas samples that is available to the encoder is the product of the number of encoded atlases and the configured maximum number of samples per atlas (M). Note that the number of samples per atlas corresponds to the luma picture size of the texture attribute video data.

3. The maximum number of atlas samples that all basic views together may use (N) is a configurable fraction of the total allowance (T). For instance, when this fraction is 50% and there are two atlases, then all basic views will fit in the first atlas. The total limit of samples from semi-basic views is calculated as ($S = T - N$).

4. The number of basic views is determined by iterating over source views in order of decreasing sample count per source view. While iterating, the total number of samples is counted as well as the number of samples in the first atlas. When there are K atlases, the first, $1 + K$'th, $1 + 2K$'th, etc. source views are assigned to the first atlas. The number of basic views corresponds to the largest number of source views that still fit in terms of the maximum number of samples N and the maximum number of samples per atlas M. The number of semi-basic views is determined analogously, considering the limit S.

Assumptions are:

1. The number of basic and semi-basic views is constrained by the number of atlases (per group) and the luma picture size, but not by the sample rate.

2. The size and aspect ratio of the source views is such that they can be packed efficiently. (It is sufficient to count samples, instead of performing trail packings.)

Finally, the number of basic views is limited to ensure that some source views are either pruned or non-coded. This allows to preserve meaningful objective evaluation on source view positions.

4.2.3.2 Prepare cost calculation

The basic view allocation is based on the partitioning around medoids (PAM) algorithm (k-medoids[2]) with basic views as k medoids among n source views but modified to use a repulsion/attraction cost function.

[2] https://en.wikipedia.org/wiki/K-medoids

The cost function requires a distance metric on source views, using the position of each source view to discriminate source views. The distance matrix is thus:

$$R = [r_{i,j}^2]$$

whereby $r_{i,j}^2$ is the squared distance [m^2] between the source view positions, calculated as:

$$r_{i,j}^2 = \left(r_{i,j}^x\right)^2 + \left(r_{i,j}^y\right)^2 + \left(\alpha \cdot r_{i,j}^z\right)^2$$

where $r_{i,j}^x$, $r_{i,j}^y$, $r_{i,j}^z$ indicate a distance between cameras i and j among three axes, and α is the vertical inhomogenity coefficient, which favors horizontal distances between cameras, penalizing cameras placed at the top or bottom of the camera rig. By default, α coefficient is set to 0.4 for omnidirectional (*i.e.*, ERP) content and 1.0 for perspective content.

The idea of the repulsion/attraction (Figure 9) is that the full configuration of source views is considered. The repulsion of medoids is always stronger than the attraction of medoids to source views: when there is only one medoid the cost is based only on attraction, and when there are multiple medoids, the cost is based only on repulsion. This avoids a parameter to balance the "forces".

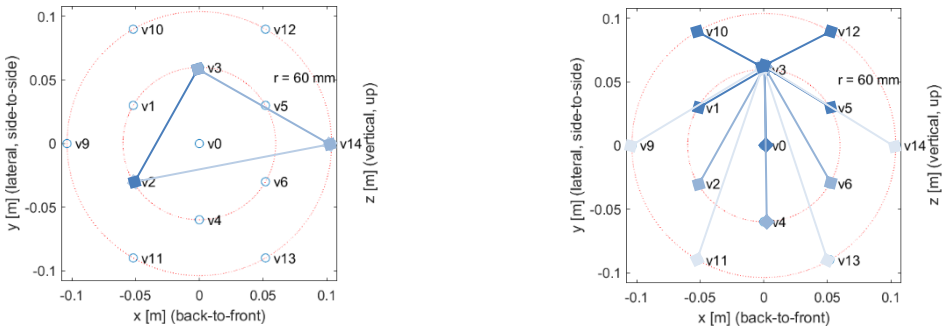

Figure 9: Repulsion of medoids v2 and v3 and v14 (left) and attraction of medoid v3 to non-medoids (right) for ClassroomVideo content

For medoids $\{c_1 \ldots c_k\}$, the repulsion cost is:

$$J = 2 \sum_{\substack{1 \le i < k \\ i < j \le k}} r_{c_i,c_j}^{-2}$$

For medoid c, the (negative) attraction cost is:

$$J = - \sum_{1 \le i < c, c < i \le n} r_{c,i}^{-2}$$

4.2.3.3 Select initial basic views

Some of the source view configurations (especially CG) exhibit symmetry, resulting in multiple solutions with equal cost. To avoid arbitrary selection (undefined behavior) or selection based on multi-view calibration artefacts, pseudo-random initialization is avoided, and instead the initial medoid is selected as the source view that is closest to the following scene position (Figure 10):

1. Maximum x value over all source view positions (tangent x-plane),
2. Average y value over all source view positions,
3. Average z value over all source view positions.

The assumption is made that $+\boldsymbol{x}$ is the forward direction, which is the OMAF convention (cf. Annex B.1). Subsequent medoids (if any) are selected one-by-one by adding the medoid that minimizes the repulsion cost.

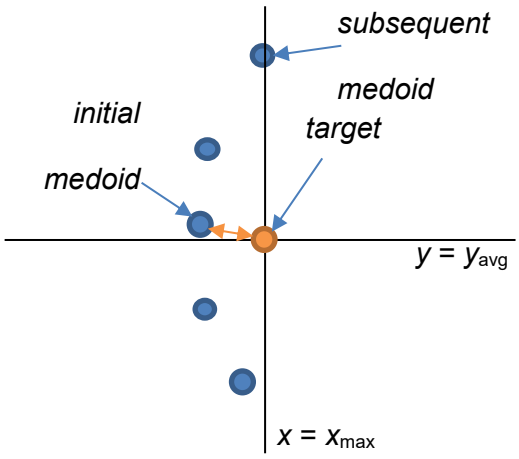

Figure 10: Initial basic view selection

4.2.3.4 Update the view labels

At each iteration, all possible swaps between a medoid (basic view) and non-medoid (additional view) are evaluated. The swap that achieves the largest cost reduction is executed. Iteration stops when cost reduction is no longer possible.

4.2.4 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.

4.2.5 Geometry quality assessment

The quality of the geometry (if present) 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 a threshold 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 the quality of the geometry is supposed to be low.

4.2.6 Atlas frame size calculation

In V3C, each atlas has a frame size 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 with patch positions and sizes aligned by this amount. In MIV, the attribute video data is always at nominal resolution, the geometry video data (if present) is scaled down

by an integer factor $N \geq 1$, and the occupancy video data (if present) is scaled down (usually to the block to patch map's resolution unless specified in the configuration file).

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 there is one attribute, geometry is present, occupancy is embedded in geometry, and no frame packing, 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.

4.2.7 Separation into entity layers

TMIV has the ability to operate in entity coding mode when entity maps are provided for the source views. In this mode, the patches extracted and packed within the atlases have active pixels that belong to a 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 resulting in savings in utilized bandwidth and improved quality. If entity coding mode is chosen, then the source views (texture 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 for each entity independently such that the layers across all views that belong to the same entity are pruned, aggregated, and clustered together. The packing combines patches of all entities together in one set of atlases.

4.3 Atlas construction

4.3.1 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 texture attribute and geometry components and camera parameters, and outputting masks per view and frame of the same size. For additional and semi-basic views, mask values are either 'pruned' or 'preserved'. 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,
- Prefer patches carrying more information,
- Maintain a realistic complexity,
- Consider temporal consistency.

4.3.1.1 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 11). The pruning graph is created in a greedy fashion, which allows to achieve the third goal.

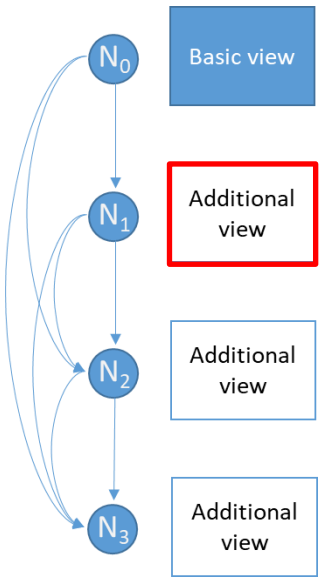

Figure 11: Pruning graph for one basic, one semi-basic, and three additional views (one of them, highlighted in red, was relabeled to semi-basic view). Basic view is assigned to a root node (node id: N_0), each additional view is assigned to a node N_i, which is a child node of all nodes N_j where $j < i$

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 4.3.1.3).
4. Select the additional view with maximum number of preserved pixels (to prefer larger patches).
5. If the current number of semi-basic views is lower than the semi-basic views limit (cf. Section 4.2.3.1), relabel the selected additional view to semi-basic.
6. Insert the selected 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.

7. Project all preserved pixels of selected view to remaining additional views.
8. Update the *pruning mask* for each remaining additional view.
9. 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 in the current test model).

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

4.3.1.2 Pruning cluster graph

The computational complexity of the pruner depends primarily on the number of basic views and additional views in a graph, because each basic view is synthesized to each additional view. The maximum complexity occurs when there are about as much basic views as there are additional views. To reduce the computational complexity, the pruner separates the basic views into clusters having at most a configurable number of basic views per cluster. Each cluster is pruned independently, thus reducing the number of basic views per additional view, at the expense of more active pixels in total.

Additional views are assigned based on two conditions: a) balance the number of views per cluster, b) maximize overlap with one of the basic views in the cluster. Semi-basic views are evenly distributed among the clusters. Because the number of basic views and clusters is limited, exhaustive search can be performed. The score of a solution is based on the sum of overlaps, and the solution with the maximum score is selected.

An example of a pruning cluster graph is provided in Figure 12, with three basic views (yellow), two semi-basic views (green), and four additional views (white). The cluster graph consists of two disjoint graphs, and should be read like this:

- v4 is pruned by v0 and v2.
- v3 is pruned by v0, v2, and v4.
- v1 is pruned by v0, v2, v4, and v3.
- v5 is pruned by v7.
- v6 is pruned by v7 and v5.
- v8 is pruned by v7, v5, and v6.

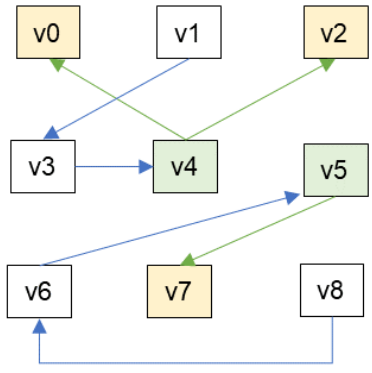

Figure 12: Example of the cluster graph, with notation

4.3.1.3 Pruning mask creation

The pruner uses three 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.
- The minimum difference between luma of a synthesized pixel and luma of all pixels within a collocated source 3×3 block should be less than a pruning luma threshold (cf. Section 4.3.1.4).

Then, as a second-pass process, the pruner updates the pruning mask that was created during the initial pruning stage and re-identify the pixels that are not to be pruned among the pixels that were initially determined to be pruned. The main object of this process is to consider the global color component differences that can exist among different source views. The procedure is as follows and is applied for each pruning pair:

- With respect to the pixels that were determined to be pruned, pixel-by-pixel color differences are calculated between the synthesized view from the parent node and the source view assigned to the child node.
- By using the least squares method, the fitting function that can optimally model these color differences is calculated.
- The pixels that comply with this fitting function within certain range defined by a threshold are judged as the inliers and those pixels are remained as to be pruned. Meanwhile, the outliers are updated as not to be pruned within the pruning mask.
- To reduce the runtime of the fitting operation, a uniform sampling-based approach, that samples all the corresponding pixels at uniform interval, is used to determine the input pixels for the fitting function.

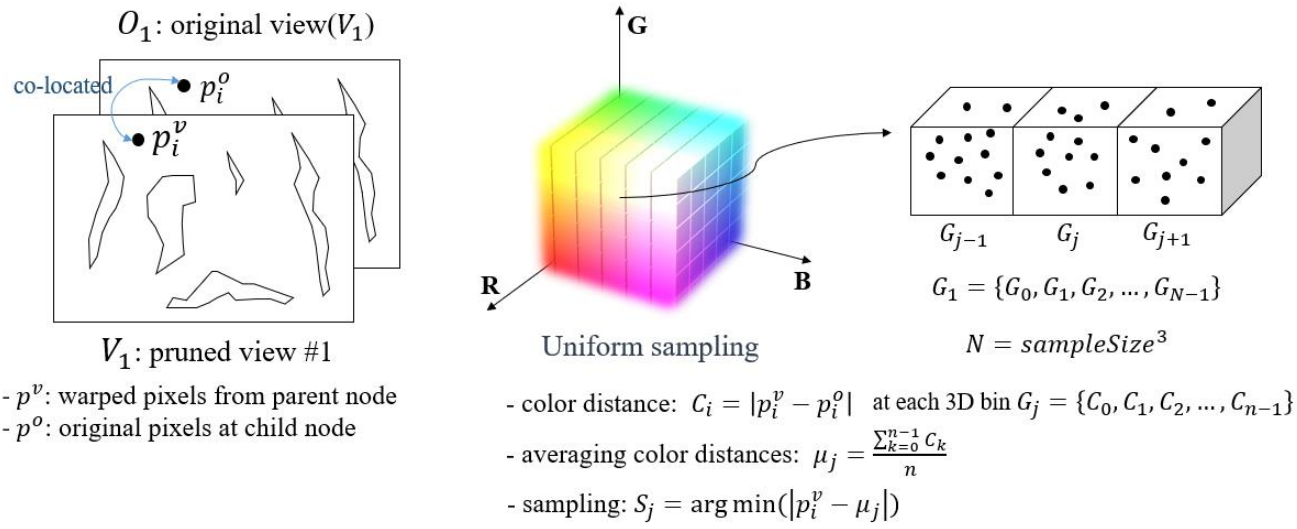

Figure 13: Uniform sampling in second-pass pruner

- The uniform sampling-based approach is conducted with respect to R, G, and B channels (i.e., the input texture component is converted to RGB space if provided in different color space).

Accordingly, the final sampled values are determined by the pixels that are the closest to averaging color distances of all pixels within a cube, which denotes the uniformly divided 3D bin on the RGB color space.

- Given $\boldsymbol{f}_1$ which is the fitting function between V_1 and O_1, the outliers are filtered using the following equation

$$\text{If } |\boldsymbol{f}_1(p_i^v) - p_i^o| > maxColorError$$
$$\text{then } p_i^v \text{ is NOT to be pruned.}$$

and the used parameters are as follows:

- "*maxColorError*" is the threshold to judge whether pruning or not in the second-pass process (default value is 0.1).
- *"sampleSize*" is the number of samples to uniformly divide the pixels on the RGB color space (default value is 32. 0 means no sampling).

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 reset.
- For the dilation, a pixel that has at least one non-empty neighbor is activated.

When creating the pruning masks of a single entity, only pixels that are part of the entity layer are activated. This includes the pruning masks of the basic views.

The pixel effective information is also calculated in this process. The effective information of a pixel measures the minimum degree of non-overlap with respect to its reference pixel, calculated by the sum of depth and luma differences. The pruner updates the effective information map as well as the pruning mask.

4.3.1.4 Pruning luma threshold calculation

The pruning luma threshold (§4.3.1.3) adapts to sequence characteristics, *i.e.*, noise level. The base value of pruning luma threshold (set in the configuration file) is modified by a global luma standard deviation. The standard deviation is calculated for the first frame of the sequence, during the geometry quality assessment step.

At first, an empty set A is created. In order to populate the set A, first of all, all pixels are reprojected between all combinations of 2 source views. For each pixel, the luma of the pixel is compared with the luma of all pixels in the 3×3 neighborhood of the collocated one. If the smallest difference is 0, the luma difference between the reprojected pixel and the center of the collocated block is being included into the set A.

The standard deviation which modifies the value of pruning luma threshold, is calculated as a standard deviation of a set A, containing luma differences calculated for a subset of pixels.

4.3.2 Pruning mask aggregation

The pruning masks (per entity) are aggregated frame-by-frame by activating the active samples of the pruning mask in the aggregated pruning mask. 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 aggregation result. Figure 14 illustrates for a pruned view at frame i, the aggregation of active 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 geometry component, accounting for the motion within the scene.

If the total size of atlases does not allow packing all the patches, the pixel with low effective information will be further pruned in aggregated pruning mask until reach the pixel rate limitation.

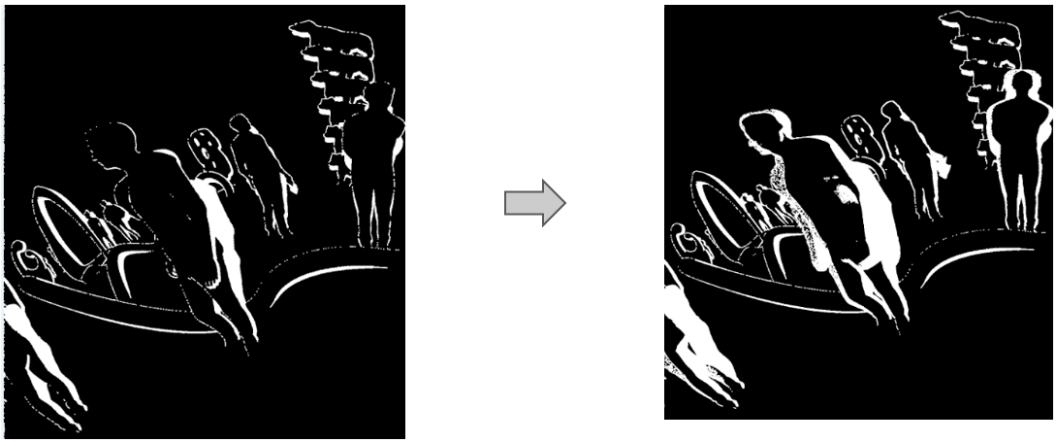

Figure 14. Aggregated mask evolution within an intra period

4.3.3 Clustering active pixels

This block is in charge of identifying "clusters". For an additional view, a cluster is a connected set of pixels that are active in the aggregated mask (of an entity). The connection criterion of a pixel is the presence of at least one other pixel among the eight neighbors.

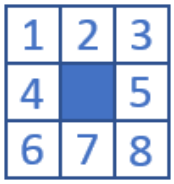

Figure 15: Eight-pixel neighborhood for defining the connectivity criteria for region growing

An example of the clustering is illustrated in Figure 16 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.

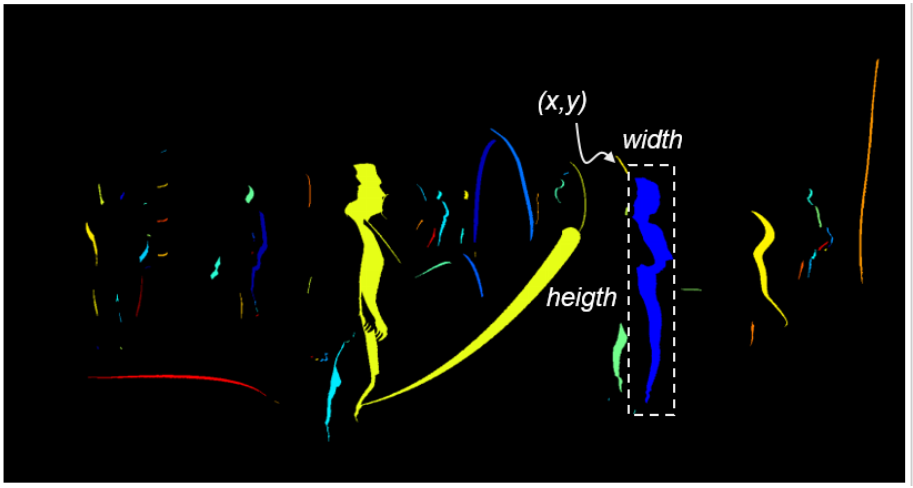

Figure 16: Clusters represented in false color on a pruned view

The result of clustering for a semi-basic view is a single cluster, containing all pixels that are active in the aggregated mask (of an entity). The size of this cluster is the same as the size of the semi-basic view, and its position is set to [0, 0]. For such a cluster, merging and splitting operations (§4.3.4 and §4.3.5) are not performed.

4.3.4 Cluster merging

Smaller clusters may be completely embedded inside the bounding box of another (larger) cluster within a pruned input view (clusters *a* and *b* in Figure 17). The cluster merging includes the smaller clusters inside the bounding box of the larger cluster and generates a single cluster out of the larger and the several smaller clusters. It results in the reduction of the number of patches and the number of associated parameters. As depicted in Figure 17, by merging the clusters *a* and *b* only two patches are generated instead of three.

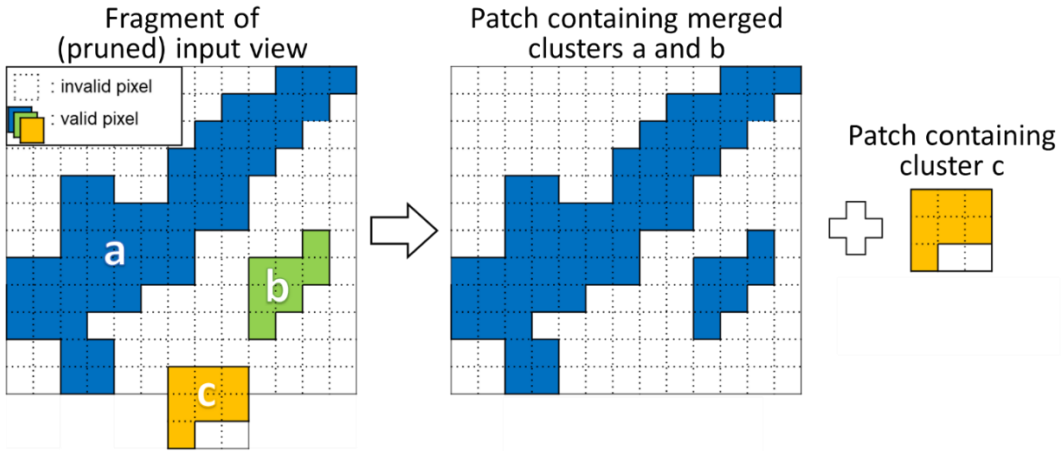

Figure 17: An example of cluster merging

In the merging process, the effective information of each cluster is additionally analyzed. If the difference of effective information density in clusters *a* and *b* is larger than a threshold, these clusters are not merged.

4.3.5 Cluster splitting

In order to reduce spatial redundancy of data in the atlas, irregularly-shaped clusters (*e.g.*, large yellow cluster in Figure 16) 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 a threshold. 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 recursively (Figure 18). Within the entire bounding box of the cluster, the number of blocks (cf. Section 4.3.6) 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 less than a threshold, the cluster is split in half. Splitting of C-shaped cluster usually results in two L-shaped clusters.

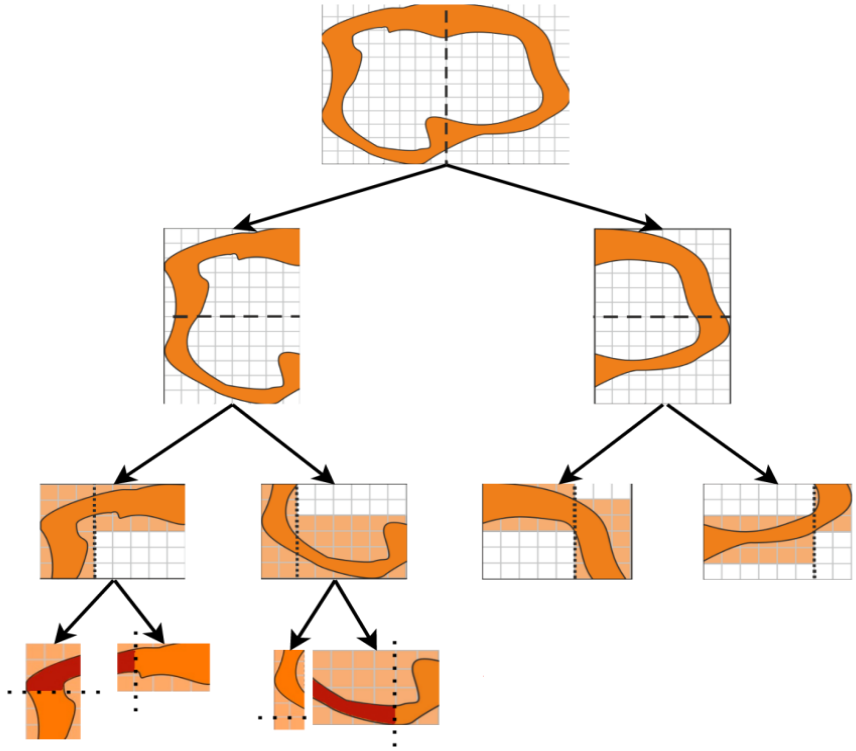

Figure 18: Recursive splitting of the cluster; dashed lines: C-splitting, dotted lines: L-splitting, wide-dotted lines: information-based splitting (cf. Figure 19)

The effective information may be unevenly distributed across cluster. If the total size of atlases does not allow packing all the patches, the information-based cluster splitting is performed recursively in order to preserve more effective information. Each cluster is divided into two smaller clusters if the difference in effective information density between them exceeds a certain threshold. The split is performed in a position which provides highest difference between information density between two sub-clusters (Figure 19). This additional cluster split divides the clusters into clusters with high effective information density and those with low effective information density, and the cluster with high effective information density is more likely to be retained in the atlas.

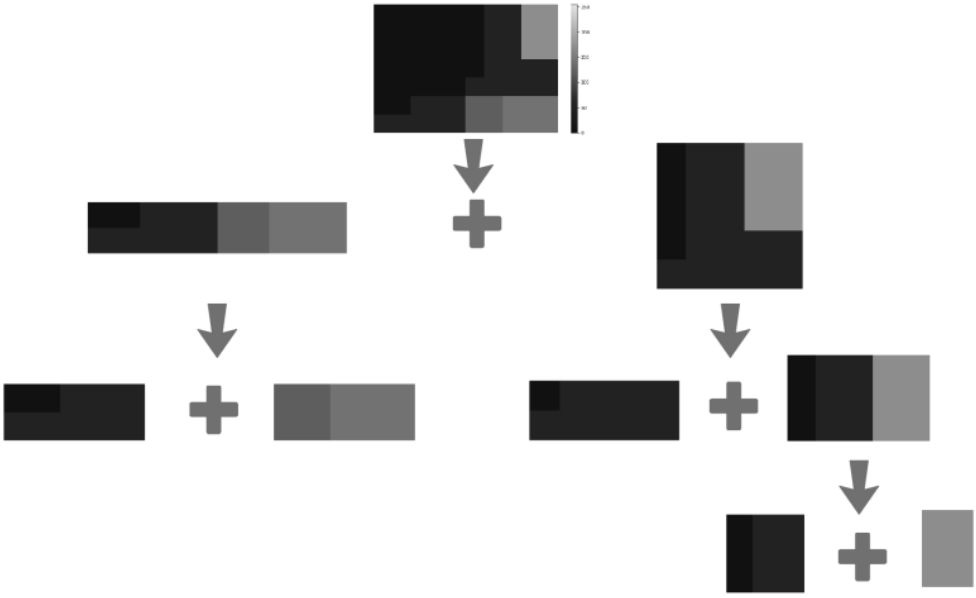

Figure 19: Recursive effective-information-based cluster splitting; the brighter the region, the higher the effective information

4.3.6 Patch packing

The packing process sequentially packs each cluster into the atlases. Clusters are packed as patches, containing a cluster and its bounding box aligned to a N×N grid, where size of the N is one of the packing process input parameters. The cluster is always arranged in the center of a patch, and the size of horizontal and vertical margin of the patch without pixels of the cluster (*hM* and *vM* in Figure 20) is transmitted as a part of patch parameters.

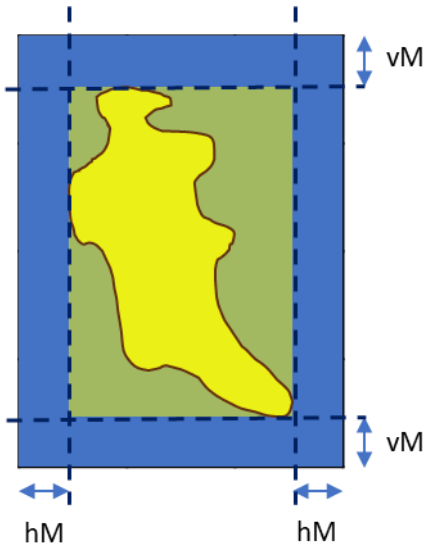

Figure 20: Positioning of cluster (yellow) within a patch (blue); *hM* and *vM* are horizontal and vertical margins of the patch, transmitted within metadata.

The patch packing process is based on a version of the MaxRect algorithm [8]. It considers the available "Used Space" first, by examining the space which is effectively occupied. In a second pass, the "Free space" is considered. It is made of intricated loops as 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 minimum patch size (set in the encoder configuration):
                        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 patch ID (indexing patches within the patch list),
- The atlas ID (indexing the atlas that a given patch belongs to), position and size in the atlas,
- The projection ID (indexing the view that a given patch belongs to), position and orientation in the projection plane,
- The entity ID (indexing the entity that a given patch belongs to) (or 0).

The patch 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).

When per-patch signaling of inpainting data is enabled, patches that originate from the inpainted background view are marked with the ‘pdu_inpaint_flag’. The decoder only uses these patches to fill in the missing data.

Special care is taken to handle basic and semi-basic views. They are never split, rotated or flipped because an appropriate number of basic and semi-basic views (§4.2.3.1) and suitable atlas frame size (§4.2.6) are calculated. Also, because basic views often have the same number of pixels, the ordering of clusters may be arbitrary. Clusters with the same number of pixels are ordered by cluster ID to avoid undefined behavior.

Note that the optional rotation of 90° is clockwise from atlas frame to projection plane, as illustrated in Figure 21. The sample in the top-left corner of is the reference for specifying the position.

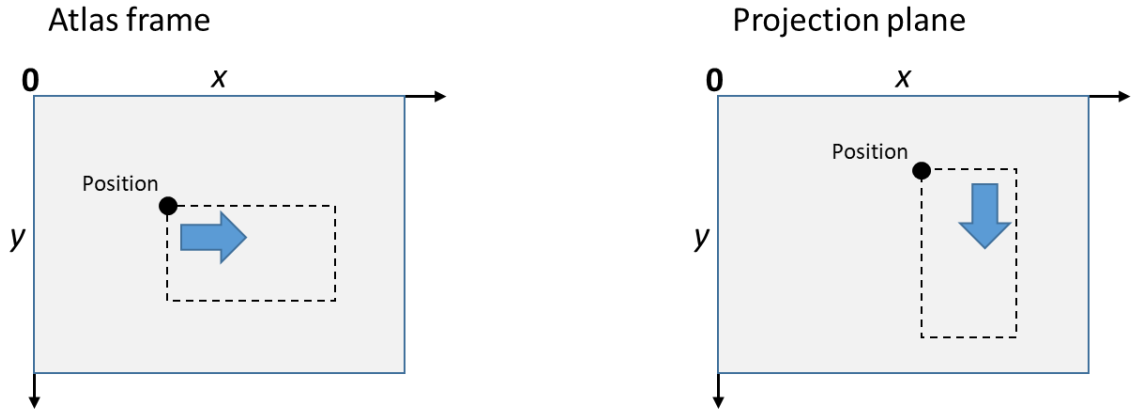

Figure 21: Definition of 90° patch rotation

If the total size of atlases does not allow packing all the patches, patches with higher effective information density are prioritized and packed first. The effective information carried within a patch is calculated by accumulating the value of effective information of active pixels in the patch.

4.3.7 Chroma dynamic range scaling

Two chroma components of texture attribute of each source view may be additionally modified (if the optional chroma scaling is enabled). In such a case, the dynamic range of both chromas of each view is normalized to the range of $[0, 2^m - 1]$, where m is the number of bits per sample used for texture attribute component (cf. Figure 22, where m was set to 10).

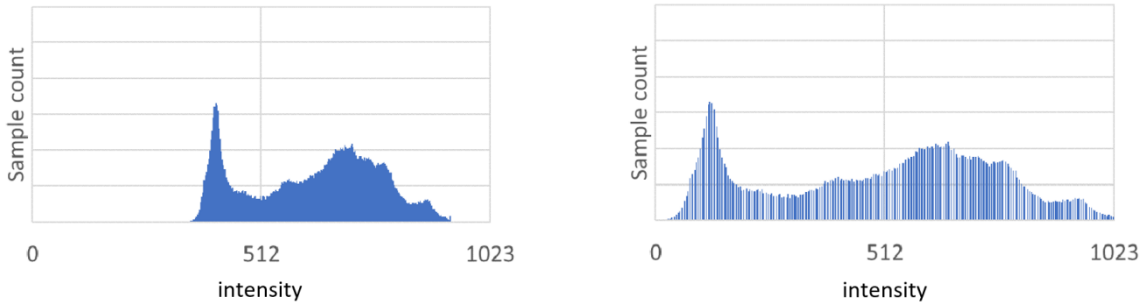

Figure 22: Histogram of a chroma component of a source view: before (left) and after (right) dynamic range scaling

The original range of chroma components for each view are transmitted within the common atlas frame in order to restore the original values at the decoder.

4.3.8 Patch average value modification

After packing patches into atlases, all the attribute patches values are modified, to reduce the number and magnitude of edges between neighboring patches and edges between occupied and unoccupied regions in attribute atlases. The average value of each component of the patch is set to a neutral color, 2^{m-1} for m-bps video (Figure 23). The patch attribute offsets are added in order to restore the original attribute values at the decoder side are sent within atlas data.

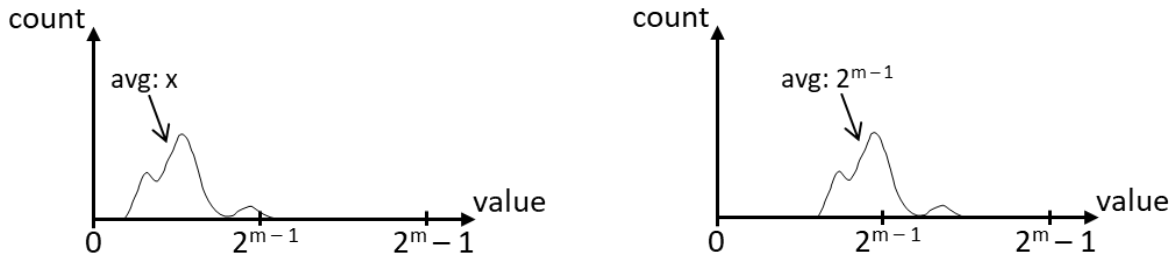

Figure 23: Histogram of an attribute component of a patch: before (left) and after (right) average value modification

If changing the average value to 2^{m-1} causes overflow (pixel exceed the range [0, $2^m - 1$]), the new average value is set according to the size of the overflow (Figure 24).

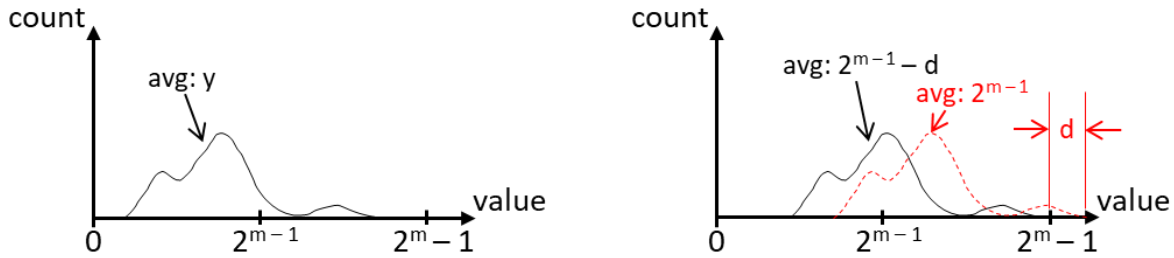

Figure 24: Histogram of an attribute component of a patch: before (left) and after (right) average value modification (overflow avoiding)

4.3.9 Color correction

TMIV has the ability of aligning different color characteristics of source views. If the optional color correction is enabled, color characteristics of each source view are aligned to the color characteristics of a reference view, corresponding to the view captured by the camera which is the closest to the center of the camera rig.

All the pixels from other views are reprojected to the reference view. For each pixel, the color difference between pixel's value and value of corresponding pixel in the reference view is calculated (separately for each texture attribute component).

Then, the patch attribute offsets (§4.3.8) sent within atlas data are modified by subtracting the color correction offsets averaged over the entire patch.

4.3.10 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 entity 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 (texture attribute and geometry) being written to the patches within the formed atlases.

Figure 25 illustrates the generation of an atlas, with the successive write of patch 2, 5 and 8. While the patch packing algorithm uses 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 for an entire GOP by analyzing the aggregated pruning mask. A pixel of a patch is copied to the atlas if there is any non-zero value in a collocated $N \times N$ (cf. Section 4.3.5) block of the pruning mask. Otherwise, it is filled using neutral attribute value and its geometry is set to zero, expressing the invalidity of a sample.

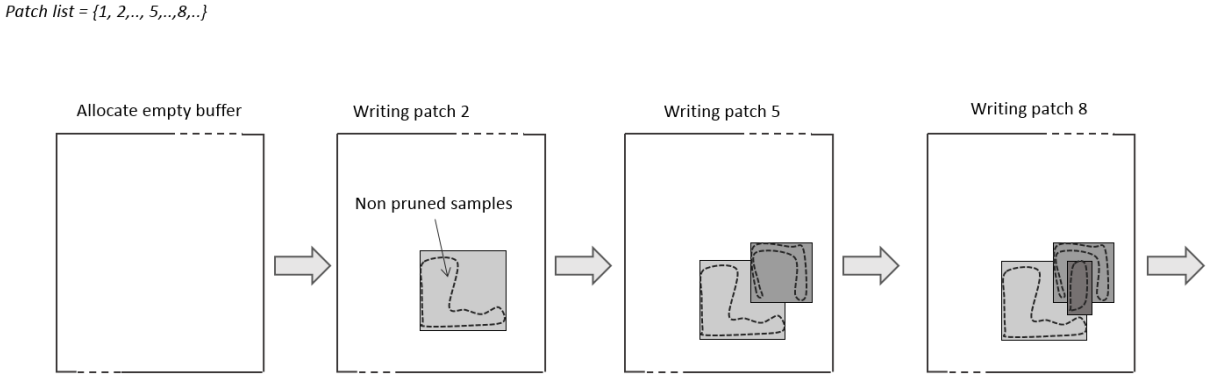

Figure 25: Successive writing of patches into an atlas

4.4 Atlas operations

4.4.1 Geometry coding

4.4.1.1 Linear geometry scaling

An atlas value is either "invalid/non-occupied" or a geometry value expressed in meters, with maximum geometry value set to 1 km. MIV specification [2] specifies how to encode occupancy information within geometry atlases, if occupancy is not present explicitly. The decoding is based on a normalized disparity range, a geometry threshold, and an optional clamping start value. These values are signaled per view or even per patch. Assuming m bits full range geometry atlases with max value $k = 2^m - 1$, the transformation is described in pseudo-code as:

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

Line 1 is part of the block to patch map decoder [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 committee 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 26 and Figure 27:

- When a source view has only valid geometry values, "*depthOccMapThreshold*" is set to zero. This effectively encodes full occupancy (Figure 26).
- 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 27).

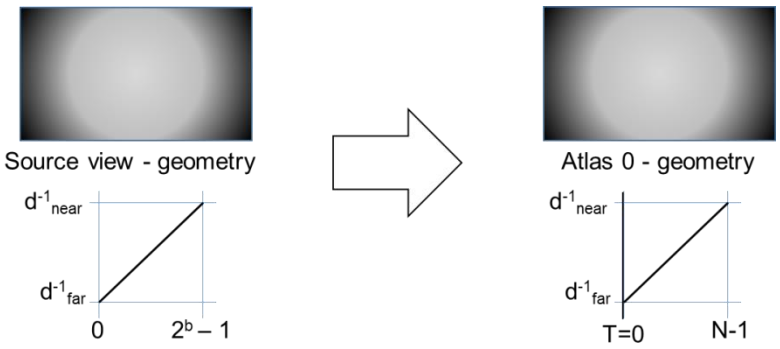

Figure 26: When the source material (with b bits per sample) has only valid geometry values, the geometry coder only performs u(b) to u(m) or u(m – 1) scaling and the geometry threshold is set to zero to signal full occupancy; N = 2^m if the geometry has a good quality (§4.2.5) and 2^{m-1} otherwise

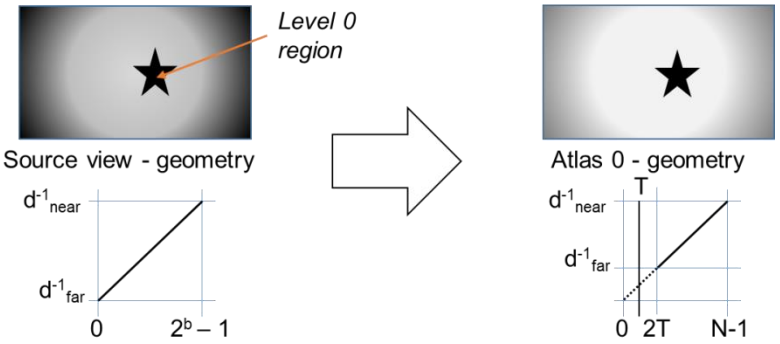

Figure 27: When the source material (with b bits per sample) has invalid geometry values, the geometry coder not only performs u(b) to u(m) or u(m – 1) scaling, but it also sets the geometry threshold to a configurated value (T) and the normalized disparity range is modified such that value 2T corresponds to the far geometry; N = 2^m if the geometry has a good quality (§4.2.5) and 2^{m-1} otherwise

When occupancy video of a given atlas is present (*i.e.*, occupancy is not embedded in geometry), the geometry of that atlas is encoded at the full range (*i.e.*, $T = 0$).

For content with poor-quality geometry component (§4.2.5), N is set lower to use only part of the dynamic range of the video sub bitstream. It reduces the total bitrate without significant reduction of rendering quality.

In order to utilize the whole dynamic range from 0 (or $2T$) to $N-1$, d_{near}^{-1} and d_{far}^{-1} values are recalculated once per GOP (for each view independently), as:

$$d_{near}^{-1} = \max_{all\ pixels\ in\ GOP} d^{-1}(pixel)$$
$$d_{far}^{-1} = \min_{all\ pixels\ in\ GOP} d^{-1}(pixel)$$

4.4.1.2 Piecewise linear geometry scaling[3]

To maximize the efficiency of geometry coding, the entire range of min-max normalized depth value is divided into a predefined number of equal intervals (set in the encoder configuration), and each interval is adaptively scaled according to the importance of the geometry samples in the corresponding interval.

[3] This feature is not supported in MIV v.1 but added to the software as part of TMIV10.1 to facilitate further improvements.

As shown in Figure 25, the original geometry value in each interval is mapped to the corresponding scaled geometry range.

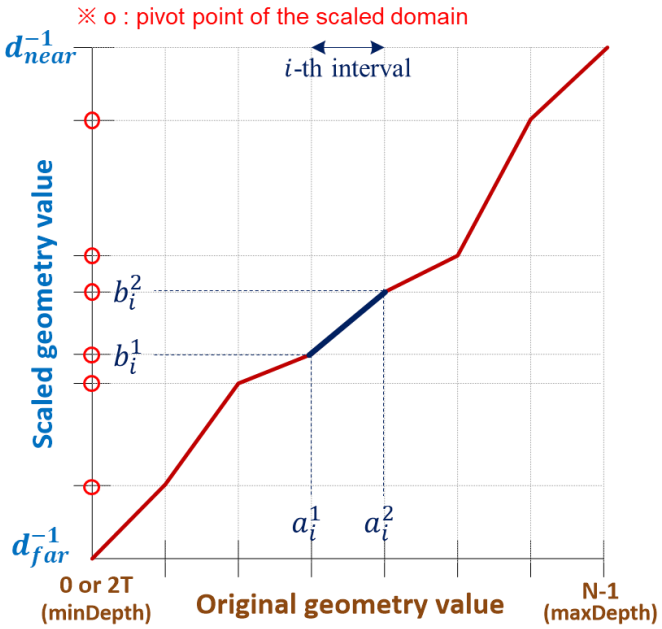

Figure 28: An example of piecewise linear scaling (for 7 intervals)

The piecewise linear scaling is defined for the i-th interval as follow:

$$d' = \frac{(b_i^2 - b_i^1)}{(a_i^2 - a_i^1)} * (d - a_i^1) + b_i^1$$

$$b_i^2 = b_i^1 + (maxDepthValue - minDepthValue) * \frac{p_i}{P_{total}}$$

$$P_{total} = \begin{cases} \sum_i p_i, & for\ good\text{-}quality\ geometry \\ \max\left(\sum_i p_i,\ 1\right) & for\ poor\text{-}quality\ geometry \end{cases}$$

d is an original geometry value to be scaled, and i is the interval index for the d value. The i-th interval on the original geometry range $(a_i^1, a_i^2]$, is mapped to the i-th interval on the scaled geometry range $(b_i^1, b_i^2]$, which is defined by the consecutive pivot points b_i^1, b_i^2. The scaling value of each interval is determined according to the histogram of geometry values that are considered as edge regions. p_i means the probability value of the i-th interval in the histogram, and P_{total} is the sum of the probability values of all intervals. The range of the i-th interval in the domain to be remapped is determined according to the ratio of p_i to P_{total}. The higher the importance of the geometry value, the steeper slope of the linear equation is derived with the larger the difference between b_i^1 and b_i^2. In this way, more codewords are allocated to the interval with higher importance in the remapped domain, which results in more efficient representation of geometry. Inverse scaling is required to perform rendering at the decoder side, and it can be performed in the reverse way of the forward scaling. The pivot values of the scaled domain and their number are sent to the decoder side for inverse scaling. The number of pivot values is equal to the number of intervals −1 because it excludes both ends of the full depth dynamic range.

The derivation of piecewise linear scaling model is performed as follows:

1. The original geometry range is divided into several equal intervals. The default interval number is set to 16 and can be adjusted in the encoder configuration file.
2. Calculate the percentage of sample occurrence in each geometry interval and limit the occurrence frequency to the limited range [lowerLimitProb, upperLimitProb] to avoid over-scaling. Different ranges of limitation are set according to the quality of geometry.
3. Adjust the interval range by scaling based on the calculated statistics resulting from step 3.

The scaling is applied to keep the range of remapped geometry within the ±25% of the case of uniform distribution by setting the limitation range to $\left[\frac{0.75}{number\ of\ interval}, \frac{1.25}{number\ of\ interval}\right]$. For content with poor-quality geometry component (§4.2.5), the total probability values can be lower than one to reduce the dynamic range of the remapped geometry values. The min/max values of the probabilities of overall intervals are set to lowerLimitProb and upperLimitProb, and the probabilities of the remaining intervals are linearly mapped to within the limit range by probability scaling.

4.4.2 Geometry downscaling

When enabled 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.

4.4.3 Occupancy downscaling

If the TMIV encoder is configured to output occupancy video data (instead of embedding the occupancy information in the geometry video data), then the full-resolution occupancy maps are downscaled by a configurable scaling factor. The default factor is the inherent resolution of the occupancy maps (*N* in §4.3.10). For entity-based coding a higher resolution is recommended. The decoder reconstructs the full-resolution occupancy maps by performing upscaling using nearest neighbor interpolation. Note that for complete atlases (*e.g.*, atlases that include basic views only), occupancy maps may not be output since all pixels are occupied.

4.4.4 Embedded occupancy error correction

For sequences with high quality geometry (§4.2.5), the "*depthOccMapThreshold*" (§4.4.1) is asymmetrical and 1.5 times higher at the decoder side; *i.e.*, the TMIV encoder uses a threshold 1.5 times lower, than signaled within the MIV bitstream. When geometry quality is low, "*depthOccMapThreshold*" is equal in encoder and decoder.

4.4.5 Patch boundaries filtering

In order to reduce amount of high frequencies in the atlas, marginal rows and columns of all patches (blue regions in Figure 20) are being modified, and the texture in these regions is removed and iteratively inpainted.

In the first iteration, for each empty pixel its 3x3 neighborhood is analyzed. If there is any non-empty neighbor, new luma and chroma values of the pixel are calculated by averaging values from the entire neighborhood. Then, the size of the window is increased, and the second iteration is executed. The window size is iteratively increased until reaching the size of 63x63, or until all pixels are inpainted.

After inpainting, all the inpainted regions are additionally filtered (by averaging neighboring pixels), resulting in smoother transitions within inpainted area.

Moreover, if the geometry has a poor quality (§4.2.5), the geometry atlas is filtered analogously. Otherwise, only the texture atlas is modified.

4.4.6 Frame packing

Frame packing can be applied on the video data components (functionality is integrated in TMIV) or on the resulted sub-bitstreams after the video coding (requiring an external tool and limited to VVC Test Model – VTM).

4.4.6.1 Packing of video data components into a packed video data

Atlas video components are divided into multiple regions and packed under each other (per atlas) such that the attribute region is on top, the geometry regions (if present) below it, and the occupancy regions (if present) at the bottom. For a single tile setting, the texture attribute is considered one region, the geometry and occupancy components are divided row-wise into multiple regions based on their scaling information along the x-dimension since the packing is done along the width of an atlas. When multiple tiles are available then the number of regions is equal to number of regions in one tile setting multiplied by the number of tiles. For instance (assuming 1 tile and frame packing is enabled),

- In MIV anchor or MIV View anchor with geometry scale 2,

  # Regions = 1 (for texture attribute) + 2 (for geometry) = 3

- In explicit occupancy test case for Classroom sequence with geometry scale 2 and occupancy scale 16,

  # Regions = 1 (for texture attribute) + 2 (for geometry) + 16 (for occupancy) = 19

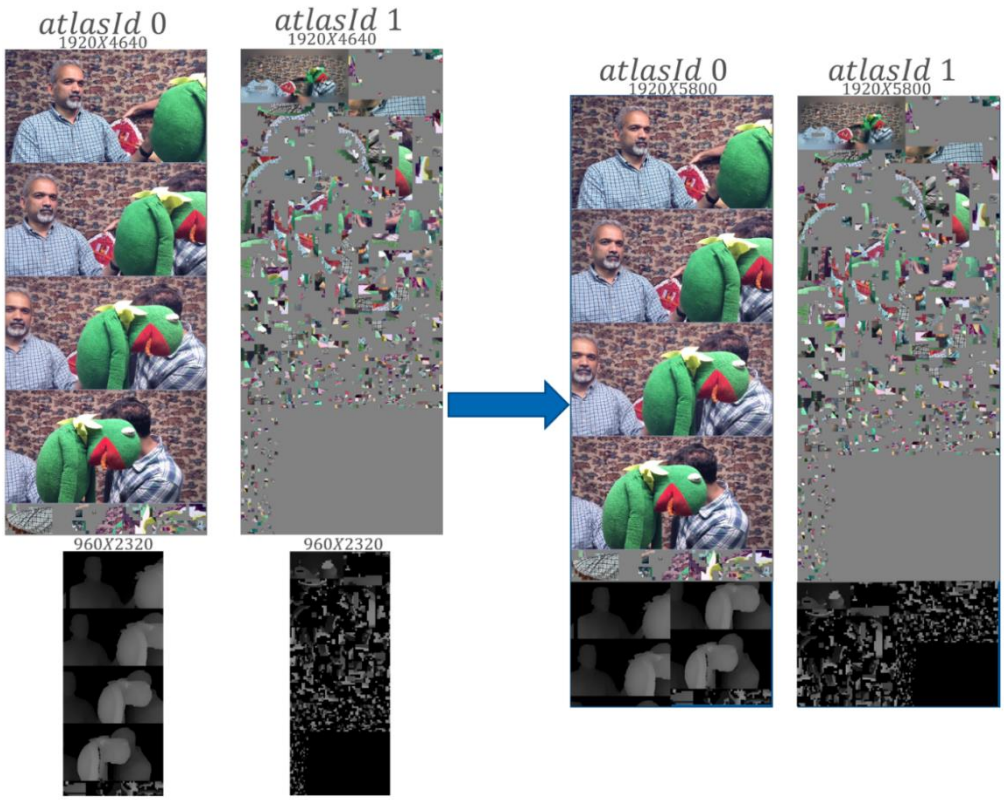

Figure 29: Frame packing for MIV anchor – Frog sequence at frame 0, featuring 3 regions

Example of frame packing in MIV anchor for the Frog sequence is shown in Figure 29. Note that the frame packing information are updated to reflect the applied packing process so regions can be mapped back from PVD to AVD, GVD, and OVD at the decoder side.

4.4.6.2 Packing of non-attribute video data into a packed video data

Frame packing of (§4.3.8) requires the video components packed in the same atlas video to have the same bit depth. This does not take advantage of the advanced capabilities provided in video encoders of state-of-art video coding standards (e.g., VVenC) where subpicture coding can be used to enable coding the geometry in larger bit depth compared to the other components (which improves the reconstruction quality). In this implementation, the non-attribute video components can be encoded together in a separate packed video unit and in a subpicture that is different from that the attribute video unit is encoded in. i.e. The geometry component (if present) is divided into multiple regions and packed vertically to have the same height as the attribute component. In explicit occupancy test case, the occupancy component is divided into multiple regions and packed right to the geometry component. The geometry and occupancy components are divided column-wise into multiple regions based on their scaling information along the y-dimension since the packed video data should have the same height as the attribute component to utilize subpicture merging.

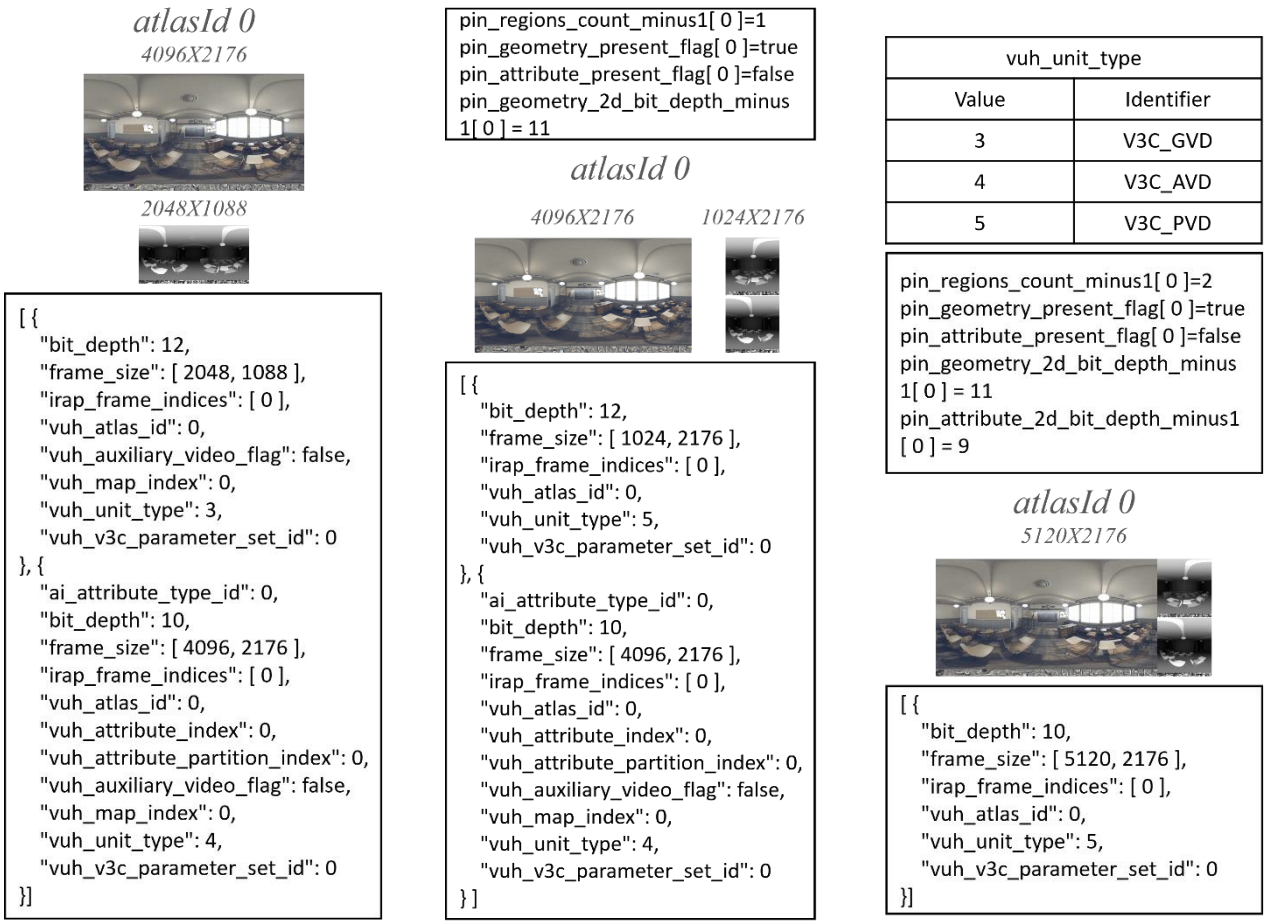

Figure 30: Geometry packing for MIV anchor – ClassroomVideo sequence at frame 0: MIV anchor (left), geometry packing (center), geometry packing + attribute bitstreams merging (right)

Example of a standalone geometry packing in MIV anchor for the ClassroomVideo sequence is shown in Figure 30. Note that integer values 3, 4, 5 for vuh_unit_type represents V3C_GVD, V3C_AVD, and V3C_PVD, respectively. Center figure represents the geometry packing for MIV anchor, where AVD and PVD (containing GVD) have the same height, where the PVD has two regions. subpicture encoding is applied to the video components, then the subpicture bitstreams can be merged using subpicture merger, where PVD becomes to the right of AVD, as shown in the right side of Figure 30. Because the AVD was included into the PVD, it has 3 regions. TmivBitstreamMerger is used to modify the packing information of the V3C bitstream and out-of-band metadata to reflect the subpicture merging.

4.4.6.3 Packing of video sub-bitstream components

A subpicture merging software[4] allows for encoding several VVC bitstreams separately, and merge selected encoded bitstreams into a single VVC compliant bitstream with multiple sub-pictures. Figure 31 presents the packing with two VVC bitstreams, one for the texture attribute video data, and one for the geometry video data. An example of a packed video data for a given atlas featuring a texture video data at full resolution and a downscaled geometry video data for ClassroomVideo content is shown in Figure 32. By encoding sub-picture separately, it can be ensured that the rate-distortion optimization is correct for each packed component and that the bitrate of the merged bitstream will be approximately

[4] "AHG3/AHG12: Subpicture merging software", ISO/IEC JTC1/SC29/WG11 input document m54168, June 2020, online meeting.

the same as the sum of the separate sub-picture bitstreams. Note that the subpicture merging software supports merging for subpictures which are divisible by the coding tree unit (CTU) size (*e.g.*, 64 in HEVC and 128 in VVC).

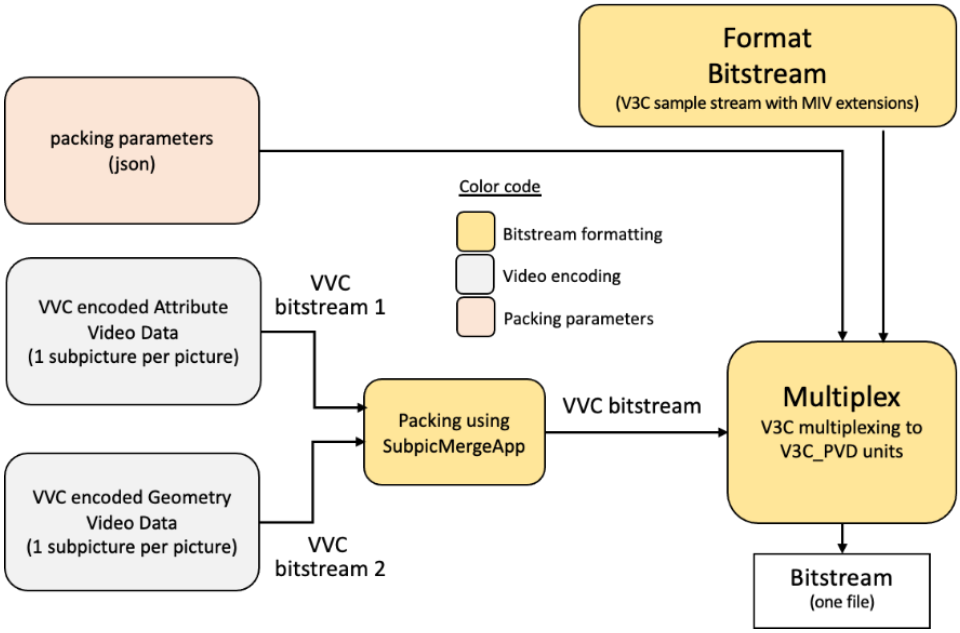

Figure 31: Encoding of texture and geometry components into one packed video sub-bitstream

To merge subpictures that are not divisible by the CTU size, they must be in the last column or row of the merged picture. The width and height of each subpicture can be calculated using the following equations:

$$pic_{w_{CTU}} = \left\lfloor \frac{pic_w + CTU_w - 1}{CTU_w} \right\rfloor$$

$$pic_{h_{CTU}} = \left\lfloor \frac{pic_h + CTU_h - 1}{CTU_h} \right\rfloor$$

where $pic_{w_{CTU}}$ and $pic_{h_{CTU}}$ denote width, height of a picture in CTUs, while pic_w and pic_h represent width, height in luma samples. CTU_w and CTU_h denote width, height of a CTU in luma samples.

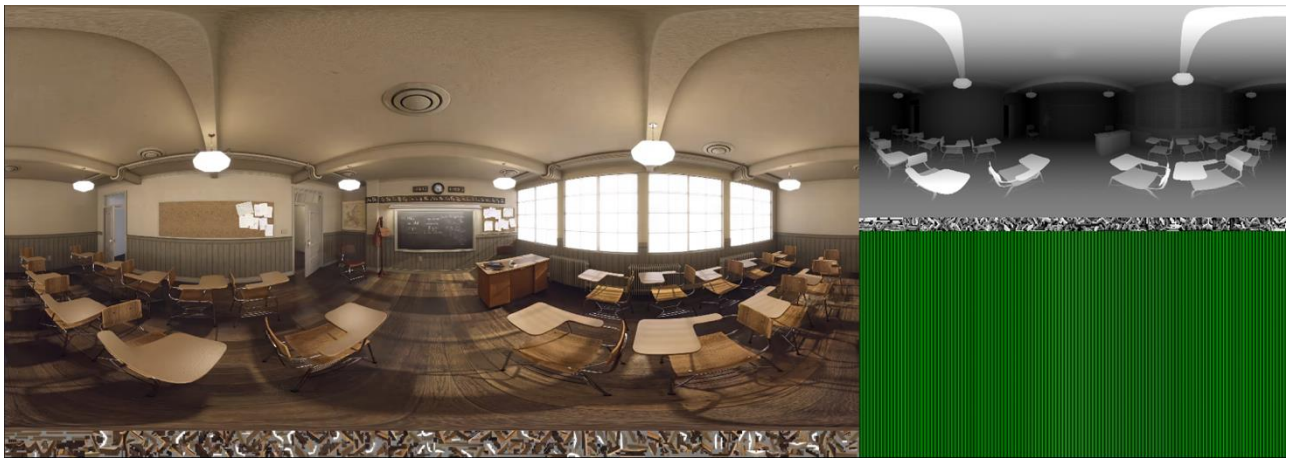

Figure 32: An example one frame of classroom video sequence with packed atlases

4.4.7 Colorized geometry coding

Colorized geometry coding allows coding the higher bit-depth geometry by embedding the extended bit-depth information to U and V component. It requires more bitrate but improves the rendering quality. Let f_n be the quantization function with n-bit for 16-bit geometry information G. In a conventional MIV system, 10-bit geometry information G_{10} is obtained by $f_{10}(G)$. Similarly, N-bit $(N > 10)$ geometry information G_N is obtained by $f_N(G)$ and then the first 10 bits are assigned to the Y component, and last $(N - 10)$ bits are assigned to U and V components, respectively. Figure 33 graphically explains this approach when the target bit-depth is 12.

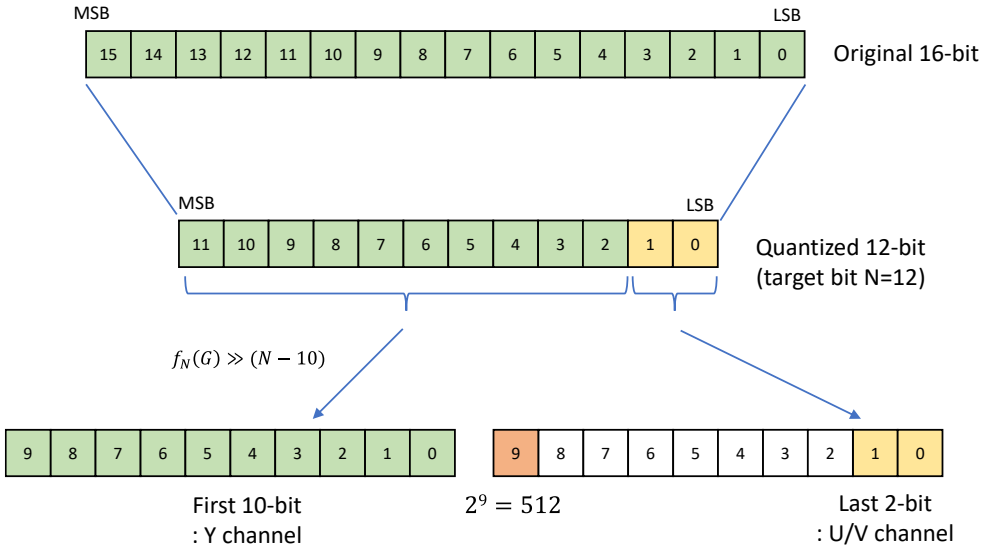

Figure 33: Example of colorized geometry coding (N = 12)

The data in U/V components are adjusted by adding 2^9=512 for the efficient compression. In the TMIV decoder, the reverse process is applied, that is, subtracting 512 for U/V components, and then restoring the 2^N-bit geometry information by combining the data in Y and U/V components. It is common that the video codecs adopt the subsampling of chrominance components such as YUV 4:2:0 format. In this case, the resolution of Y-channel is $H \times W$ while both U and V components only support $\frac{H}{2} \times \frac{W}{2}$. To assign the additional bits of Y component, the odd number of rows are mapped to U component, and the even number of rows are mapped to V component, respectively. Then, every two adjacent pixels p_1 and p_2 are merged to a single pixel p_m as in Figure 34.

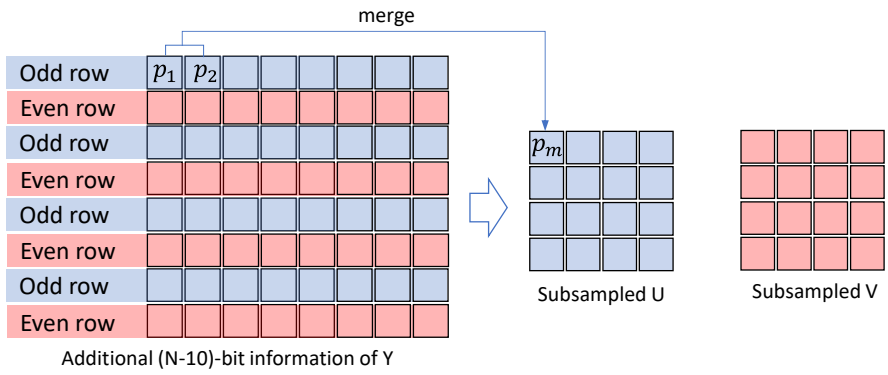

Figure 34: Example of additional (N-10)-bit information of Y component to assign U and V components (YUV 4:2:0)

4.5 Separated background view coding

The separated background coding allows TMIV to independently encode pre-separated background and non-background views. A background view is 2D rectangular arrays of view samples consisting of attribute frames and corresponding geometry frame as a collection of static or rarely-changing parts (e.g., "background") of a MIV scene. And the non-background view is 2D rectangular arrays of view samples excluding the view samples represented by the background view. Typically, the foreground objects can be represented by the non-background views (see Figure 35). The background views can be merged spatially or/and temporally for purposes such as reducing runtime in the decoder, replacing background on the scene, improving temporal consistency in background area etc. In particular, the spatially merged background view can be prepared by reprojecting all background source views to a single camera position (e.g., the central camera).

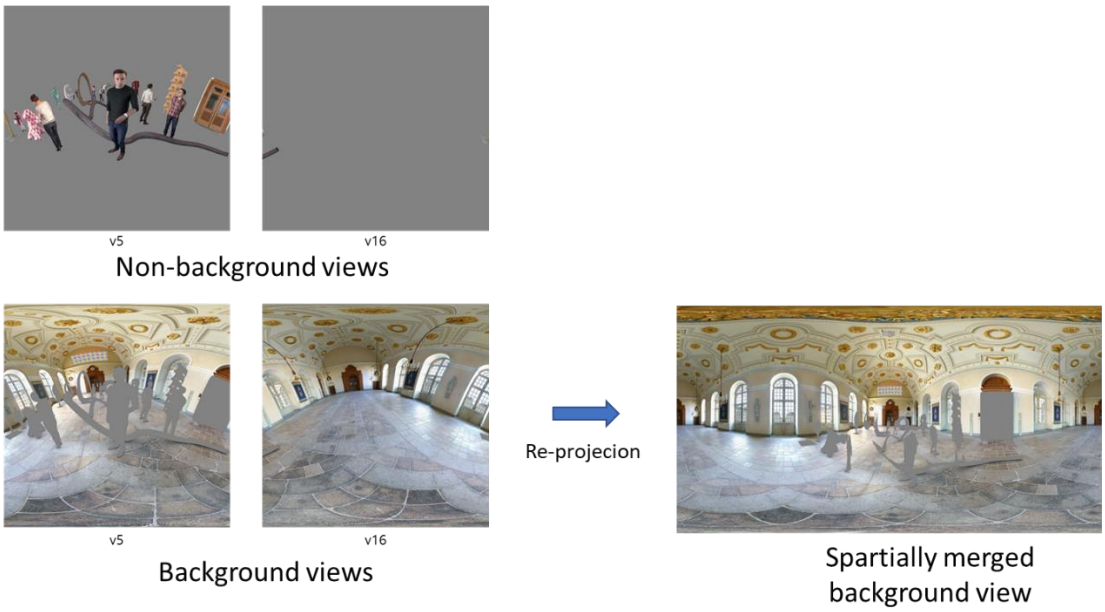

Figure 35: Example of separated background views and non-background views

For separated background view coding, the TMIV encoder receives the background source views and non-background source views respectively, described in Figure 36. And then, the pre-separated views are encoded by each single group encoder independently, but metadata are merged into a single bitstream. Therefore, non-background and background views can be set by two different types of encoding parameters and packed into different atlas frames (see Figure 37)

In the process of selecting the basic views in non-background view encoding, the view labeler selects a view with as much valid region as possible. It is noted that the valid region is determined by the distribution of moving objects in the non-background views. In the first step, an initial basic view is determined as the view with the maximum value when accumulating the valid pixels within central area of non-background views. To ensure that the initial basic view is included in the basic views, the view label update step (described in section 4.2.3.4) is not performed because it may exclude the initial basic view determined in the first step.

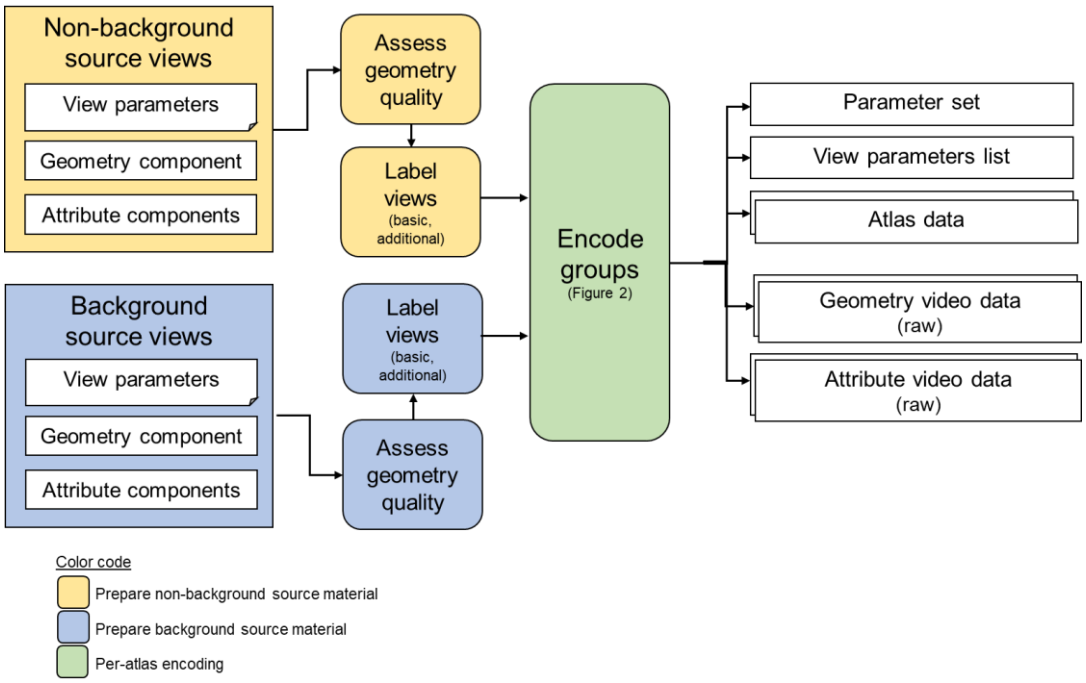

Figure 36: Top-level diagram of the separated background view encoder

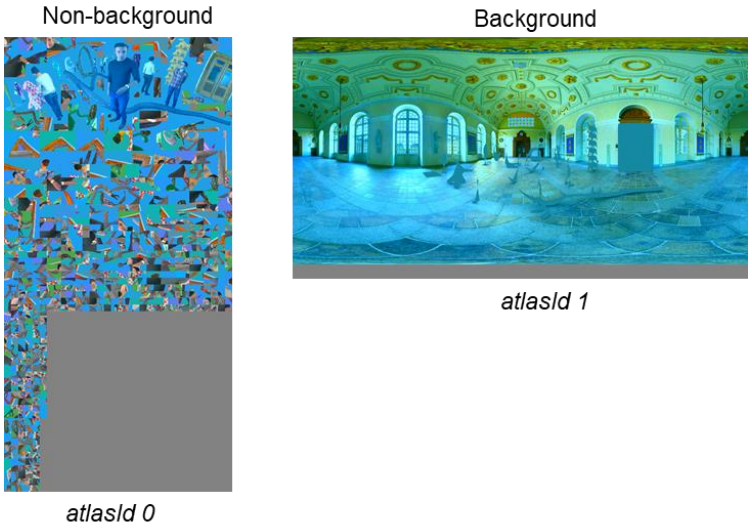

Figure 37: Example one frame of Museum video sequence when non-background views and background view are encoded

As a useful use case, the background view can be a static view that is composed of objects with little movement, which can be created by merging the spatially merged background views over a period of time. In this case, '*staticBackgroundFlag*' is set to true and '*intraPeriod*' is typically set to 1 in the encoding configuration for background views (note that '*intraPeriod*' is typically set to 32 for non-background views), which allow the video codec to operate at different frame rates each other.

4.6 Video encoding

MIV is agnostic to the video codec and the test model is designed to work with external video coding tools. TMIV currently includes two video codecs; the first codec is HEVC with Main 10 profile and random-access configuration where the TMIV decoder is capable of decoding HEVC sub bitstreams by inclusion of the HEVC Test Model (HM). The second codec is VVC using the Versatile Video Encoder (VVenC) implementation of it.

4.7 Multi-plane image encoder

4.7.1 Multi-plane image

A multi-plane image (MPI) is a layered representation of a 3D scene. The scene[4] is decomposed into a set of planar or spherical layers (see Figure 38) sampled at different depths from a given reference point of view. Each layer is a color + transparency frame obtained by projecting the part of the 3D scene contained around the layer location on the same reference camera. This reference camera is positioned at the given reference point of view. The reference camera is a perspective camera when using planar layers, or a spherical (typically equirectangular) camera when using spherical layers.

In the following, we consider the reference camera with resolution W x H and S layers.

In the case of MPI, there is no view synthesized inside TMIV encoder. There is only one MPI camera at the input of the TMIV encoder. The TMIV encoder processes it. Generated metadata carries the parameters for that camera.

Transmitting MPI over MIV requires the activation of the transparency and the use of depth constant patches. The layer index, that a patch is issued from, is written in atlasPatch3dOffsetD of this patch, and is used at decoder side for synthesis.

The input source view for the TMIV MPI encoder has the reference camera parameters (projection, resolution, …) and also the number of layers.

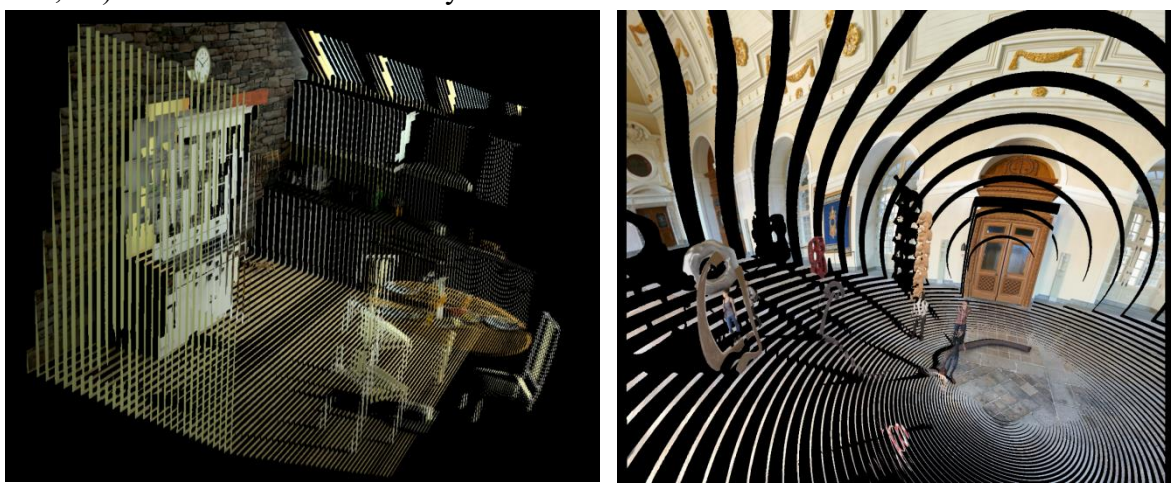

Figure 38: Texture MPI layers of different projections; perspective Kitchen (left) and equirectangular Museum (right)

4.7.2 Input MPI format

An MPI content in raw storage cannot be directly input to the TMIV encoder.

An MPI content in packed compressed storage (PCS) can be input to the TMIV encoder.

4.7.2.1 Raw storage

Raw storage of an MPI (cf. Section 4.7.1) is not effective in terms of memory usage because a lot of samples are empty. Moreover, it induces many small disks accesses which may reduce the I/O efficiency. An example of MPI content is shown in Figure 39.

Raw storage is defined as below:

- texture layers must be provided as a single file,
- transparency layers must be provided as a single file.

Each file (texture or transparency) is a temporal concatenation of the stacked layers sorted from the furthest to the closest one with respect to the reference camera. The layer index is the quantized normalized disparity coded on *ceil(log2(S))* bits. Raw storage is illustrated in Figure 40.
The transparency is coded on 16 levels only, hence 4 bits would be enough. But an 8 bits format is used to ease file manipulation.

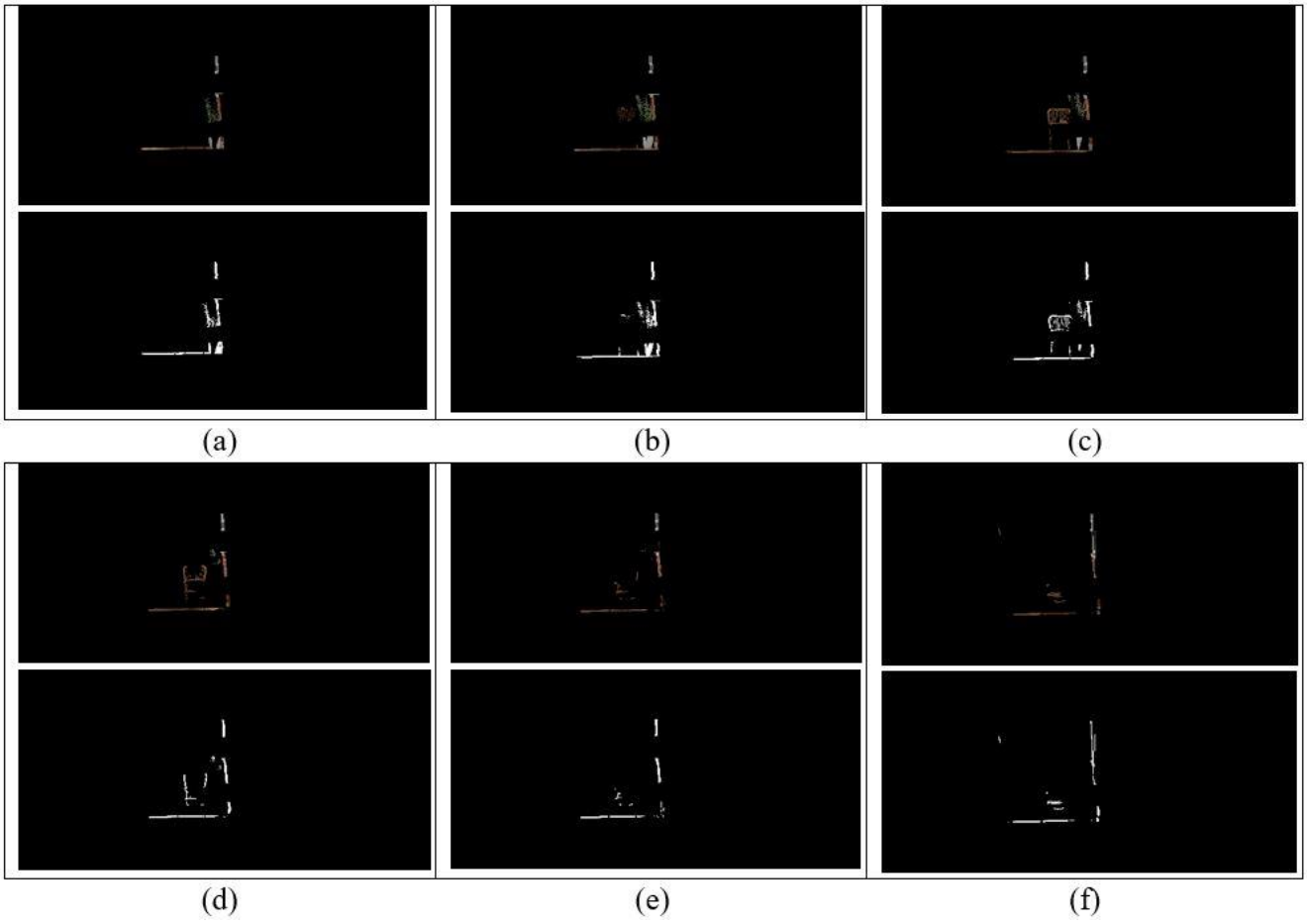

Figure 39: Example of MPI layers from Mpi_Fan, from layer 40 (a) to layer 45 (f). Texture is on the first row and transparency is on the second row

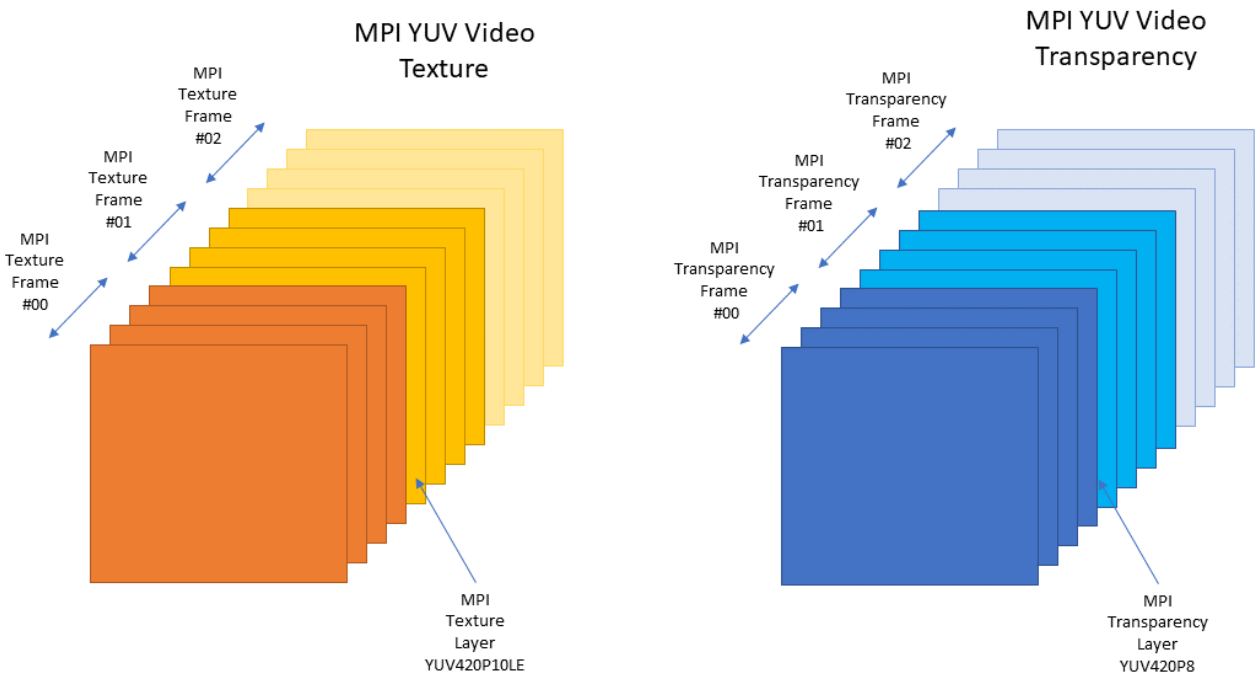

Figure 40: MPI raw storage diagram. In this simplified example, the MPI content is made of 4 layers (sorted from far to close) and 3 concatenated temporal frames

4.7.2.2 Packed compressed storage (PCS)

To mitigate raw storage issues, a compressed version of the MPI is defined. This compressed version is the one needed at input of TMIV encoder.
For a given temporal frame, the number of active layers of a given pixel (i,j) is given by $N_{i,j}$ ($N_{i,j}$ in [0, S]). We can then define for each temporal frame of the MPI:

- $N_{i,j}$: the number of active layers associated to pixel (i,j)
- $C_{i,j,k}$: the color of pixel (i,j) for the k-th active layer
- $D_{i,j,k}$: the layer index of pixel (i,j) for the k-th active layer
- $T_{i,j,k}$: the transparency of pixel (i,j) for the k-th active layer

where k is an integer in the range [1, $N_{i,j}$]. The layer index $D_{i,j,k}$ is the quantized normalized disparity coded on *ceil(log2(S))* bits.
The W x H number of active layers per pixel are first provided in the bitstream, and the concatenated list of all samples is immediately stacked after as presented in Figure 41.

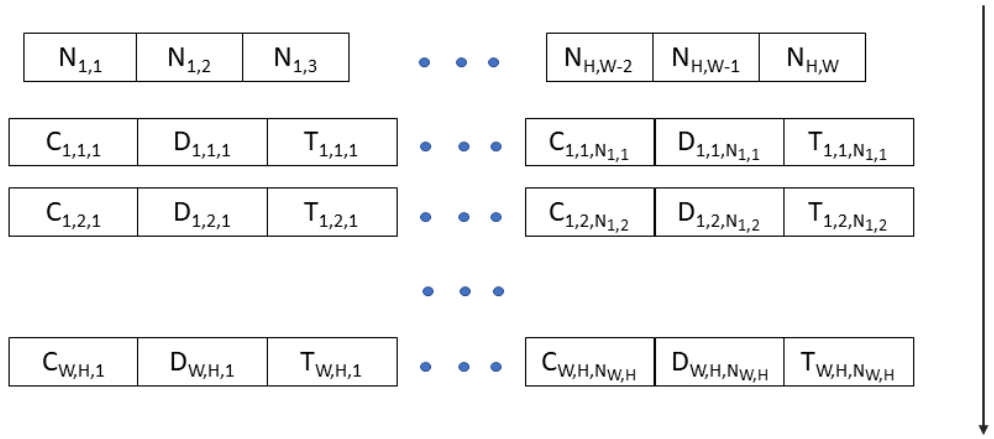

Figure 41: MPI packed compressed storage (PCS) layout.

When reading an MPI temporal frame, one first read the W x H frame containing the number of active layers per pixel. The total number of active layers is easily derived (sum over all pixels). It then makes possible to read the whole sample list at once for current temporal frame. This results in only two disk "accesses" to read a temporal frame. Once in memory, this packed structure can be kept as it and accessed by the means of a dedicated table derived from the number of active layers per pixel.

4.7.2.3 Raw storage to PCS

TMIV encoder can only handle MPI content in PCS compressed version. An MPI content in raw storage should be first converted into the compressed version to be processed by the TMIV encoder. A specific executable is available for this conversion. Please refer to Section 4.7.2.1 for the expected format of an MPI content in raw storage.

4.7.3 MPI encoding process

An MPI is a non-redundant representation of the 3D scene. Therefore, there is no need for a pruning process inside the TMIV Encoder. The processing steps for encoding an MPI content with the TMIV MPI encoder (cf. Figure 42) are presented below.

The first step of the TMIV MPI encoder is the mask creation.

For each layer, the transparency is read, and a transparency map is created. Then a thresholding operation is performed to detect occupied pixels (threshold is set to 0 in the current implementation) which leads to a binary mask per layer. This operation is repeated for each frame of an intra-period, and an aggregated binary mask per layer is generated at the end of the intra-period. Hence this aggregation process is done as in Section 4.3.2 per layer instead of per view. In the end, there is one binary mask per layer for the intra-period.

These aggregated masks are then sent to the clustering process, which delivers clusters to the packing process for each layer. This clustering step is the same as the one described in Figure 2, cf. Section 4.3.3, Section 4.3.4 and Section 4.3.5.

The obtained clusters are then packed using the process described in Figure 2, cf. Section 4.3.6, but the

sorting operation is changed to get clusters from distant layers packed first rather than the biggest clusters. This sorting allows an efficient rendering process at decoder side, known as reverse painter's algorithm.

Finally, two attribute atlases are generated, one for the texture and one for the transparency similarly to what is done for texture and geometry in a regular process.

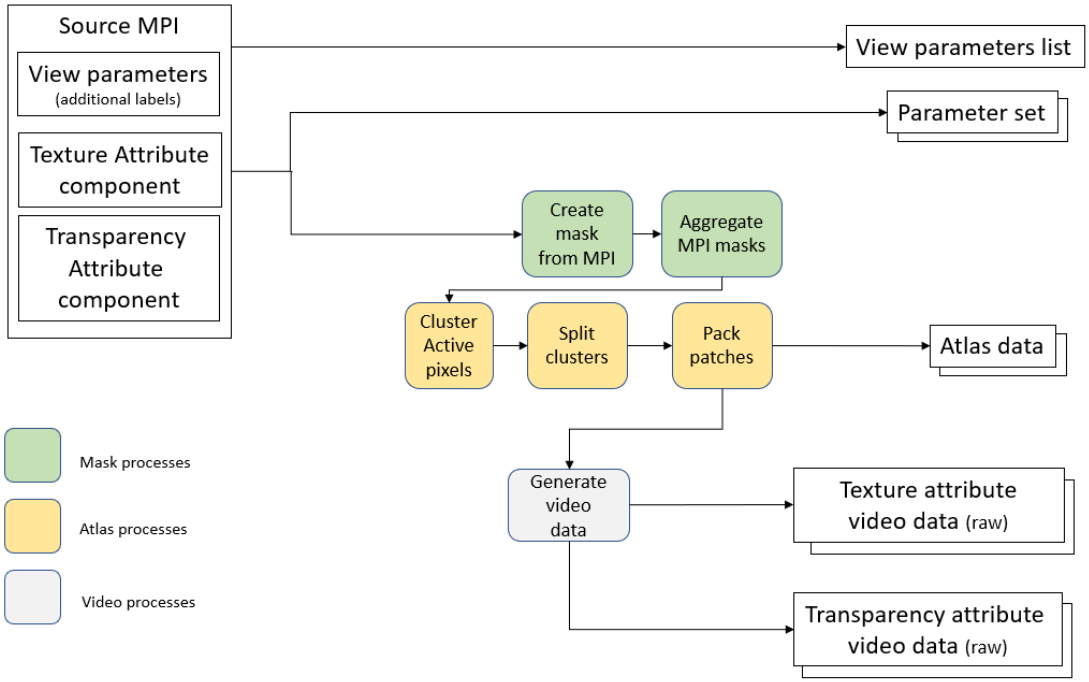

Figure 42: Top-level diagram of the TMIV MPI encoder

4.8 Bitstream formation and multiplexing

The output of the TMIV encoder is a V3C sample stream with MIV extensions (Figure 43). The V3C sample stream consists of a V3C parameter set, common atlas data, atlas data, optional geometry video data, optional attribute video data, optional occupancy video data, and optional packed video data. The atlas data is a NAL sample stream which includes also the SEI messages. The common atlas data sub bitstream contains the view parameters list while the regular atlas data sub bitstreams contain the patch data. The 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 or VVenC as an external tool to encode entire video sub bitstreams at once. The formatting and multiplexing are 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 (if present), attribute (if present), occupancy (if present), and packed (if present) video data are output as raw video files.
4. The raw video is encoded using the HEVC test model (HM) or the versatile video codec (VVenC) 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.

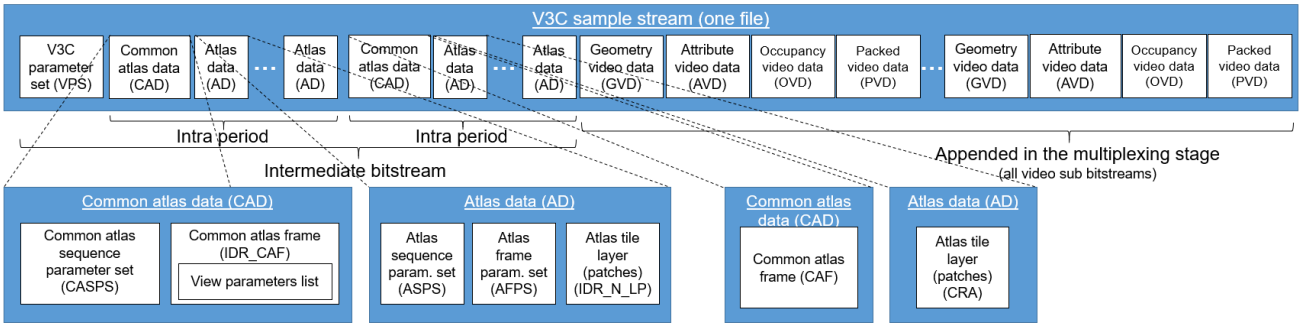

Figure 43: V3C sample stream with MIV extensions

5. Description of the rendering processes

The TMIV decoder follows the MIV decoding process described in the MIV specification [2] including the demultiplexing & decoding order, bitstream parsing, video decoding, frame unpacking, block to patch map decoding, and resulting conformance points. This section describes the non-normative renderer (Figure 44) starting from the conformance points. This includes the following stages:

- Block to patch map filtering including entity filtering and patch culling to speed up the rendering,
- Reconstruction processes including occupancy reconstruction, attribute average value restoration, and pruned view reconstruction,
- Geometry processes including geometry scaling, depth value processing, and depth estimation,
- View synthesis including unprojection, reprojection, and merging,
- Viewport filtering including inpainting and viewing space handling.

The output of the TMIV renderer (which can be run explicitly or as part 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 with texture attribute and geometry components. 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.

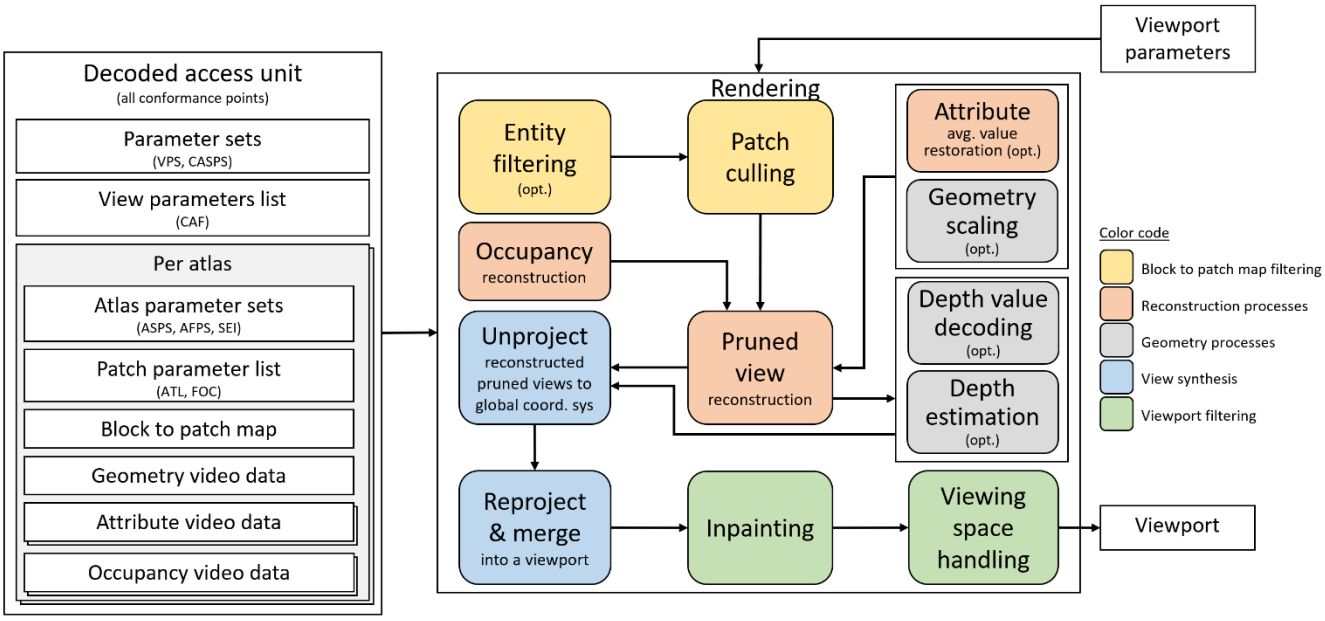

Figure 44: Process flow for the TMIV renderer

In addition to the regular TMIV rendering, support for MPI rendering is added to the software and described further in Section 5.7.

5.1 Background video data decoding

This is an optional stage that is enabled by the separated background coding. When the atlas video data is *static*, which is described in the MIV specification, the MIV decoder receives new frames of the background video data components only at the start of intra period for non-background, and then it receives the previous frames of video data components during the remaining intra period.

5.2 Block to patch map filtering

5.2.1 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 use case 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 H of the MIV specification.

5.2.2 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 ($P_i'(x, y), i = 0, 1, \ldots, 7$) has no overlap with the target viewport, the patch is culled.

The patch map is updated as illustrated in Figure 45. If the patch is culled, the corresponding atlas's samples are labeled as unused and ignored during the rendering process.

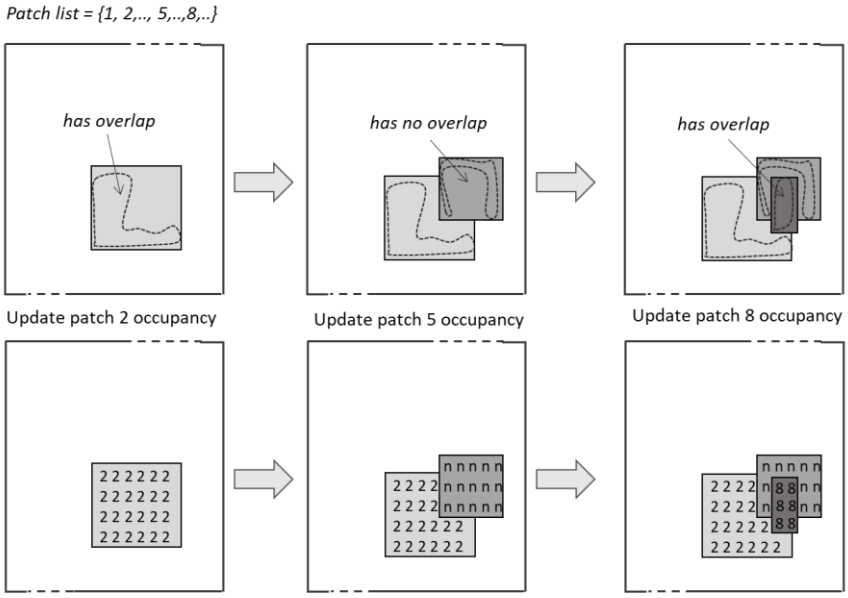

Figure 45. Occupancy map update in an ordered manner with patch culling

5.3 Reconstruction processes

5.3.1 Occupancy reconstruction

This process reconstructs an occupancy frame at nominal atlas resolution whether it is embedded in the geometry frame, signaled explicitly, or no occupancy information case (*i.e.*, atlas is fully occupied).

- When occupancy is embedded in the geometry frame, the occupancy frame is extracted from the geometry one (after the geometry upscaling process) by comparing its pixel values against the "*depthOccMapThreshold*" defined in Section 4.4. When the threshold is smaller or equal to the geometry value, the occupancy value at the given pixel is set to 1 otherwise it is set to 0.
- When occupancy video sub bitstream is present (*i.e.*, signaled explicitly), thresholding process is applied to retrieve the binary occupancy from the decoded one and nearest neighbor interpolation is performed to reconstruct the occupancy map at nominal atlas resolution.
- In the case of no occupancy being signaled (*i.e.*, atlas is fully occupied or atlas patch with constant depth), the occupancy frame at nominal atlas resolution is filled with ones.

More details on the occupancy reconstruction process are available in Annex H of the MIV specification.

5.3.2 Chroma dynamic range restoration

If the optional chroma dynamic range is enabled, the TMIV decoder restores the original dynamic range of each chroma component for all transmitted views using values transmitted within the common atlas frame.

5.3.3 Attribute mean value restoration

Please refer to Annex H of the MIV specification for more details.

5.3.4 Pruned view reconstruction

The reconstruction of the pruned views is an operation opposite to video data generation performed in the encoder (Section 4.3.10). 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 46: patches 2, 3, 5, 7 and 8 are copied to proper position in proper views, based on their position in the view they belong to. Except for basic views, significant part of most reconstructed views is empty.

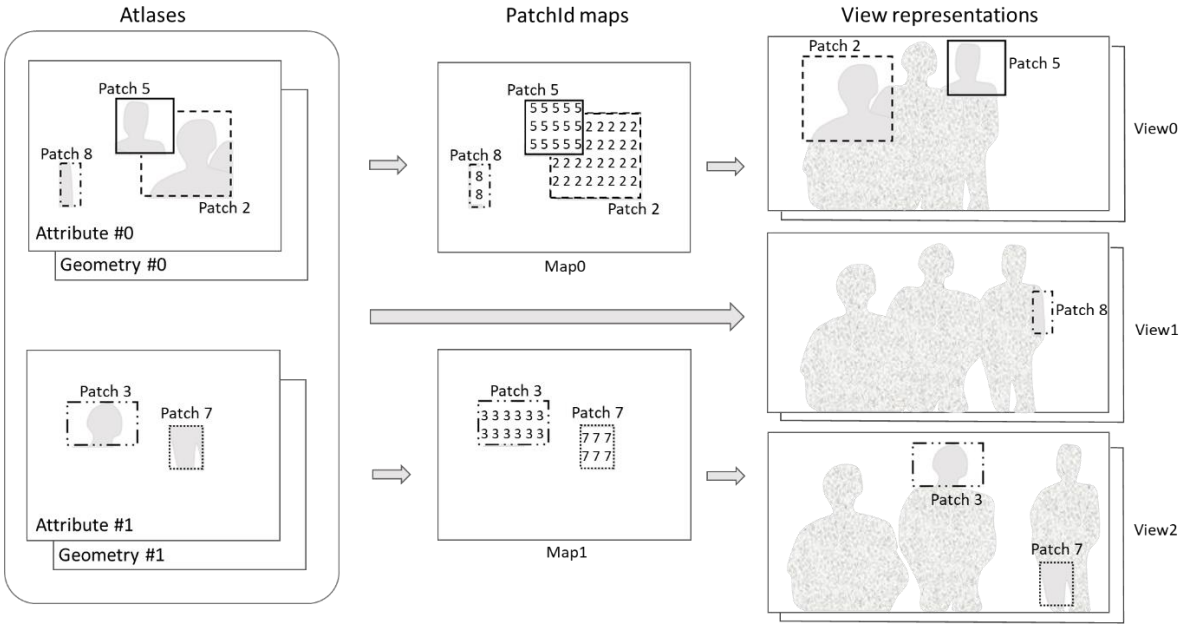

Figure 46: Pruned view reconstruction

By default, entire patches are copied to pruned source views. However, for patches containing pixels from additional views, there is a possibility to skip boundaries of each patch by setting "*patchMargin*" parameter in the TMIV decoder configuration file. In this case, *k* leftmost and rightmost columns, as well as *k* topmost and bottommost rows of each patch are not copied to pruned source view.

5.4 Geometry processes

5.4.1 Colorized geometry recovery

Please refer to Annex H of the MIV specification for more details.

5.4.2 Geometry upscaling

Please refer to Annex H of the MIV specification for more details.

5.4.3 Depth value decoding

Please refer to Annex H of the MIV specification for more details.

5.4.4 Depth estimation

When TMIV encoder operates in the MIV Geometry Absent or MIV Extended Decoder-Side Depth Estimation profile, a depth estimation process may be invoked at the decoder side inputting the reconstructed pruned views and the associated view parameters to compute and output the depth maps (*i.e.*, the geometry frames). The renderer then uses them along with the texture pruned views to render the targeted viewports.

Currently, TMIV software does not have an integrated depth estimation tool but it is possible run the TMIV decoder to output the view parameters and the texture pruned views, use a standalone MPEG depth estimation software such as IVDE [4] and DERS [5] to estimate the depth maps externally, and then feed them into the TMIV renderer to proceed with the rest of the rendering operations

5.5 View synthesis (unprojection, reprojection, and merging)

The TMIV proposes two alternatives for the synthesis. The first one is RVS-based [6], described in Section 5.5.1, the second one is the View Weighting Synthesizer (VWS), described in Section 5.5.2.

5.5.1 RVS-based synthesizer

The RVS-based synthesizer[5] is based on RVS [6] with a per-pixel blending scheme based on the Philips CfP response.

5.5.1.1 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 texture 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 texture attribute lists, and triangle descriptor lists (which is very much like OpenGL). The

[5] In the reference software the name is AdditiveSynthesizer

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 5.3.4).

The RVS-based synthesizer has currently no support for handling encoder-side inpainted data.

5.5.1.2 Rendering from atlases

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

- Multiple *k*-bps texture attribute atlases and *l*-bps 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 *k*-bps texture attribute and *l*-bps geometry components.

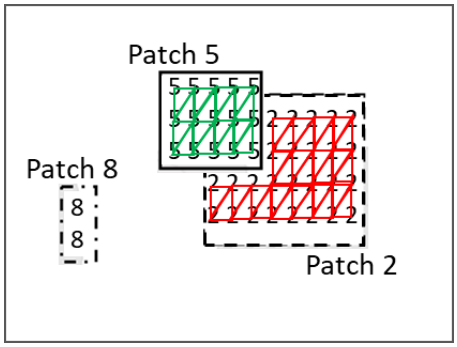

Figure 47: Creating a mesh from an atlas. Triangles between pixels from Patch 5 and 2 are omitted. Note that Patch 8 is not drawn because no triangle can be formed. Unused pixels are skipped too

The process is to build a mesh (Figure 47) from each of the atlases:

- The **vertex descriptor list** is formed pixel-by-pixel:
  - Skip or write dummy values for unoccupied pixels,
  - Looking up the atlas parameters list using the Patch ID in the block to patch map,
  - Looking up the view parameters list using the Projection ID 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.
- The **triangle descriptor list** is formed by:
  - For each pixel consider two triangles [/]
  - Add the triangle when all vertices have the same Patch ID.

This mesh is then rasterized using barycentric interpolation of texture 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.

5.5.1.3 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).

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

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 target camera.	Prefer nearby views over views 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.

5.5.2 View weighting synthesizer

5.5.2.1 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 [9] instead of triangulation. Optionally (by setting parameter "*filterReprojectedPrunedDepthMaps*" to "true" in the TMIV decoder configuration file), warped geometry maps may be additionally refined by morphological erosion and dilation in order to reduce impact of video encoding artifacts. 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 5.5.2.2. 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. Finally, areas of the viewport which are co-located with edges within the visibility map are blurred.

5.5.2.2 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 4.3.1.1 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 48). 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:

4. 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.

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

6. 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.

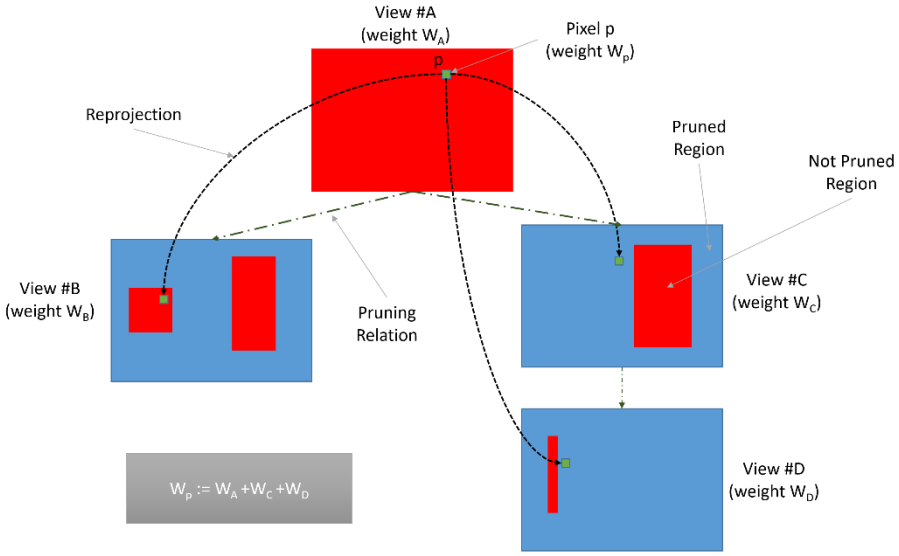

Figure 48: graph-based pruning: weight recovery procedure

5.5.2.2.1 Handling of inpainted data

Some patches within the atlases belong to the inpainted background view pre-synthesized at the encoder side (see Section 4.2.2). They are of lower quality than the original source views. That pre-inpainted data is projected to the target viewport but otherwise excluded from the steps of making a visibility map and viewport shading. Only in the shading stage where the viewport visibility is invalid, i.e., missing data, the reprojected inpainted data is inserted. For pixel-locations where the resulting blending weight is below a threshold and for the locations where pre-inpainted data is unavailable, the target depth is set to invalid, making them available for inpainting as a post-processing step.

5.5.2.3 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.
filterReprojectedPrunedDepthMaps:		
dilateCount	int	Number of dilation filtering iterations applied on reprojected depth maps.
erodeCount	int	Number of erosion filtering iterations applied on reprojected depth maps.

edgeBlurringBlockSize	int	Size of the block, in which the texture values are blurred.
edgeBlurringDepthThreshold	float	Maximum difference of depth values considered as “similar depth” for edge blurring purposes.

5.6 Viewport filtering

5.6.1 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 texture 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 [7]). 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 49a). 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 49b). 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.

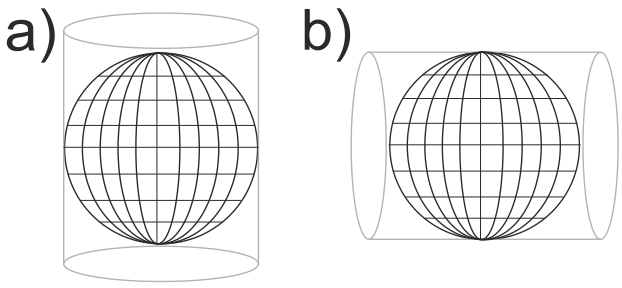

Figure 49: Cylinders used in the projection of a sphere on a flat image in a) equirectangular projection and b) transverse equirectangular projection

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 50a). In a second step, all columns of such image are expanded (Figure 50b), to be of the same length (Figure 50c).

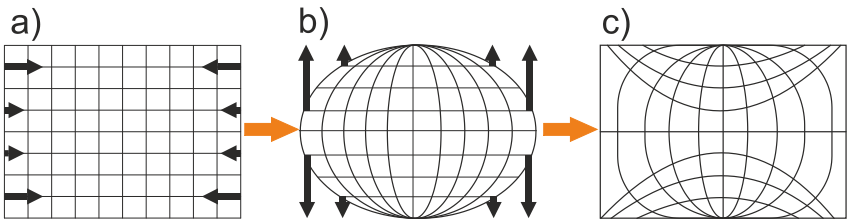

Figure 50: Fast reprojection of an equirectangular image (a) to transverse equirectangular image (c). Black arrows show direction of change of size of respective rows and columns of images

5.6.2 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 51, two modes of viewing space are illustrated, as well as viewing direction with the arrows.

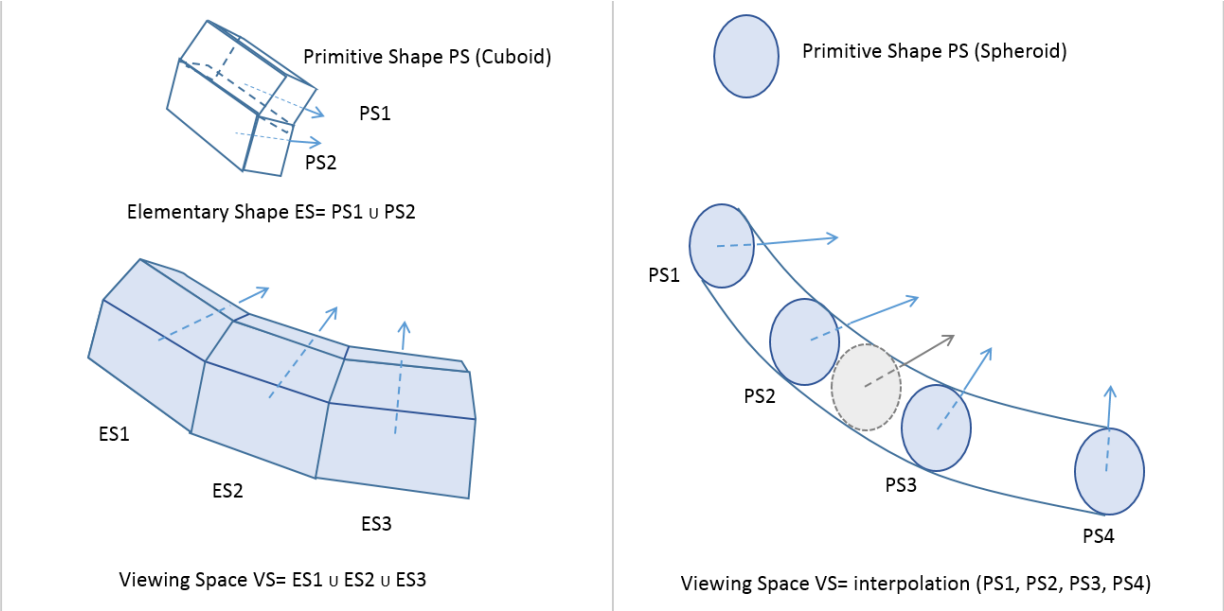

Figure 51: Illustration of VS creation with additive CSG (left) and interpolation (right)

5.7 Multi-plane image renderer

The MPI renderer is simpler and faster than the MIV renderer since the complexity (visibility, anti-aliasing, *etc*.) is handled during the creation of the MPI and not when the view synthesis is done.

Figure 52 shows the MPI version of Figure 44 for the TMIV renderer. As regards to the block diagram in Figure 28, the differences are the following:

- The layer depth value decoding block works on patch basis and outputs a constant depth value per patch.
- For the view synthesis, the rendering is done by projecting and blending the different layers from the closest to the farthest along each ray, taking into account the associated transparency values (reversed Painter's algorithm [11]). The process operates along each ray starting from the optical center of the MPI reference view center and related to a viewport pixel, accumulates and blends the value of each MPI layer along that ray until the result increases up to the saturation value of 1. Since this saturation value correspond to the opaque value, there is no need to get the values of what is behind as seen from the viewport and all further layers are discarded for that ray.

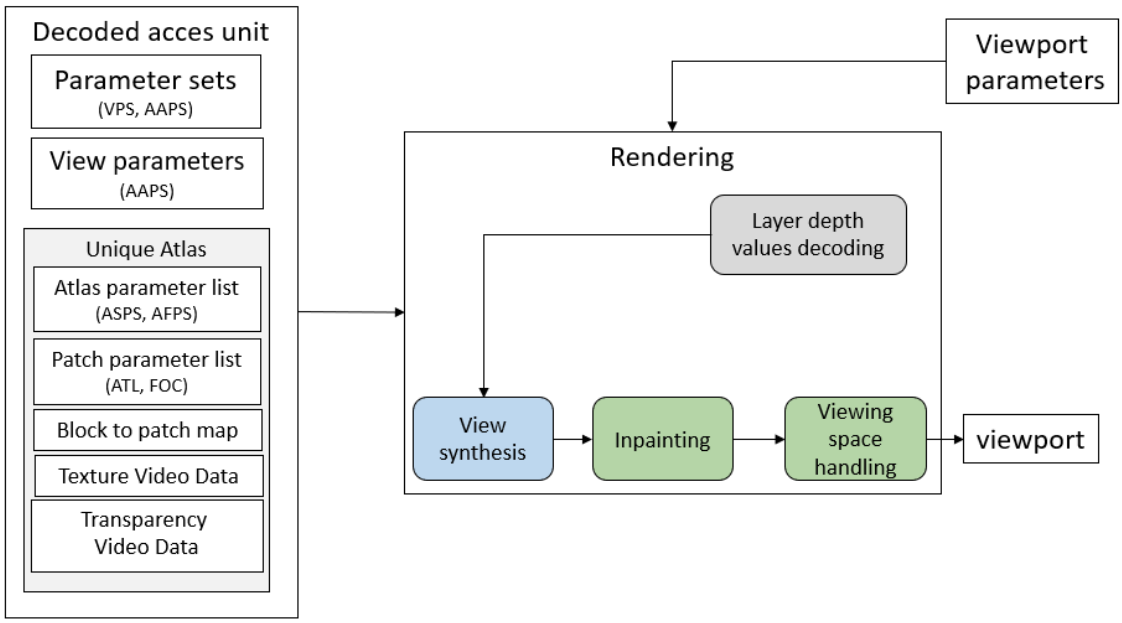

Figure 52: Process flow for TMIV MPI renderer

References

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

[2] *Text of ISO/IEC DIS 23090-12 MPEG immersive video 2nd edition*, ISO/IEC JTC1/SC29/WG04 N0533, April 2024, Sapporo, Japan.

[3] *Common test conditions for MPEG immersive video*, ISO/IEC JTC1/SC29/WG04 N0577, October 2024, Antalya, Turkey.

[4] *Manual of Immersive Video Depth Estimation,* ISO/IEC JTC1/SC29/WG04 N0058, January 2021, Online.

[5] *Manual of Depth Estimation Reference Software (DERS 9.0),* ISO/IEC JTC1/SC29/WG11 N19143, Jan. 2020, Brussels, Belgium.

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

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

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

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

[10] E. Penner and L. Zhang, *Soft 3D Reconstruction for View Synthesis*, ACM Transactions on Graphics (Proc. SIGGRAPH Asia) (vol. 6), 2017

[11] Wikipedia: *Painter's algorithm*. Variants: Reverse painter's algorithm

[12] *V-PCC Codec Description*, ISO/IEC JTC1/SC29/WG7 N00012, October 2020

Annex A. Reference software and MIV website

A.1. Availability and use

The reference software (TMIV-SW) including manual is publicly available on the Gitlab server[6]

A.2. Software coordination

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

- Bart Kroon, bart.kroon@philips.com

A.3. MIV Website

The MIV website is publicly accessible at https://mpeg-miv.org which includes an overview, use cases, sample datasets, and a list of related publications and demos.

[6] 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[7] as shown in Figure 53. 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.

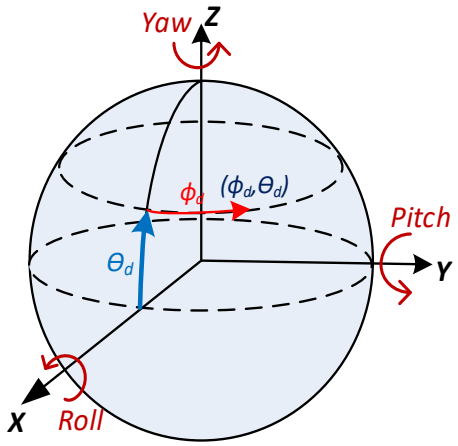

Figure 53: OMAF coordinate system illustrating the directions of positional and rotational units

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:

[7] 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[8]. 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[9] (q) to represent camera extrinsics.

The sequence configuration files and *pose traces* use Euler angles which are converted directly upon loading[10]. *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 $\boldsymbol{t} = q_{output}(\boldsymbol{t}_{input} - \boldsymbol{t}_{output})q^*_{output}$.

[8] https://en.wikipedia.org/wiki/Atan2
[9] https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
[10] https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles