ISO/IEC JTC 1/SC 29/WG 4 **N0050**

ISO/IEC JTC 1/SC 29/WG 4
MPEG Video Coding
Convenorship: CN

Document type:	Output document
Title:	Test Model 8 for MPEG Immersive Video
Status:	Approved
Date of document:	2021-01-29
Source:	ISO/IEC JTC 1/SC 29/WG 4
Expected action:	None
Action due date:	None
No. of pages:	49
Email of Convenor:	yul@zju.edu.cn
Committee URL:	https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg4

Test Model 8 for MPEG Immersive Video

Editors: Basel Salahieh, Joel Jung, Adrian Dziembowski, Christoph Bachhuber

Abstract

The Video working group has established the Draft International Standard and the 8th Test Model for MPEG Immersive Video during the 133rd MPEG meeting (January 2021) after evaluating the core experiment results and related contributions. The test model consists of this document and the reference software, providing an encoder and decoder/renderer in alignment with the specification. This document serves as a source of general tutorial information on the MPEG Immersive Video (MIV) design. It defines terminology used, process and data flow, operating modes, and description of algorithmic components adopted by the video group for the test model.

1. Introduction

The MPEG-I project (ISO/IEC 23090) on *coded representation of immersive media* includes Part 2 *Omnidirectional MediA Format* (OMAF) version 1 published in 2018 that supports 3 Degrees of Freedom (3DoF), where a user's position is static but his/her 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 (DIS) of MPEG Immersive Video [2]. The TMIV reference software (Appendix 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] provides test conditions including TMIV-based anchors.

3. Terms and Definitions

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

Table 1. Terminology definitions used for TMIV

Term	Definition
Additional view	A source view that is to be pruned and packed in multiple patches.
Basic view	A source view that is packed in an atlas as a single patch.
Clustering	Combining pixels in a pruning mask to form patches.
Culling	Discarding part of a rendering input based on target viewport visibility tests.
Entity	An abstract concept to be defined in another standard. For example, entities may either represent different physical objects, or a segmentation of the scene based on aspects such as reflectance properties, or material definitions.
Entity component	A multi-level map indicating the entity of each pixel in a corresponding view representation.
Entity layer	A view representation of which all samples are either part of a single entity or non-occupied.
Entity separation	Extracting an entity layer per a view representation that includes the desired entity component.
Geometry scaling	Scaling of the geometry data prior to encoding, and reconstructing the nominal resolution geometry data at the decoder side.
Inpainting	Filling missing pixels with matching values prior to outputting a requested target view.
Mask aggregation	Combination of pruning masks over a number of frames, resulting in an aggregated pruning mask.
Metadata merging	Combining parameters of encoded atlas groups.

Term	Definition
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 additional views resulting in pruning masks.
Pruning mask	A mask on a view representation that indicates which pixels should be preserved. All other pixels may be pruned.
Source splitting	Partitioning views into multiple spatial groups to produce separable atlases.
Source view	Indicates source video material before encoding that corresponds to the format of a view representation, which may have been acquired by capture of a 3D scene by a real camera or by projection by a virtual camera onto a surface using source view parameters.
Target view	Indicates either perspective viewport or omnidirectional view at the desired viewing position and orientation.
View labeling	Classifying the source views as basic views or additional views.

4. Description of encoder processes

4.1. Introduction

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

1. Preparation of source material by:
    - Assessing the geometry (depth map) quality, 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 view or additional view.
2. Encoding of each group separately (using the associated subset of split source views).
3. 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.
4. Encoding video sub bitstream:
    a. HEVC encoding of video sub bitstreams (each separately) using HM. The presence of geometry video data (GVD), attribute video data (AVD) or occupancy video data (OVD) depends on the encoder configuration.
    b. VVC encoding of video sub bitstreams (each separately) using VVenC. The presence of GVD, AVD or OVD depends on the encoder configuration.
5. If packed video is enabled and video sub bitstreams are VVC encoded, then GVD, AVD, and OVD can be combined into packed video data (PVD).
6. Multiplexing to combine the formatted bitstream with the video sub bitstream into a single MIV-complied bitstream.

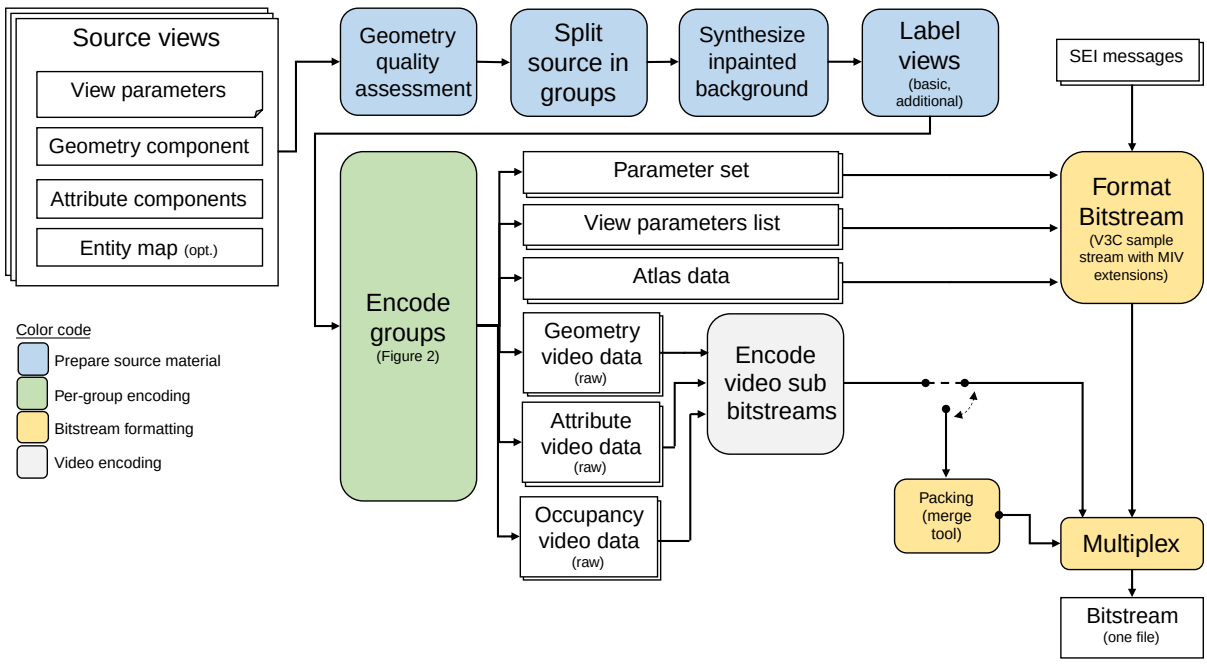

Figure 1. Top-level diagram of the TMIV group-based encoder

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

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

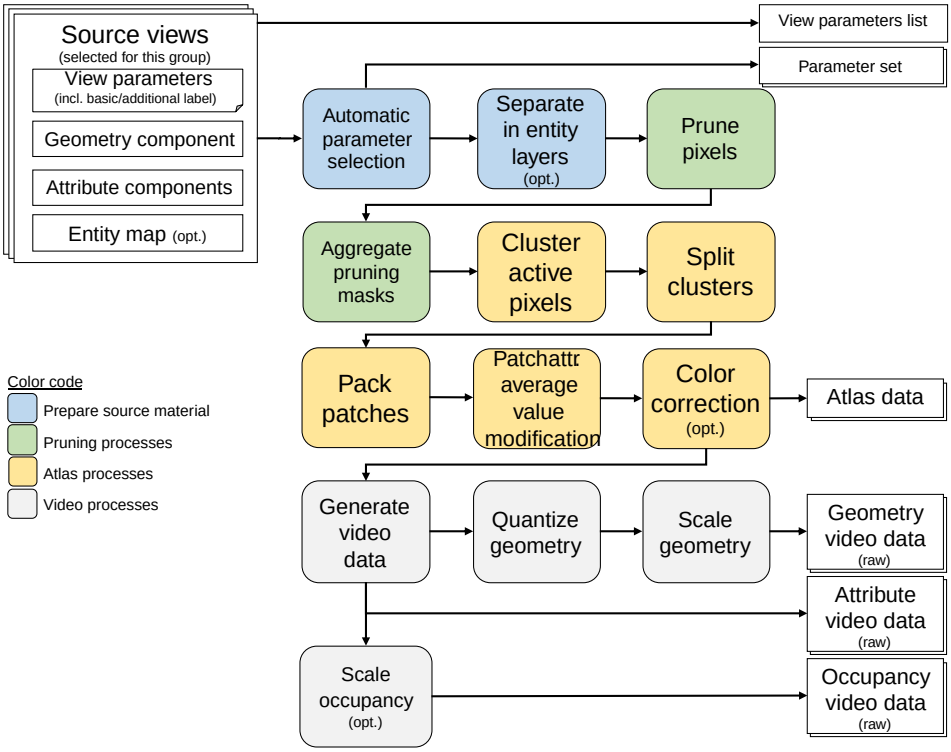

Figure 2. Top-level diagram of the TMIV single-group encoder

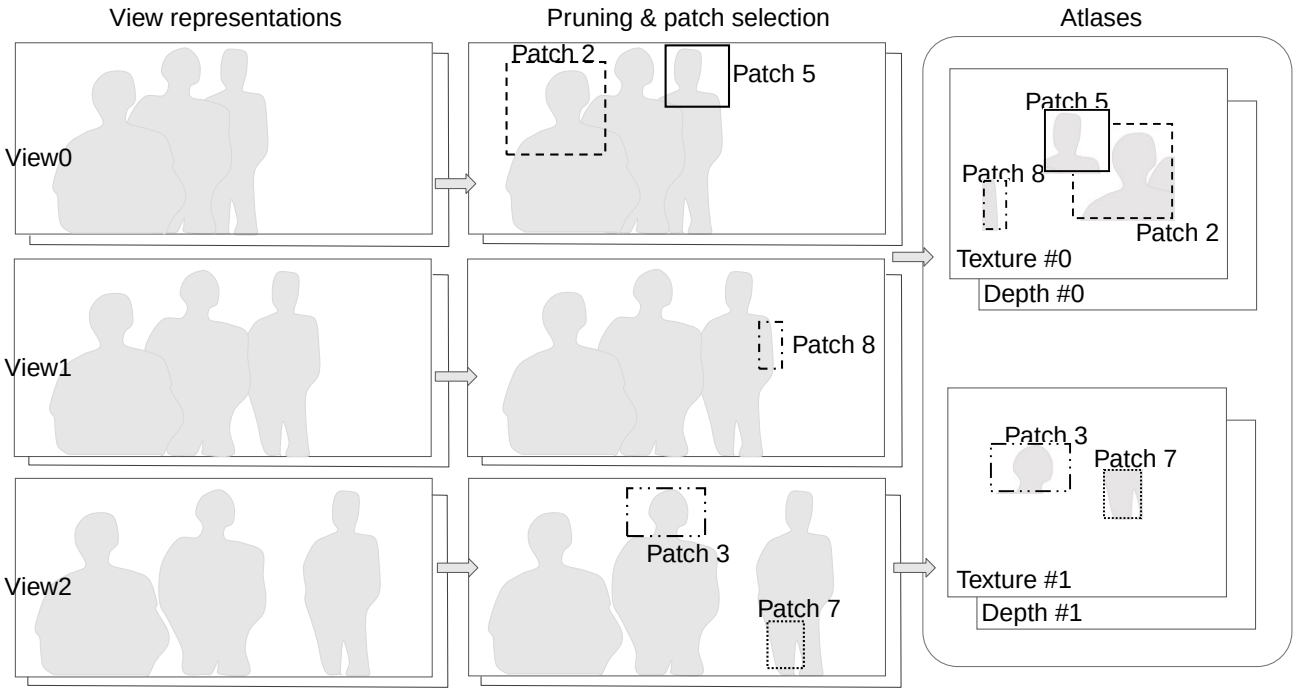

Figure 3. Representing source views using patch atlases

At the 132th MPEG meeting, multi-plane images [10] have been introduced to TMIV supporting an alternative coding mechanism that uses transparency layers.

4.2. Encoder inputs

The input to the TMIV encoder consists of a list of source views (Figure 4). The source views represent projections of a 3D real or virtual scene. The source views can be in equirectangular, perspective, or orthographic projection. Each source view should at least have view parameters (camera intrinsics, camera extrinsics, geometry quantization, etc.). A source view may have a geometry component in the form of 8-16 bits raw video with range/invalid sample values. Also a source view may have texture attribute component in the form of YC_BC_R 4:2:0 10 bits. 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.

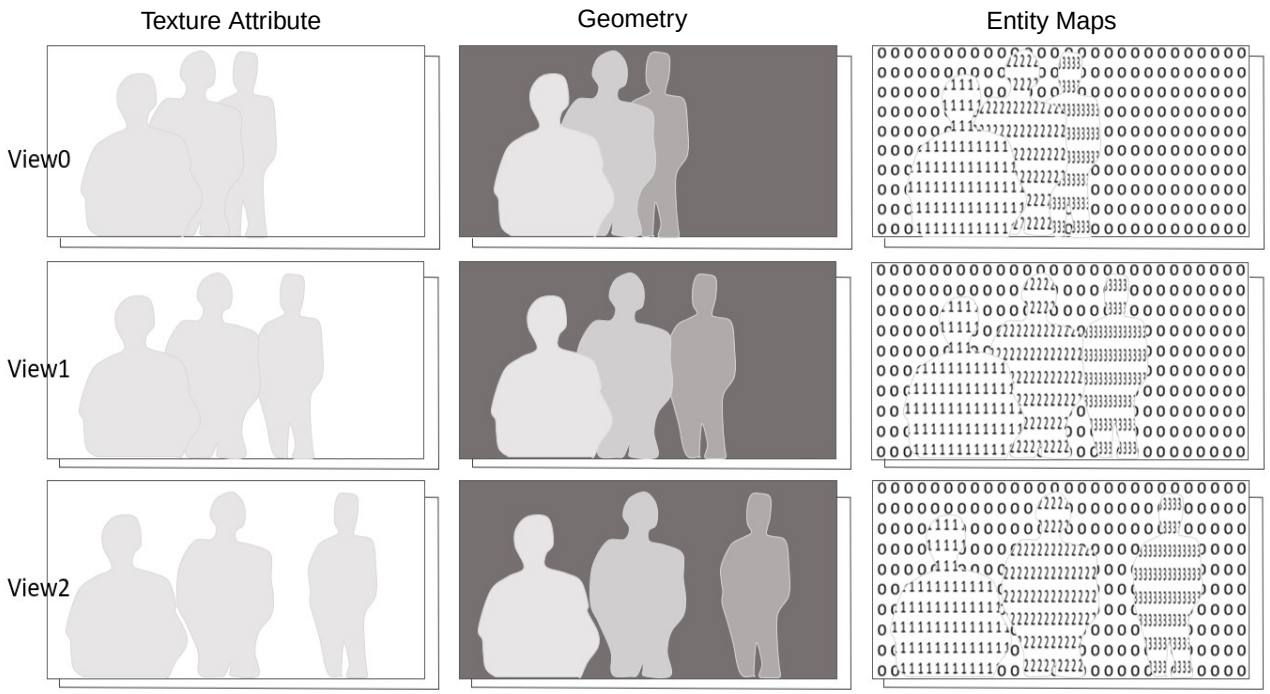

Figure 4. Input source views composed of texture attribute and geometry components, and entity maps

4.3. 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.
  - 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 or VVC sub bitstreams.
- TMIV also supports packed video data (PVD) V3C units that pack various video data types of various atlas tiles 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).

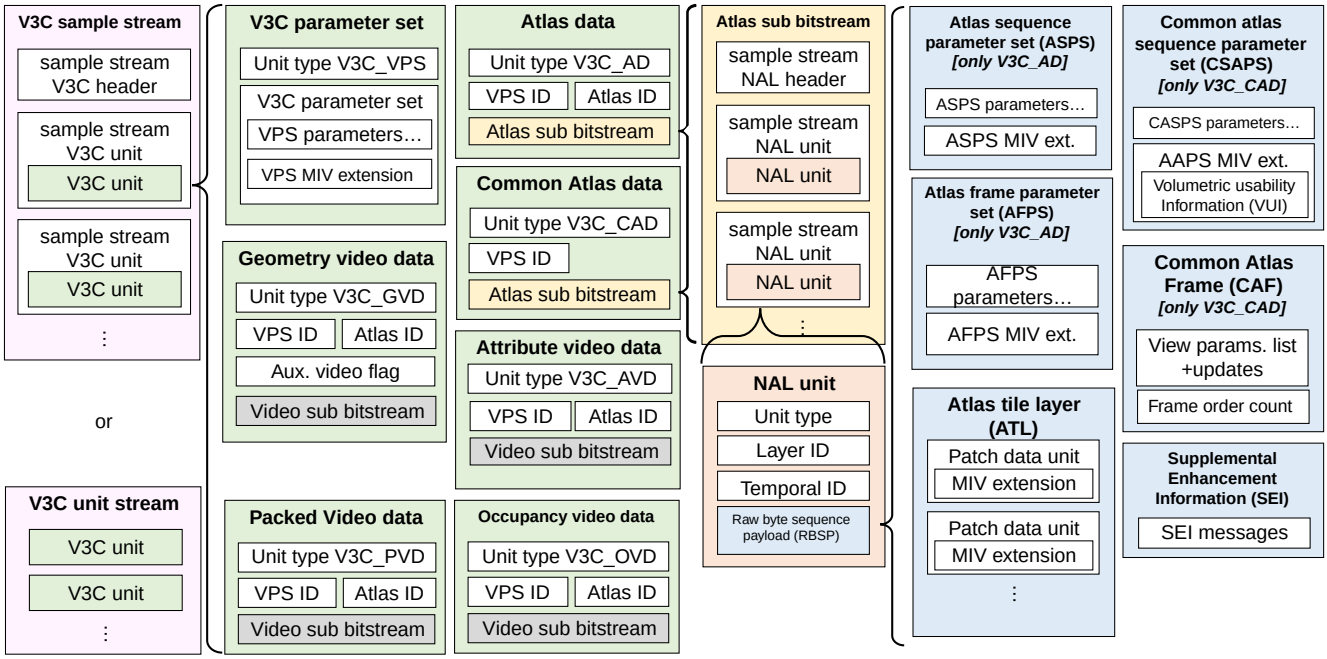

Figure 5. Structure of the V3C bitstream with MIV extensions. Some aspects of V3C that are not relevant to MIV have been omitted for clarity

4.4. 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 oppose to having fewer samples of those regions when processing all source views as a single group. An automatic process is implemented to select views per group, based on the view parameters list and the number of groups to obtain. The source views are being distributed accordingly in multiple branches, and each group is encoded independently of each other.

Source splitting operates as follows: a views pool including all available source views is formed and the number of views per group is set (by dividing the number of source views by the number of groups). The view parameters list is used to identify the range the views are spanning in Cartesian scene coordinates. The dominant coordinate axis is selected as a basis to set key positions. Key positions are located at the maximum view positions of the dominant axis across view in the views

pool. Distances of views to these key positions are computed. Based on the number of views for the group, the closest views to the first key position are selected and removed from the views pool. Then a second key position is identified and the process is repeated covering the distribution of all source views across the chosen number of groups.

4.5. Synthesizing an inpainted background view

This inpainting module is intended to create synthetic texture and geometry data that is hidden from the source views. Thereby reducing the missing data problem at the decoder-side. For this purpose it creates an ERP view with inpainted background data. This view has the following properties: It's position is the center, i.e. the mean of the source camera positions; It's field-of-view is the union of source view field-of-views (with some additional margin); Since the quality of the inpainted regions is lower than the original content, it's resolution is typically chosen to be lower than that of the source views. Hereby saving on bit- and pixelrate. This resolution is configurable.

The following steps are taken:

1. A background view is synthesized from all available source views. The 'RVS-based synthesizer', see Section 5.4.1 is used with a negated depthParameter, i.e. rendering background (when available) over foreground. Figure 6 illustrates this: The left image shows synthesis with a normal (positive) depthParameter where foreground is rendered over background. The middle image shows synthesis with a negated depthParameter where background is rendered over foreground. It de-occludes background region 'A' that is visible from the source views.
2. 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:
    a. 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. Actual depth values that are closer indicate relative foreground and actual depth values that are farther, indicate relative background.
    b. The relative foreground pixels are identified by comparison using some configurable threshold. They identify remaining foreground pixels that occlude 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
3. The masked texture and depth data are inpainted from neighboring areas. The 'PushPullInpainter' (Section 4.5.1) is used for this purpose.

The inpainted background view is identified by a Boolean flag so it is used for filling missing pixels at the decoder side. This flag is either located at the patch (pdu) level or at the view (mvp) level depending on the configuration.

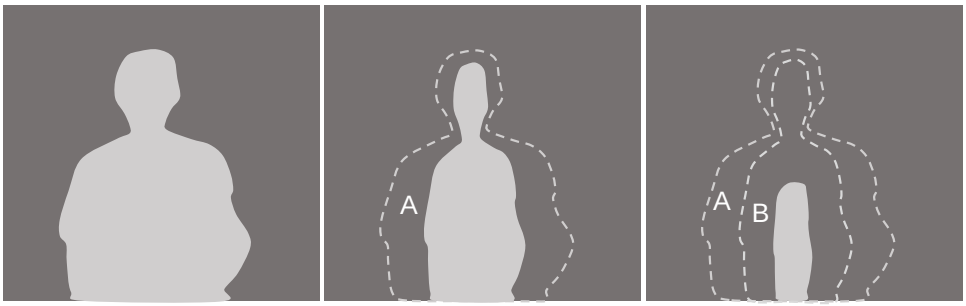

Figure 6. Steps in finding an inpaint mask for synthesizing an inpainted background view.

4.5.1. PushPullInpainter

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.

4.6. View labeling

The view labeling is split in two independent parts: view selection (Section 4.6.1) and basic view allocation (Section 4.6.2).

4.6.1. Two operating modes for view selection

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 (Section 4.6.2). 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 or additional views (Figure 7). The encoding result is one or more atlases with complete views and patches taken from the additional views.

Figure 7. View selection behavior of the view labeler when 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 additional views are not enabled

4.6.2. Basic view allocation

The labeling of basic views consists of the following steps:

1. Determine the number of basic views (hence “allocation”),
2. Prepare cost calculation,
3. Select initial basic views,
4. Update the view labels.

The inpainted view is labeled 'additional'.

4.6.2.1. Determine the number of basic views

In the data processing flow of the test model, the atlas frame size calculation (Section 4.7.2) is performed after view labeling[2]. 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. For instance, when this fraction is 50% and there are two atlases, then all basic views will fit in the first atlas.
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 + 2 *K*’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*.

Assumptions are:

1. The number of 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.

4.6.2.2. Prepare cost calculation

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

The cost function requires a distance metric on source views. While the previous view labeling method in TMIV 5 [WG11N19213] used viewport overlap as a measure of source view similarity, the current view labeler only uses the position of each source view to discriminate source views. The distance matrix is thus:

$$R = \lfloor r^2_{i,\ j} \rfloor$$

whereby $r^2_{i,\ j}$ is the squared distance m^2 between the source view positions.

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".

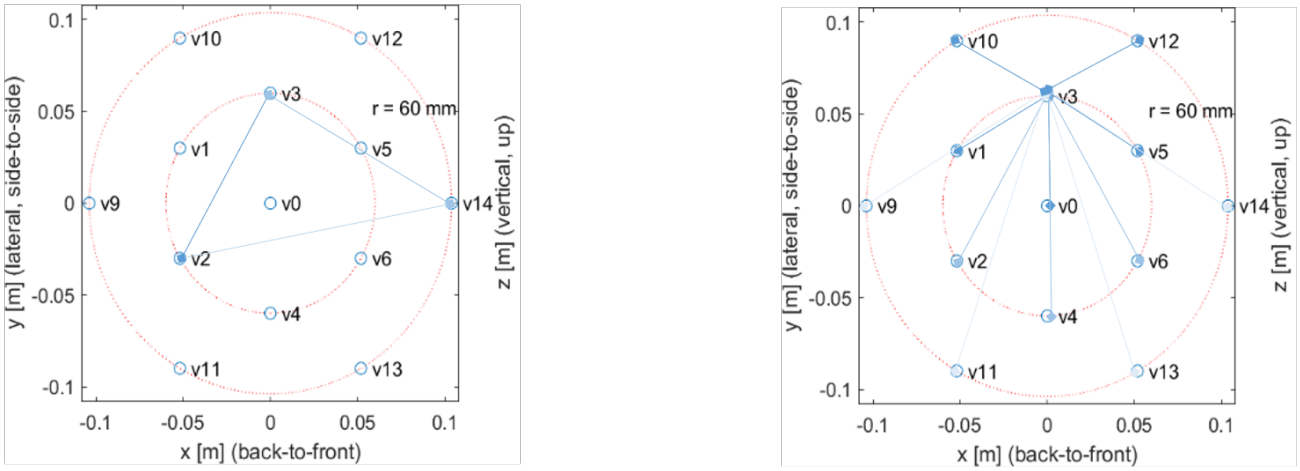

Figure 9. Repulsion of medoids v2 and v3 and v14 (left) and attraction of medoid v3 to non-medoids (right) for ClassroomVideo content

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

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

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

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

4.6.2.3. 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 $+x$ is the forward direction, which is the OMAF convention (cf. Section B.1). Subsequent medoids (if any) are selected one-by-one by adding the medoid that minimizes the repulsion cost.

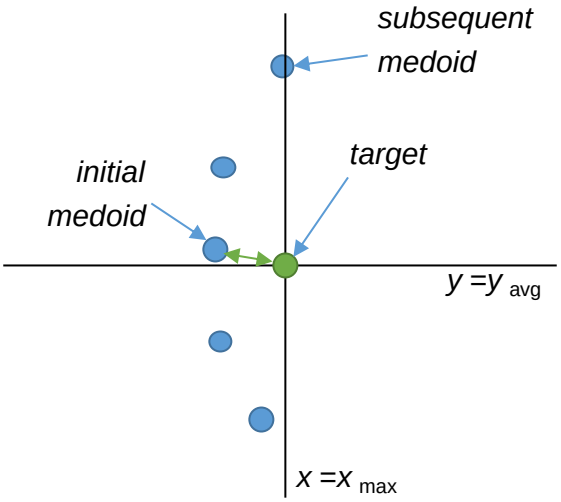

Figure 10. Initial basic view selection

4.6.2.4. 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.

4.7. 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.

4.7.1. 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.

4.7.2. 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.

4.8. 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 (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.

4.9. 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 attribute and geometry components and camera parameters, and outputting masks per view and frame of the same size. For additional 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,
- Maintain a realistic complexity,
- Consider temporal consistency.

4.9.1. 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.

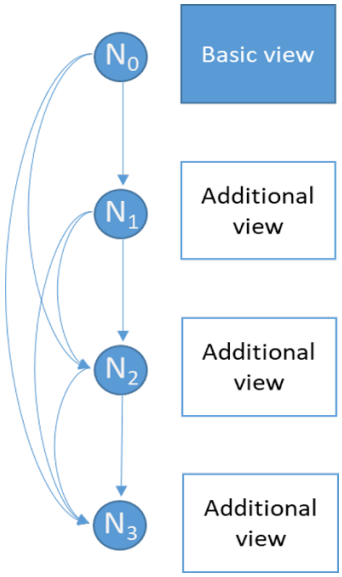

Figure 11. Pruning graph for one basic and three additional views. Basic view is assigned to a root node (node id: N_0), each additional view is assigned to a node N_i, which is a child node of all nodes N_j where $j < i$

Pruning graph creation:

1. Insert basic views into the pruning graph (as root nodes).
2. Project all pixels of all basic views to each additional view.
3. Create the *pruning mask* for each additional view (cf. Section 4.9.3).
4. Select the additional view with maximum number of preserved pixels (to prefer larger patches).

5. Insert selected additional view into the pruning graph (as a child node of all nodes already in graph) and stop if all the views are assigned to nodes in the pruning graph.
6. Project all preserved pixels of selected view to remaining additional views.
7. Update the *pruning mask* for each remaining additional view.
8. Go to 4.

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

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

4.9.2. 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 amount 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. 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 six basic views (yellow) and three additional views (white). The cluster graph consists of two disjoint graphs, and should be read like this:

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

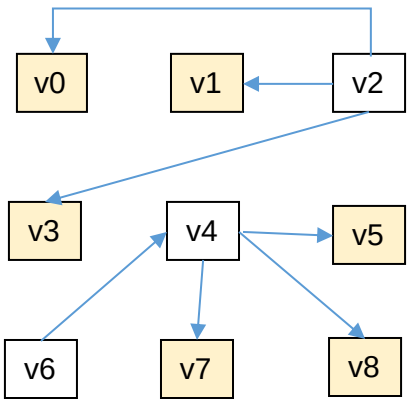

Figure 12. Cluster graph of SP, with notation

4.9.3. 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 4.9.4).

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.

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.

4.9.4. Pruning luma threshold calculation

The pruning luma threshold (Section 4.9.3) 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.

4.10. 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 13 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.

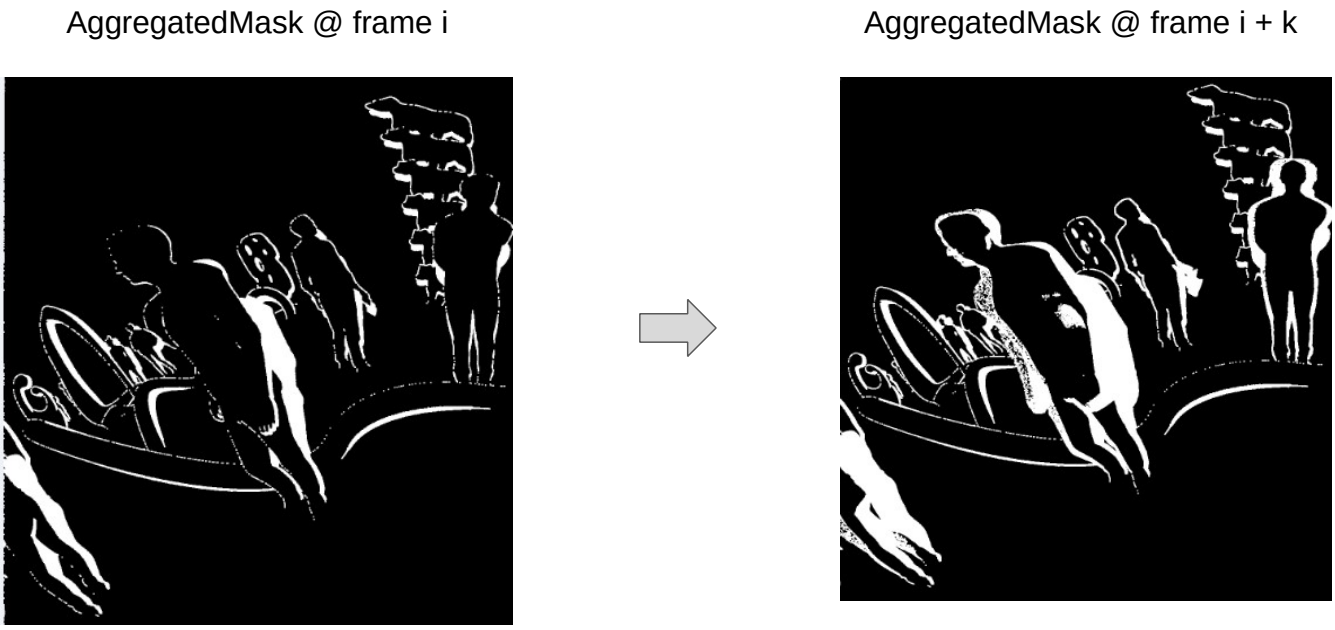

Figure 13. Aggregated mask evolution within an intra period

4.11. Clustering active pixels

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

Figure 14. Eight-pixel neighborhood for defining the connectivity criteria for region growing

An example of the clustering is illustrated in Figure 15 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.

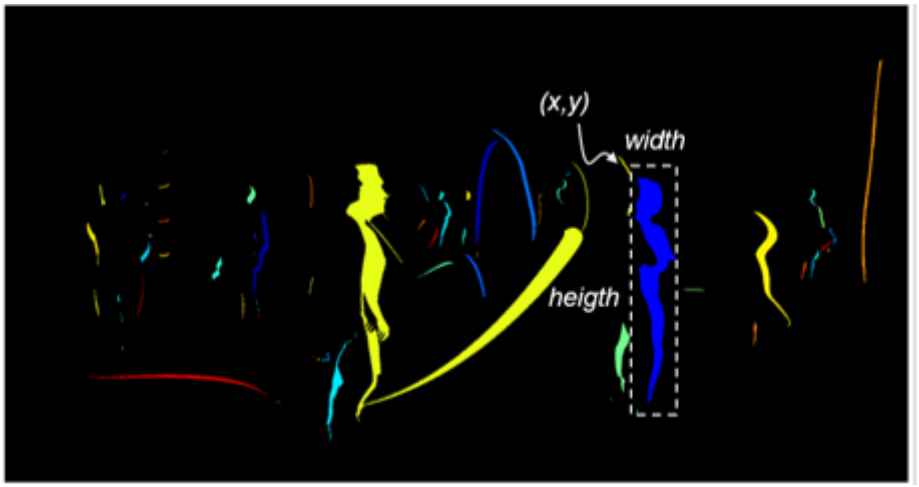

Figure 15. Clusters represented in false color on a pruned view

4.12. 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 16). 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 amount of associated parameters. As depicted in Figure 16, by merging the clusters *a* and *b* only two patches are generated instead of three.

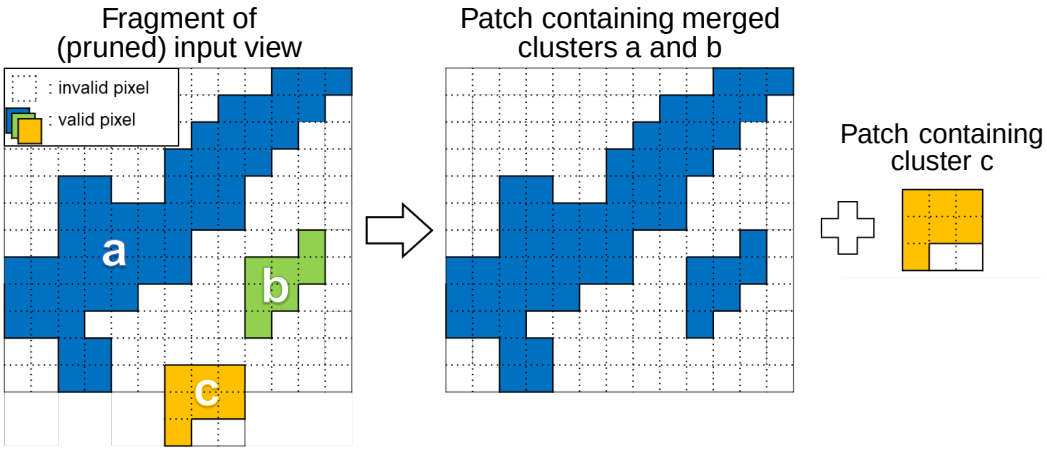

Figure 16. An example of cluster merging

4.13. Cluster splitting

In order to reduce spatial redundancy of data in the atlas, irregularly-shaped clusters (e.g. large yellow cluster in Figure 15) 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 17). Within the entire bounding box of the cluster, the number of blocks (cf. Section 4.14) 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.

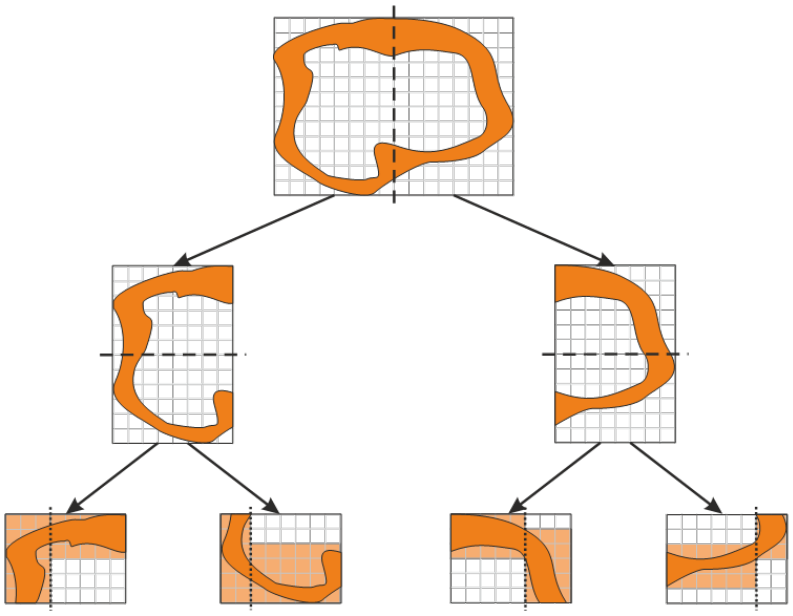

Figure 17. Recursive splitting of the cluster; dashed lines: C-splitting, dotted lines: L-splitting

4.14. Patch packing

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

- "*BlockSize*": the patch size and the patch position are multiple of the block size (number of pixels). Default value is 8.
- "*MinPatchSize*" is the number of pixels of the smallest border of the patch, below which the patch is discarded. Default value is 8.
- "*Overlap*" is the number of pixels which will be added to a frontier of a newly split patch; it prevents seam artefacts. Default value is 1.
- "*PiP*" is a flag enabling the Patch-in-Patch feature when equal to 1. It allows the insertion of patches into other patches. Default value is 1.
- "*sortingMethod*" is an integer indicating the method to be used to sort the clusters (0: by decreasing area, 1: by ascending view index). Default value is 0.
- "*enableRecursiveSplit*" is a flag enabling the use of the recursive split. Default value is true.

The 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, "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 MinPatchSize:
            Discard the patch
          Else:
            Put the part in the cluster priority list
```

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

- The 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 view 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 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 signalling 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 views. They are never split, rotated or flipped because an appropriate number of basic views (Section 4.6.2.1) and suitable atlas frame size (Section 4.7.2) are calculated. Also, because basic views often have the same number of active pixels, the ordering of clusters may be arbitrary. Clusters with the same number of active 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 18. The sample in the top-left corner of is the reference for specifying the position.

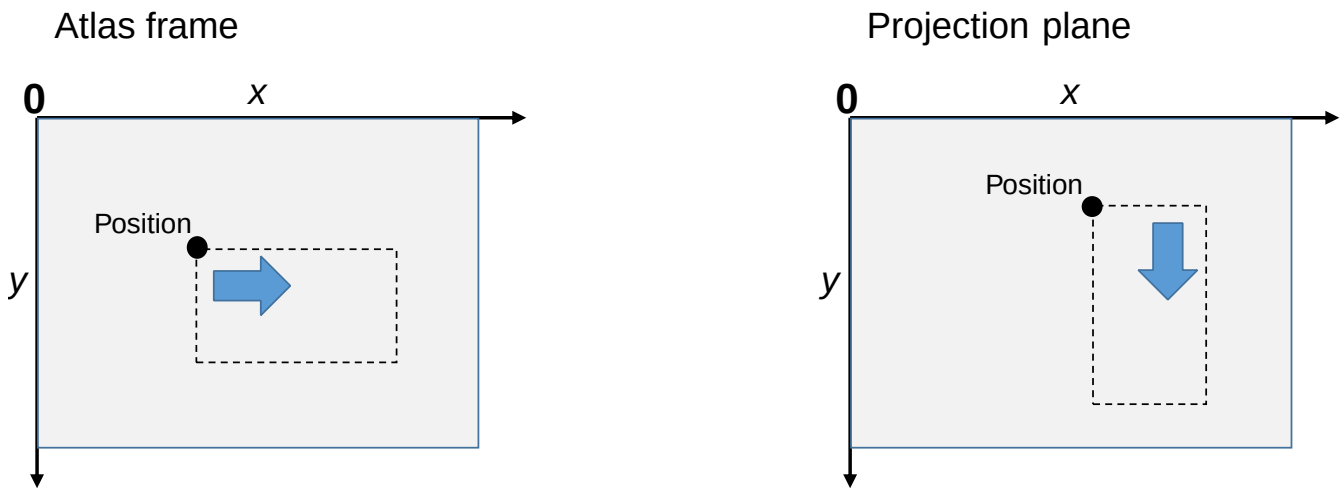

Figure 18. Definition of 90° patch rotation

4.15. Patch average value modification

After packing 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, e.g. 512 for 10bps video (Figure 19). The patch attribute offsets are added in order to restore the original attribute values at the decoder side are sent within atlas data.

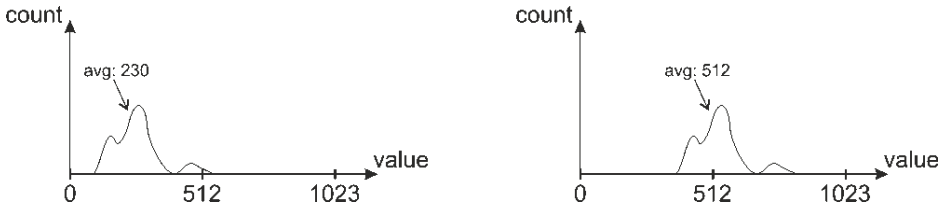

Figure 19. Histogram of an attribute component of a patch: before (left) and after (right) average value modification

If changing the average value to 512 causes overflows (pixel exceed the range [0, 1023]), the new average value is set according to the the size of the overflow (Figure 20).

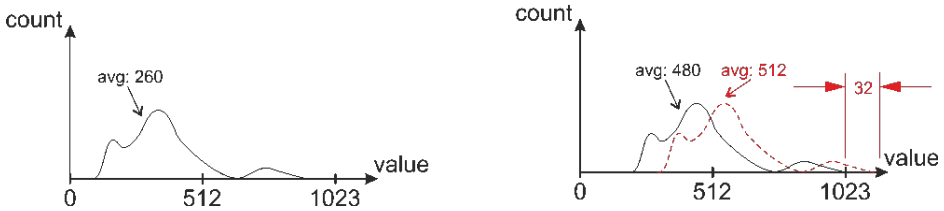

Figure 20. Histogram of an attribute component of a patch: before (left) and after (right) average value modification (overflow avoiding)

4.16. 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 attribute component).

Then, the patch attribute offsets (Section 4.15) sent within atlas data are modified by subtracting the color correction offsets averaged over the entire patch.

4.17. 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 (attribute and geometry) being written to the patches within the formed atlases.

Figure 21 illustrates the generation of an atlas, with the successive write of patch 2, 5 and 8. While the packing algorithm is using the information of samples that are mandatory and are non-pruned (represented by area inside the perimeters in dash), the copy of the patch is rectangular, resulting in a heap of possibly overlapping rectangles.

The occupancy of these rectangles is set separately for each frame by analyzing non-aggregated pruning mask. For a block size N, the mask is dilated iteratively 2N times using a 3 × 3 structuring element. A pixel of a patch is copied to the atlas if there is any non-zero value in a collocated N×N (cf. Section 4.13) block of dilated pruning mask. Otherwise, it is filled using neutral attribute and its geometry is set to zero, expressing the invalidity of a sample.

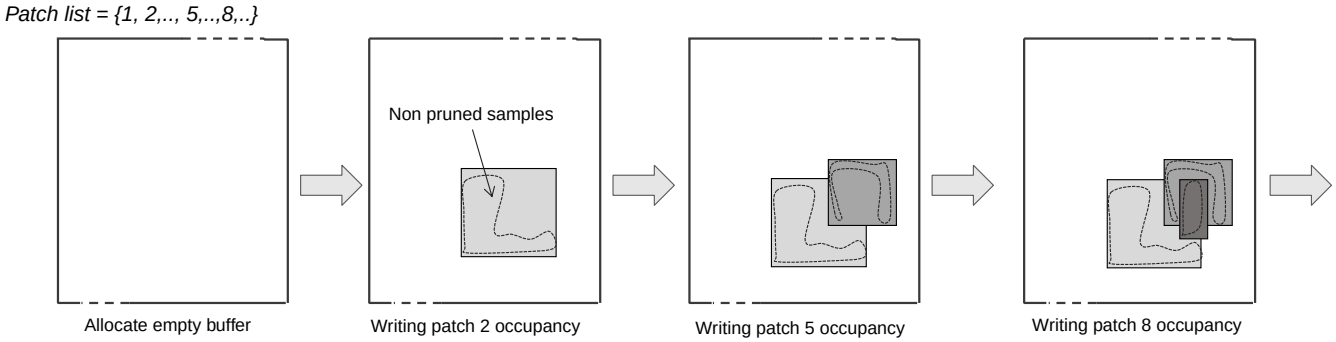

Figure 21. Successive writing of patches into an atlas

4.18. Geometry coding

An atlas value is either "invalid/non-occupied" or a geometry value expressed in meters, with maximum geometry value set to 1 km. The Draft International Standard [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 10 bits full range geometry atlases, the transformation is described in pseudo-code as:

```
valid := x >= depthOccMapThreshold
if (valid) {
  normDisp := max(1/kilometer,
    normDisp_0 + (normDisp_1023 - normDisp_0) * (max(depthStart, x) / 1023))
  depth := 1 / normDisp
}
```

Line 1 is part of the block to patch map decoder [2], lines 3...5 are part of the Synthesizer (Section 5.3) 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 22 and Figure 23:

- When a source view has only valid geometry values, depthOccMapThreshold is set to zero. This effectively encodes full occupancy (Figure 22).
- 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 23).

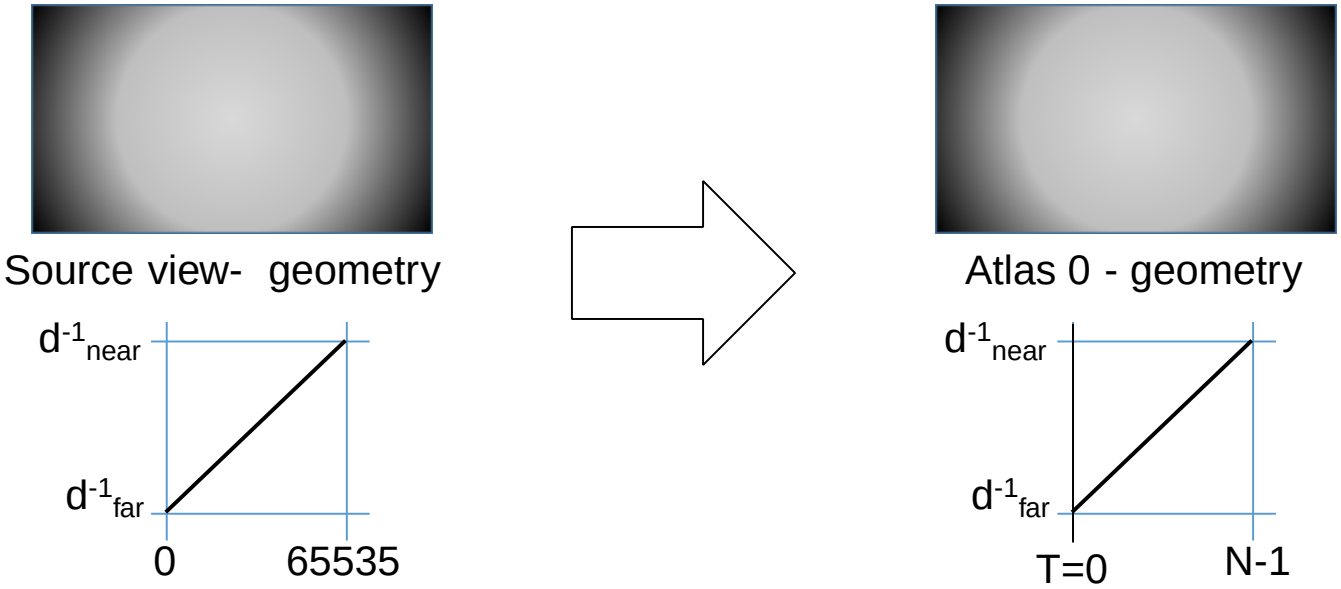

Figure 22. When the source material has only valid geometry values, the geometry coder only performs u(16) to u(10) or u(9) scaling and the geometry threshold is set to zero to signal full occupancy; N = 1024 if the geometry has a good quality (Section 4.7.1) and 512 otherwise

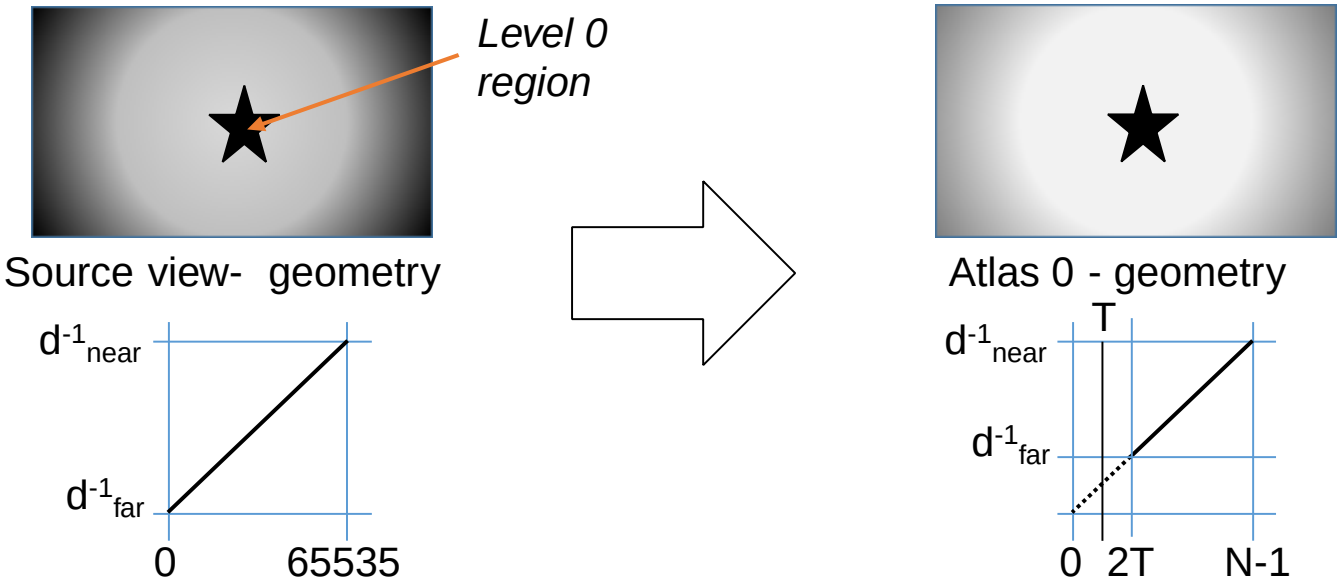

Figure 23. When the source material has invalid geometry values, the geometry coder not only performs u(16) to u(10) or u(9) scaling, but it also sets the geometry threshold to a configurated value (T) and the normalized disparity range is modified such that value 2T corresponds to the far geometry; N = 1024 if the geometry has a good quality (<secGeometryQuality>>) and 512 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 (Section 4.7.1), 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:

$$d_{\text{near}}^{-1} = \max_{\text{allpixels} \in \text{GOP}} d^{-1}(\text{pixel})$$

$$d_{\text{far}}^{-1} = \min_{\text{allpixels} \in \text{GOP}} d^{-1}(\text{pixel})$$

4.19. 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.

4.20. 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 Section 4.17). 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.

4.21. Video encoding

MIV is agnostic to the video codec and the test model is designed to work with external video coding tools. However, the default codec for all video sub bitstreams is HEVC with Main 10 profile and random-access configuration, and the TMIV decoder is capable of decoding HEVC sub bitstreams by inclusion of the HEVC Test Model (HM). VVC can also be used as an external video coding tool, using the Versatile Video Encoder (VVCenC).

4.22. Packing of video sub-bitstream components

A subpicture merging software[3] allows for encoding several VVC bitstreams separately, and merge selected encoded bitstreams into a single VVC compliant bitstream with multiple sub-pictures. Figure 24 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 25. 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 the same as the sum of the separate sub-picture bitstreams.

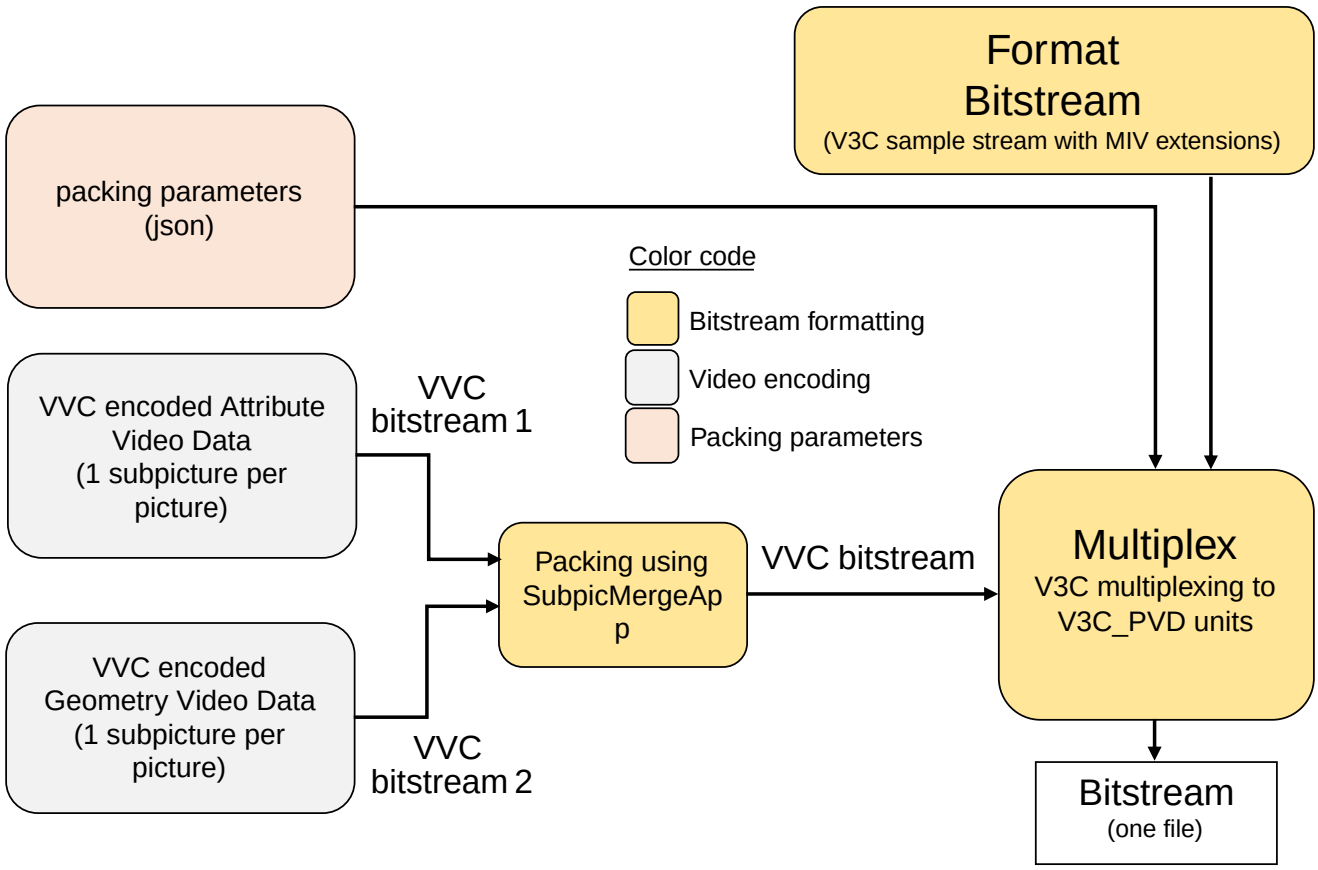

Figure 24. Encoding of texture and geometry components into one packed video sub-bitstream

Figure 25. An example one frame of classroom video sequence with packed atlases

4.23. Multi-plane image encoder

4.23.1. 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 26) 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. The reference camera is a spherical (typically equirectangular) camera when using spherical layers.

In the case of MPI, there is no view synthesized inside TMIV encoder. There is only one MPI camera at the input of 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.

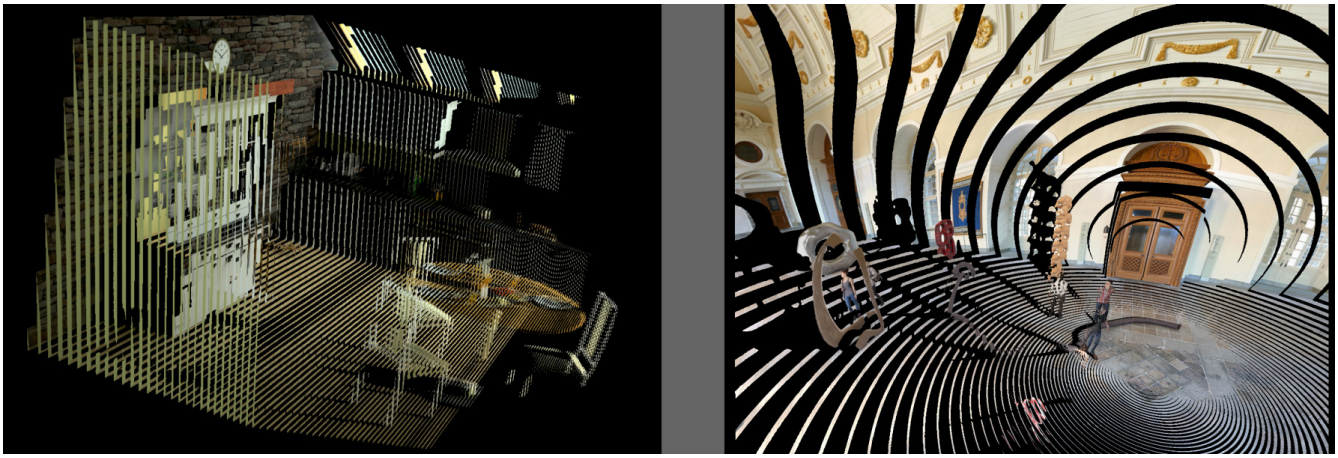

Figure 26. Texture MPI layers of different projections; perspective Kitchen (left) and equirectangular Museum (right)

4.23.2. 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.

4.23.2.1. Raw storage

Raw storage of an MPI (cf. Section 4.23.2.1) 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 27.

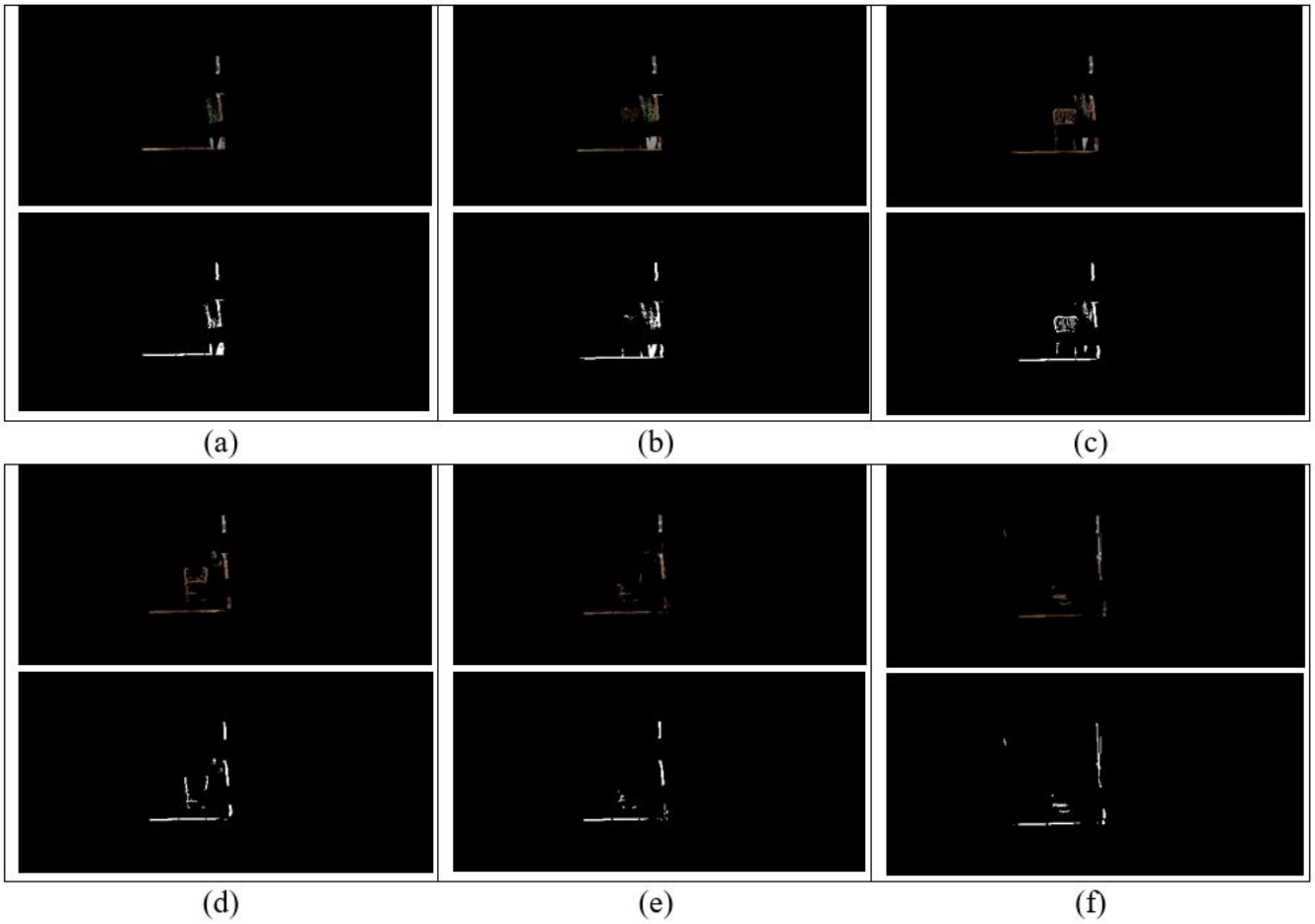

Figure 27. Example of MPI layers from Mpi_Fan, from layer 40 (a) to layer 45 (f). Texture is on the first row and transparency is on the second row.

Raw storage is defined as below:

- texture layers must be provided as a single file (YC_BC_R 4:2:0 10 bits from TMIV8.0-rc1),
- transparency layers must be provided as a single file (YC_BC_R 4:2:0 8 bits from TMIV8.0-rc1).

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. Raw storage is illustrated in Figure 28.

From TMIV8.0-rc1 implementation, transparency is coded on 16 levels only, hence 4 bits would be enough. But a YC_BC_R 4:2:0 8 bits format is used to ease file manipulation.

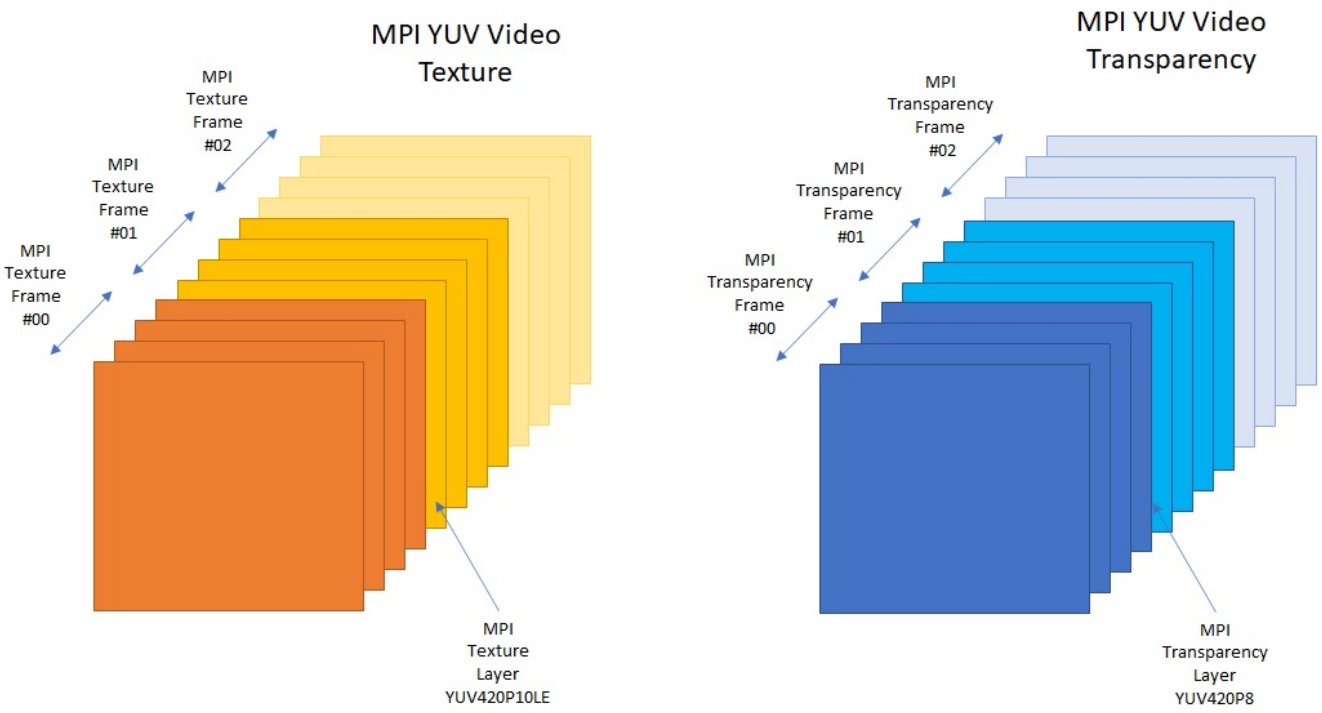

Figure 28. MPI raw storage diagram. In this simplified example, the MPI content is made of 4 layers (sorted from far to close) and 3 concatenated temporal frames.

4.23.2.2. 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.

Let's consider the reference camera with resolution W x H and S layers. 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), as a YUV400P16LE sample (2 bytes)
- $C_{i,j,k}$: the color of pixel (i,j) for the k-th active layer, as a YUV444P10LE sample (6 bytes)
- $D_{i,j,k}$: the layer index of pixel (i,j) for the k-th active layer, as a YUV400P16LE sample (2 bytes)
- $T_{i,j,k}$: the transparency of pixel (i,j) for the k-th active layer, as a YUV400P8 sample (1 byte)

where k is an integer in the range [1, $N_{i,j}$].

The W x H number of active layers per pixel are first provided as a regular YUV400P16LE frame in the bitstream, and the concatenated list of all samples is immediately stacked after as presented in Figure 29.

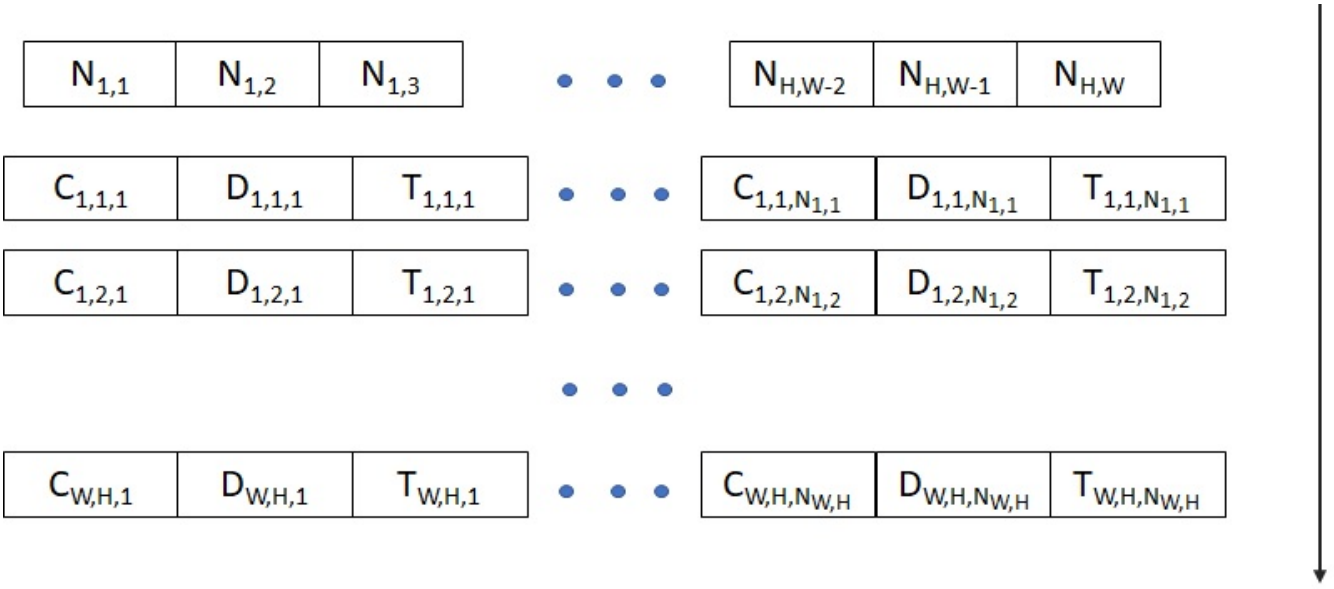

Figure 29. MPI packed compressed storage (PCS) layout.

When reading an MPI temporal frame, one first read the W x H YUV400P16LE 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 2 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 / index derived from the number of active layers per pixel.

4.23.2.3. 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 from TMIV8.0-rc1 for this conversion. Please refer to Section 4.23.2.1 for the expected format of an MPI content in raw storage.

4.23.3. 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 30) are presented below.

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 4.10 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 4.11, Section 4.12 and Section 4.13.

The obtained clusters are then packed using the process described in Figure 2, cf. Section 4.14, but the sorting operation is changed to get clusters from distant layers packed first rather than the biggest clusters (sortingMethod is set to 1 in Section 4.14). 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.

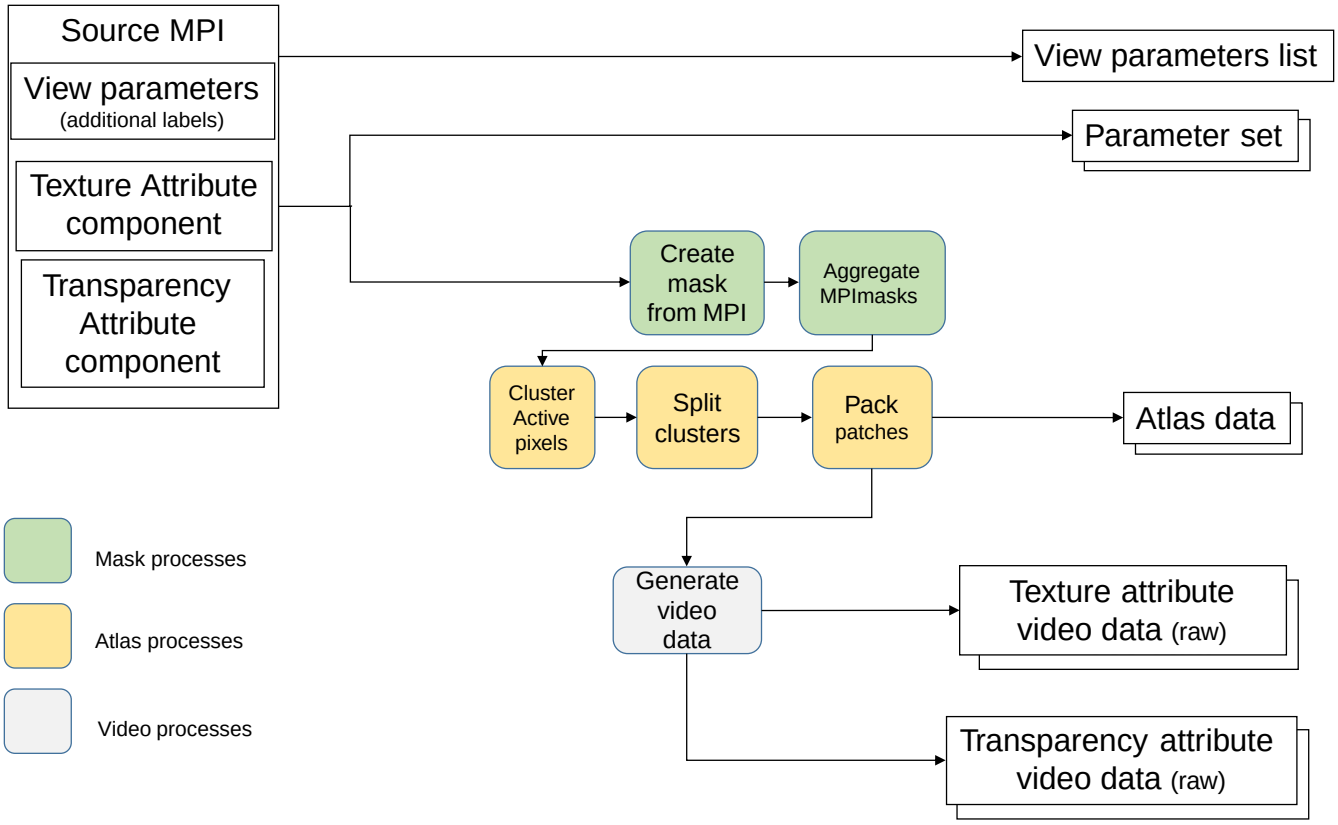

Figure 30. Top-level diagram of the TMIV MPI encoder

4.24. Bitstream formation and multiplexing

The output of the TMIV encoder is a V3C sample stream with MIV extensions (Figure 31). 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 is thus performed as follows:

1. Atlases of multiple groups are concatenated with renumbering of atlas ID's. Also, parameter set and atlas data of all groups are merged together into one parameter set and one atlas data units,

respectively.

2. An intermediate bitstream is formatted that includes no video sub bitstreams.
3. All geometry (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 verstaile videl 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.

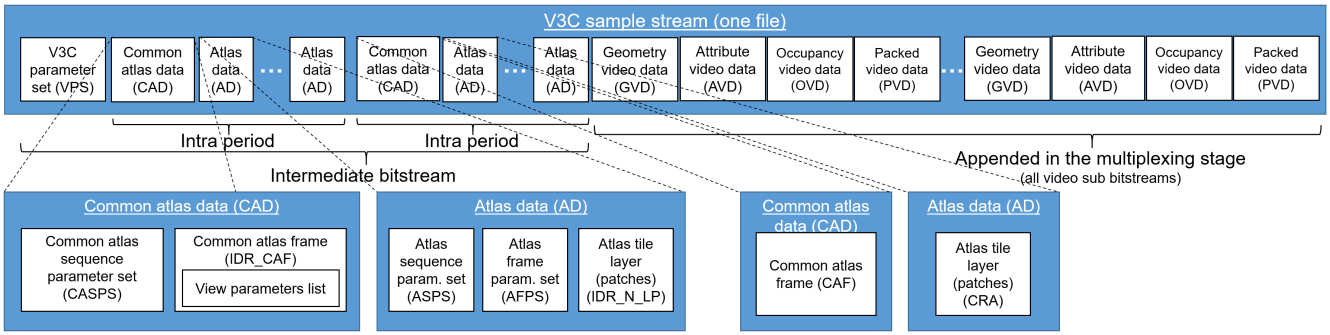

Figure 31. V3C sample stream with MIV extensions

5. Description of the rendering process

The TMIV decoder follows the MIV decoding process described in the committee draft [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 32) 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 in luma and chroma 4:2:0 format with 10 bits for attribute component and 16 bits for geometry component. It can in principle be displayed on either head mounted display (HMD) or on regular 2D monitor with tracking system feeding the updated *viewing position* and *orientation* back to the renderer for the next target view. More details on the coordinate systems, projections, and camera extrinsics can be found in Appendix B.

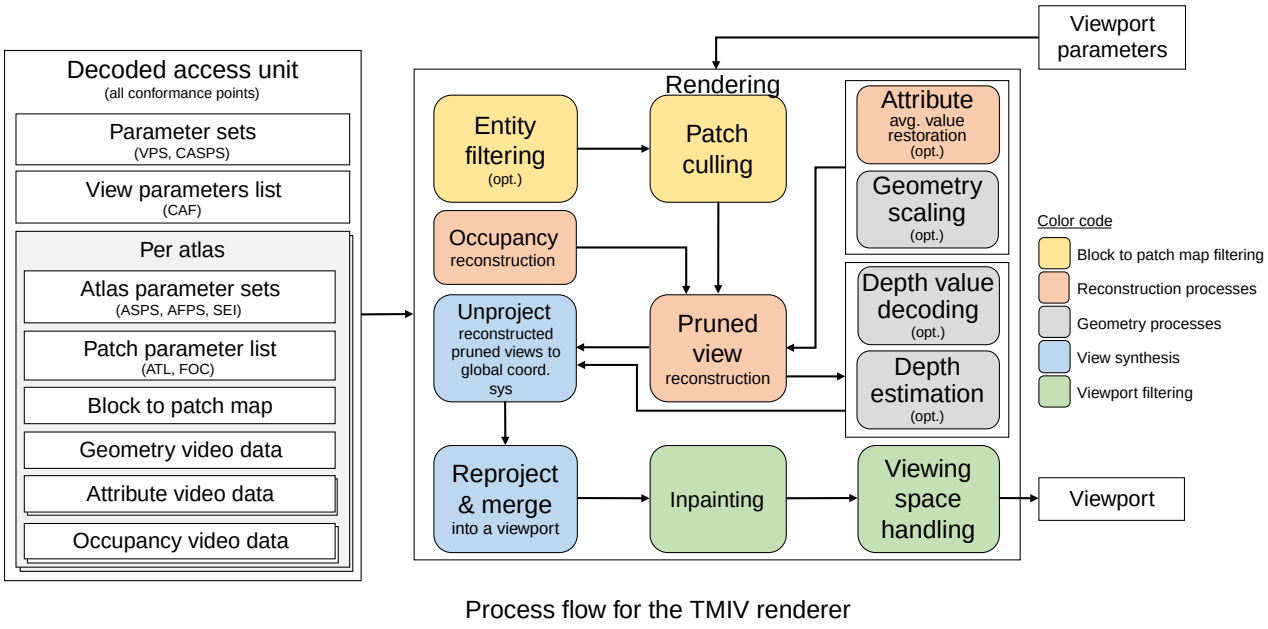

Figure 32. Process flow for the TMIV renderer

In addition to the regular TMIV rendering, support for MPI rendering is added to the software and described further in Section 5.6.

5.1. Block to patch map filtering

5.1.1. Entity filtering

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

5.1.2. 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 the area enclosed by the eight reprojected points $(P_i(x, y), i = 0, 1, ..., 7)$ has no overlap with the target viewport, the patch is culled. The patch map is updated as illustrated in Figure 33. If the patch is culled, the corresponding atlas's samples are labeled as unused and ignored during the rendering process.

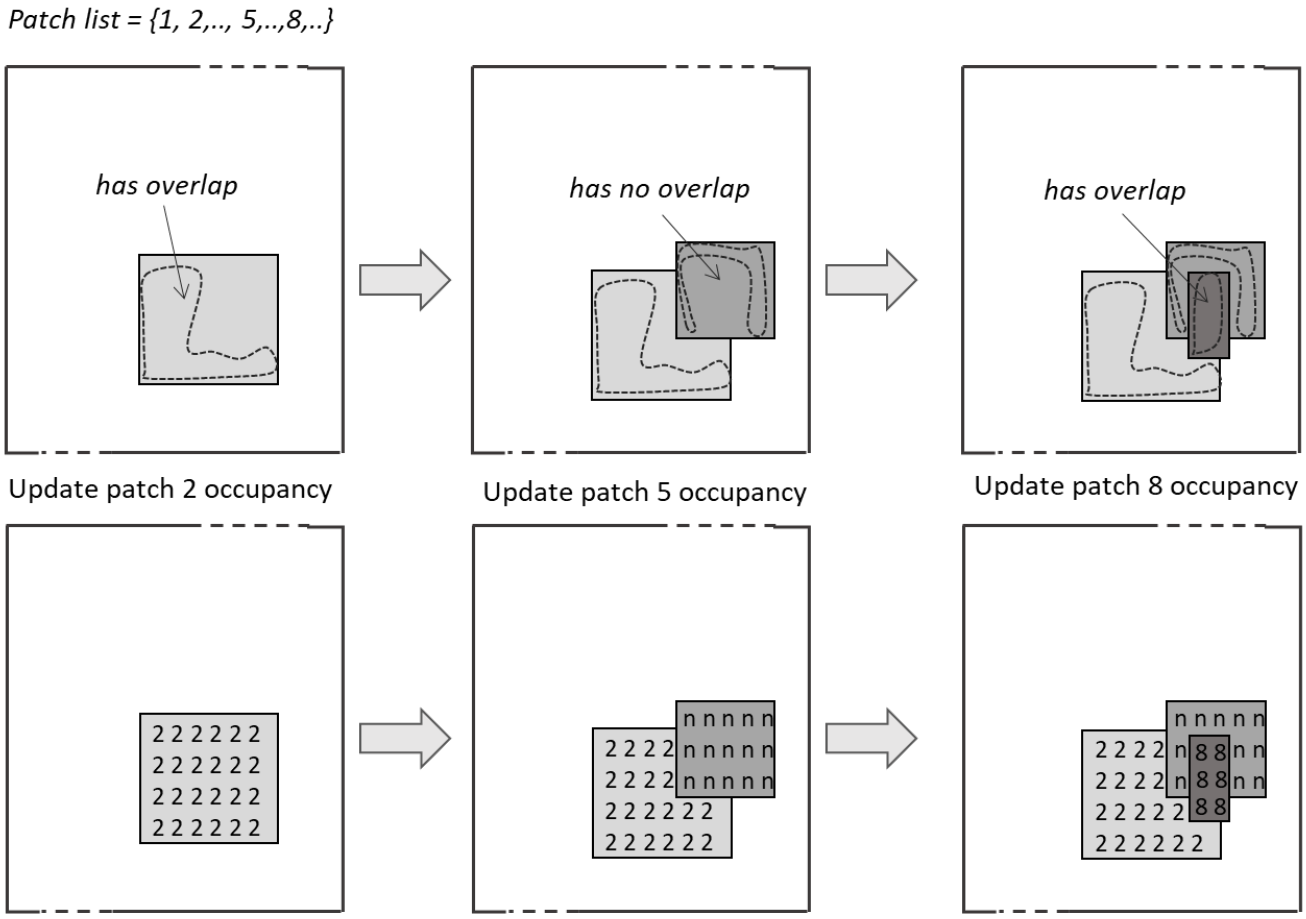

Figure 33. Occupancy map update in an ordered manner with patch culling

5.2. Reconstruction processes

5.2.1. 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 4.18. When smaller than the threshold, the occupancy value at the given pixel is set to 0 otherwise it is set to 1.
- When occupancy video sub bitstream is present (i.e. signaled explicitly), 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), the occupancy frame at nominal atlas resolution is filled with ones.

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

5.2.2. Attribute mean value restoration

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

5.2.3. Pruned view reconstruction

The reconstruction of the pruned views is an operation opposite to video data generation performed in the encoder (Section 4.17). 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 34: 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.

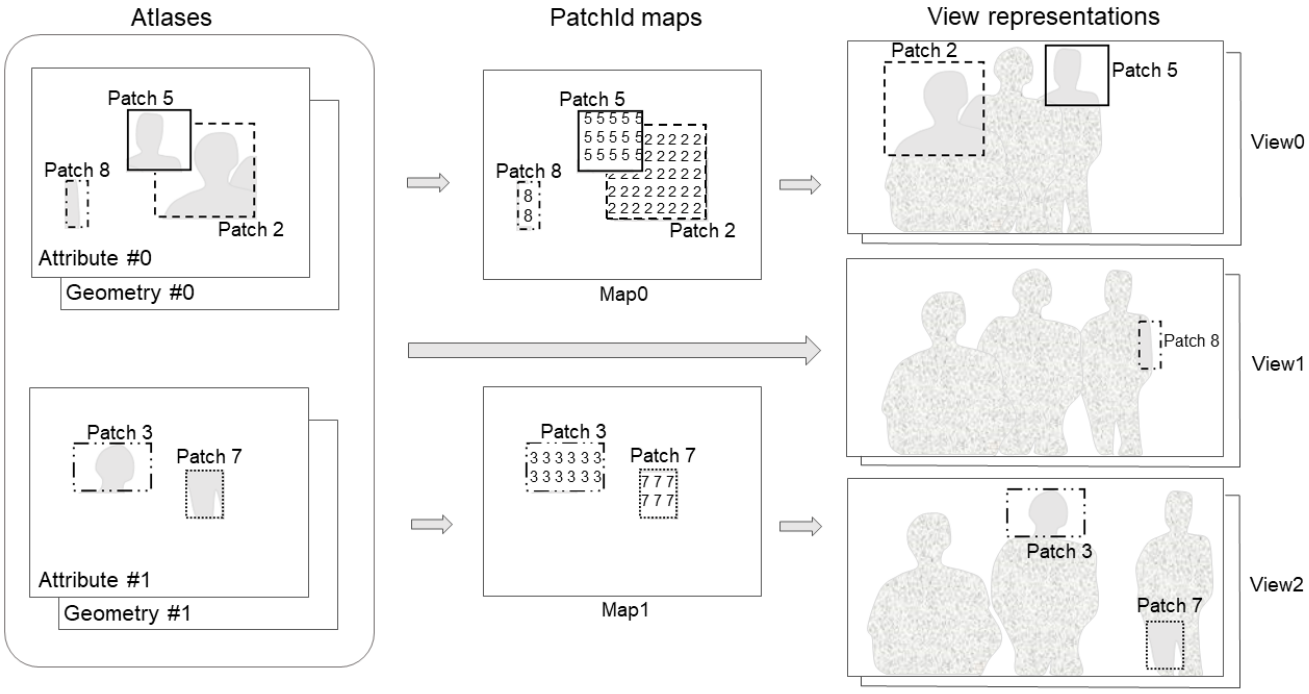

Figure 34. Pruned view reconstruction

5.3. Geometry processes

5.3.1. Geometry upscaling

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

5.3.2. Depth value decoding

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

5.3.3. Depth estimation

When TMIV encoder operates in the geometry absent profile, no geometry video sub bitstreams are present. Thus, 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.

5.4. View synthesis (unprojection, reprojection, and merging)

The TMIV proposes two alternatives for the synthesis. The first one is RVS-based [6], described in Section 5.4.1, the second one is the View Weighting Synthesizer (VWS), described in Section 5.4.2.

5.4.1. RVS-based synthesizer

5.4.1.1. Overview

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

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

The RVS-based synthesizer may synthesize directly from atlases, thus for this synthesizer there is no necessity of pruned view reconstruction (Section 5.2.3).

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

5.4.1.2. Rendering from atlases

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

- Multiple 10 bits attribute atlases and 10 bits geometry atlases (normalized disparities),
- Block to patch map per atlas,
- Parameters including an atlas parameters list and a camera parameters list,

- Target camera parameters for a *perspective viewport* or an *omnidirectional view*.

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

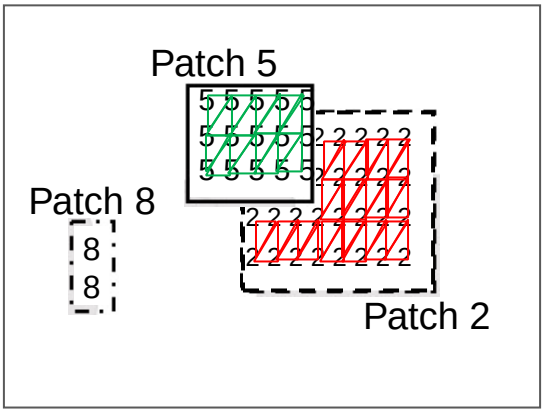

Figure 35. Creating a mesh from an atlas. Triangles between pixels from Patch 5 and 2 are omitted. Note that Patch 8 is not drawn because no triangle can be formed. Unused pixels are skipped too

The process is to build a mesh (Figure 35) 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 View 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 converted to YC_BC_R 4:4:4.
- The **triangle descriptor list** is formed by:
  - For each pixel consider two triangles [/]
  - Add the triangle when all vertices have the same Patch ID.

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

5.4.1.3. Pixel blending

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

$$I_{\text{blend}} = \sum_i w(\gamma_i, d_i, s_i) I_i$$

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

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.

5.4.2. View weighting synthesizer

5.4.2.1. Overview

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

- Visibility: this step aims at generating a geometry map for the target viewport. First a warped geometry map is generated for each input view, by unprojecting/reprojecting pixels from this view towards the target view. It uses splat-based rasterization [9] instead of triangulation. From the warped geometry maps, a single geometry map is generated, namely the visibility map. This selection process is based on a pixel-wise majority voting process which takes into account the weight of each view, described in Section 5.4.2.2. Finally, the visibility map is cleaned out using a post median filtering to remove outliers.
- Shading: this step aims at computing the target viewport color. Each input view's pixel is blended into the target viewport with a contribution/weight taking into account its consistency with the visibility map and the weight of the view it belongs to. Input contours are detected and discarded from the shading stage to avoid ghosting.

5.4.2.2. Weighting strategy

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

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

To check for the tridimensionality, a test on the singularity of the covariance matrix of the view positions is performed. The contribution of each pixel in the visibility and shading pass is thus weighted by the contribution of its associated view. However, when dealing with pruned input views, this information is incomplete and an additional step which makes use of the pruning information as defined in Section 4.9.1 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 36). 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.

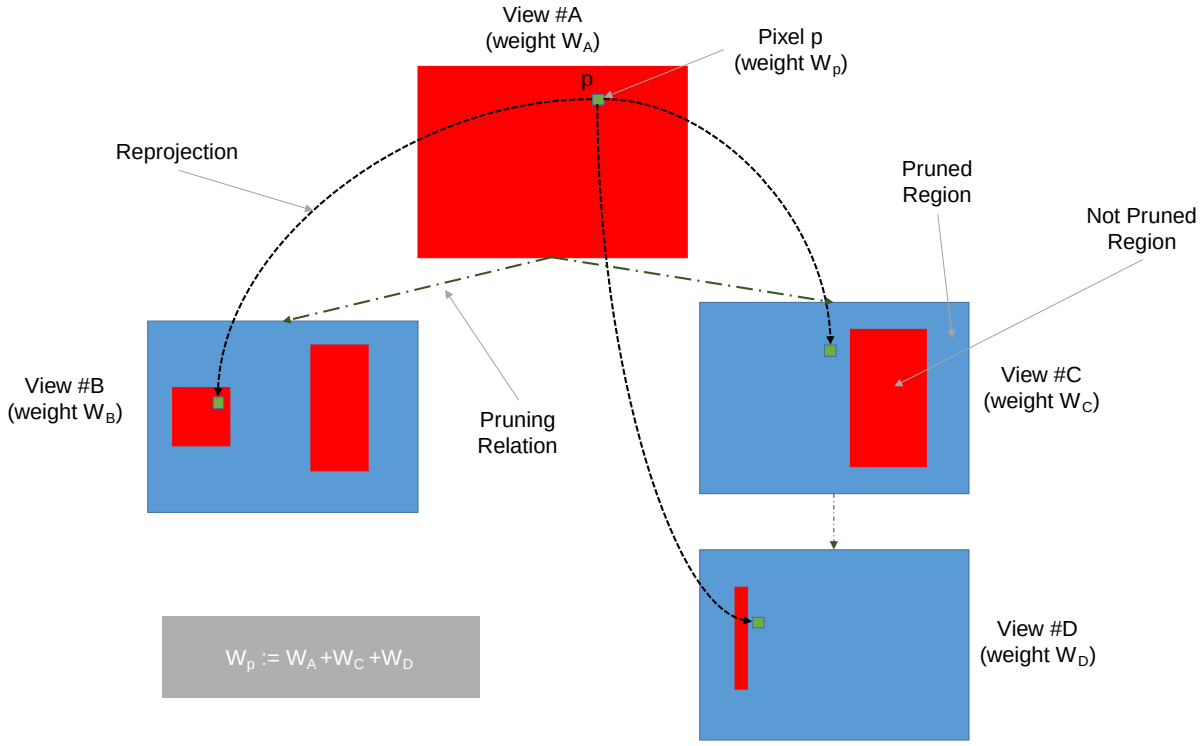

Figure 36. Graph-based pruning: weight recovery procedure

5.4.2.3. Handling of inpainted data

Some patches within the atlases belong to the inpainted background view presynthesized at the encoder side (see Section 4.5). They are of lower quality than the original source views. That pre-inpainted data is projected to the target viewport but otherwise exluded from the steps of making a visiblity 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.

5.4.2.4. Parameters

The parameters of VWS are presented in Table 3.

Table 3. Parameters of the view weighting synthesizer

Parameter	Type	Description
angularScaling	float	Drives the splat size at the warping stage.
minimalWeight	float	Allows for splat degeneracy test at the warping stage.
stretchFactor	float	Limits the splat max size at the warping stage.
overloadFactor	float	Geometry selection parameter at the selection stage.
filteringPass	int	Number of median filtering pass to apply to the visibility map.
blendingFactor	float	Used to control the blending at the shading stage.

5.5. Viewport filtering

5.5.1. 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 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 37a). 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 37b). 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.

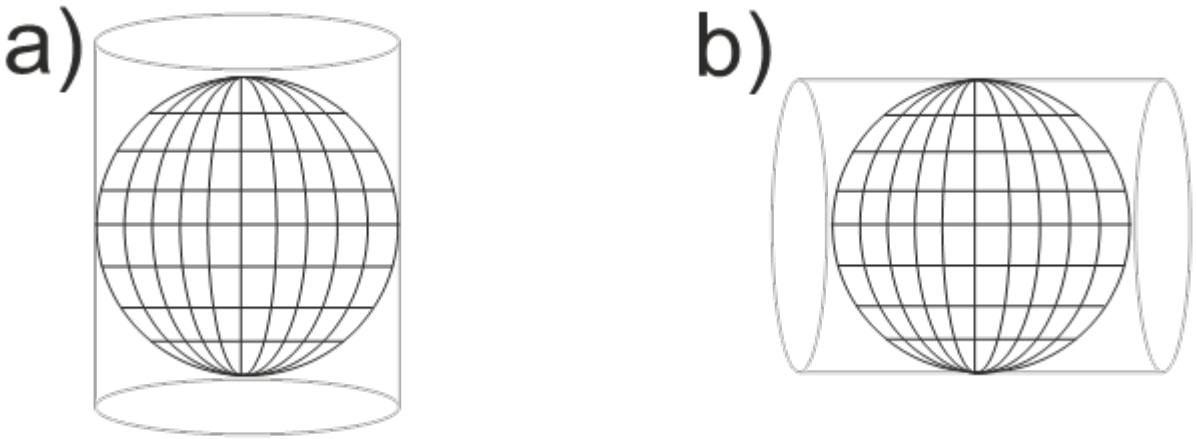

Figure 37. Cylinders used in the projection of a sphere on a flat image in a) equirectangular projection and b) transverse equirectangular projection

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 38 34a). In a second step, all columns of such image are expanded (Figure 38b), to be of the same length (Figure 38c).

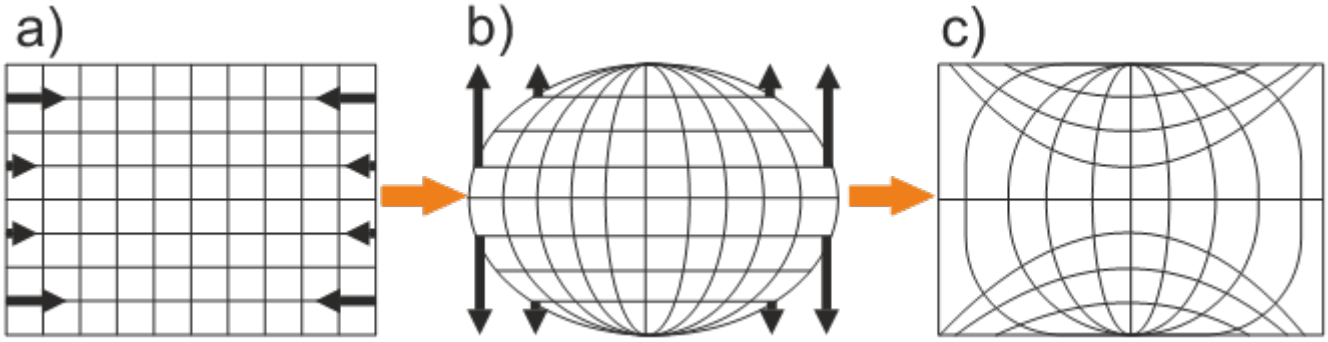

Figure 38. Fast reprojection of an equirectangular image (a) to transverse equirectangular image (c). Black arrows show direction of change of size of respective rows and columns of images

5.5.2. 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 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 39, two modes of viewing space are illustrated, as well as viewing direction with the arrows.

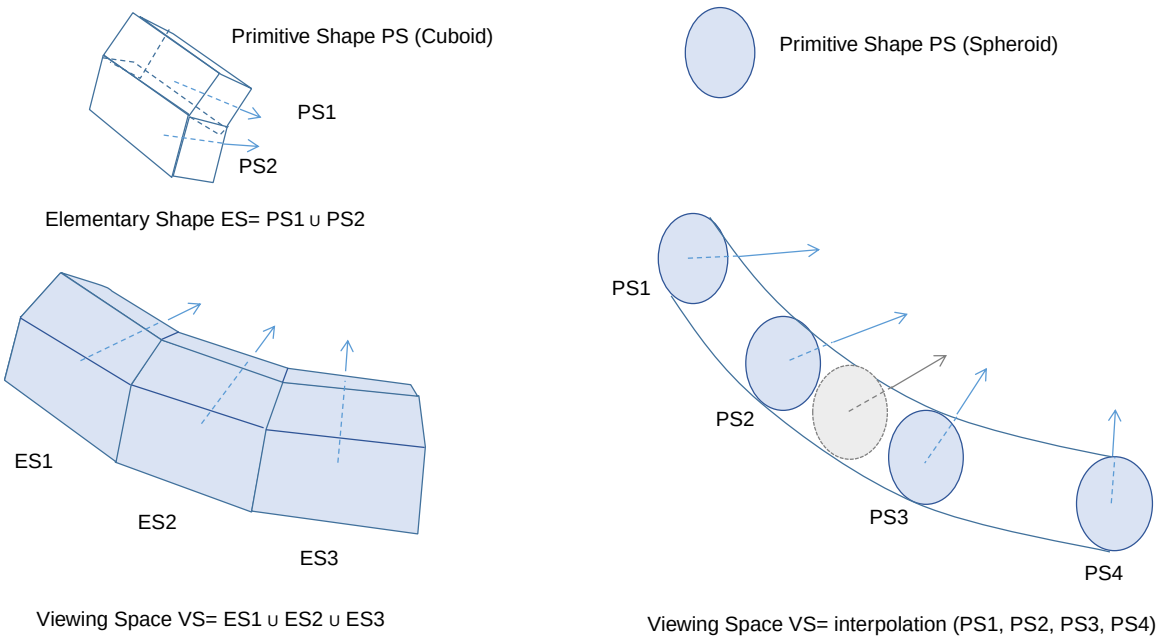

Figure 39. Illustration of VS creation with additive CSG (left) and interpolation (right)

5.6. 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 40 shows the MPI version of Figure 32 for the TMIV renderer. As regards to the block diagram in Figure 32, 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.

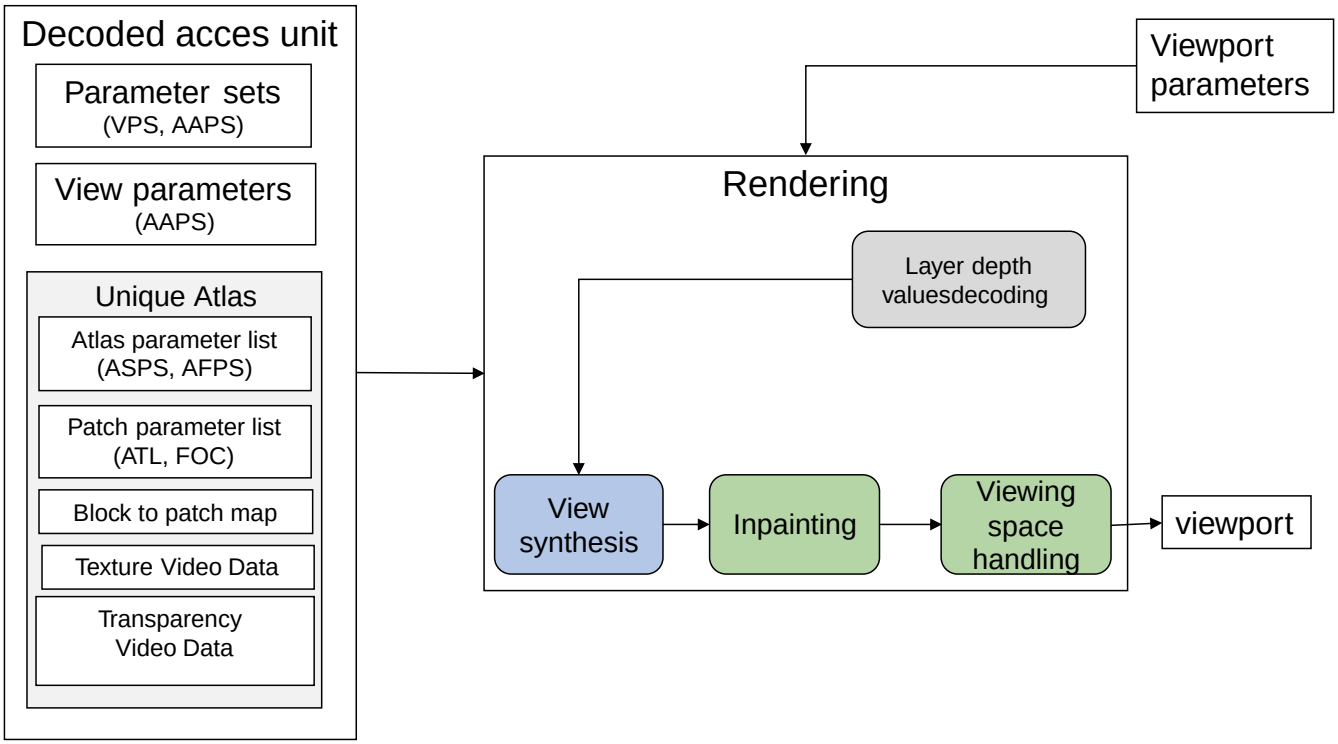

Figure 40. Process flow for TMIV MPI renderer

References

- [1] *Call for Proposals on 3DoF+ Visual*, ISO/IEC JTC1/SC29/WG11 N18145, Jan. 2019, Marrakesh, Morocco.
- [2] J. Boyce, B. Chupeau, L. Kondrad, *Text of ISO/IEC DIS 23090-12 MPEG Immersive Video*, ISO/IEC JTC1/SC29/WG04 N0049, Jan. 2021, Online.
- [3] J. Jung, B. Kroon, *Common Test Conditions for MPEG Immersive Video*, ISO/IEC JTC1/SC29/WG04 N0051, Jan. 2021, Online.
- [4] *Manual of Immersive Video Depth Estimation 3*, ISO/IEC JTC1/SC29/WG04 N0058, Jan. 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] *Soft 3D Reconstruction for View Synthesis*, Eric Penner and Li Zhang, 2017, ACM Transactions on Graphics (Proc. SIGGRAPH Asia) (vol. 6).
- [11] Wikipedia: Painter's algorithm. Variants: Reverse painter's algorithm.
- [12] V-PCC Codec Description, ISO/IEC JTC1/SC29/WG7 N00012, October 2020

Appendix A: Reference software

A.1. Availability and use

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

A.2. Software coordination

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

- Bart Kroon, bart.kroon@philips.com
- Franck Thudor, franck.thudor@interdigital.com
- Christoph Bachhuber, christoph.bachhuber@nokia.com

Appendix 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 as shown in Figure 41. 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 $\mathbf{x} = (x, y, z)^T = x\hat{x} + y\hat{y} + z\hat{z}$. For an untransformed camera the origin is the cardinal point.

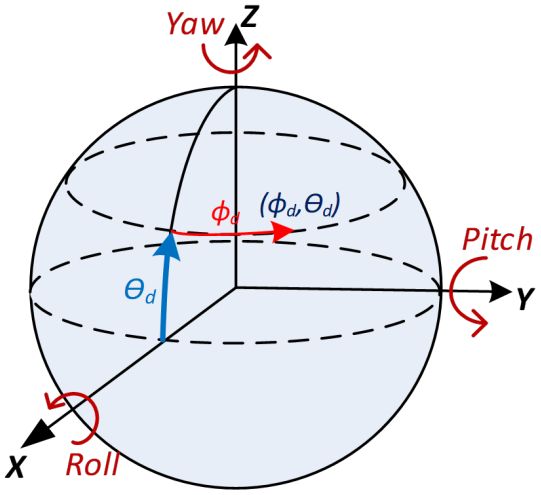

Figure 41. OMAF coordinate system illustrating the directions of positional and rotational units

The definition of image coordinates is:

- The top-left image corner is $(0, 0)$,
- The top-left pixel center is at $(\frac{1}{2}, \frac{1}{2})$,
- $\hat{x}_{\text{image}}$ points right,
- $\hat{y}_{\text{image}}$ points down.

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

B.2. Perspective projection

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

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

Projection:

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

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

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

Unprojection:

The matching projection equation is

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

where d is geometry in meters and x_{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 is a length in meters.

B.3. Equirectangular projection

For equirectangular projection the image is mapped on a horizontal angular range ϕ_1, ϕ_2 and vertical angular θ_1, θ_2 angle as specified in the JSON content metadata file.

Unprojection:

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

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

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

The ray direction is:

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

and the world position is

$$\mathbf{x}_{\text{world}} = r\hat{r},$$

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

$$r = |\mathbf{x}_{\text{world}}|,$$

$$\hat{r} = \frac{\mathbf{x}_{\text{world}}}{r},$$

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

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

$$\phi = \text{atan2}\left(\left\langle \hat{r}, \hat{y} \right\rangle, \left\langle \hat{r}, \hat{x} \right\rangle\right)$$

$$\theta = \sin^{-1}\left\langle \hat{r}, \hat{z} \right\rangle$$

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

$$x_{\text{image}} = w\frac{\phi - \phi_2}{\phi_1 - \phi_2}$$

$$y_{\text{image}} = w\frac{\theta - \theta_2}{\theta_1 - \theta_2}$$

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 ($\mathbf{t}$) and unit quaternions (q) to represent camera extrinsics.

The sequence configuration files and pose traces use Euler angles which are converted directly upon loading. *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 ($\mathbf{x}$) from an input camera to a virtual (output) camera are combined into a single affine transformation (f):

$$f : \mathbf{x} \to q\mathbf{x}q^* + \mathbf{t}$$

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

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

[2] Reordering of the data processing flow is a subject of study in MIV CE-2.8 [WG11N19486]

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

[4] To transform an input MVD scene representation into an MPI representation, the 3D scene is first created by unprojecting each view from the MVD to the 3D scene (note that this is out of scope of the test model and TMIV implementation).