

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

**Coding of moving pictures and audio**

**Convenorship: UNI (Italy)**

**ISO/IEC JTC 1/SC 29/WG 11 N18577**

**Document type: Approved WG 11 document**

**Title:** **Test Model 2 for Immersive Video**

**Status: Output document**

**Date of document: 2019-08-02**

**Source: Video**

**Expected action:**

**No. of pages: 29**

**Email of convenor: leonardo@chiariglione.org**

**Committee URL: mpeg.chiariglione.org**

**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

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

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC 1/SC 29/WG 11 N18577**

**Gothenburg, SE – July 2019**

|  |  |
| --- | --- |
| **Source:** | MPEG-I Video |
| **Title:** | Test Model 2 for Immersive Video |
| **Editors:** | Basel Salahieh, Bart Kroon, Joel Jung, Marek Domański |

**Test Model 2 for Immersive Video**

**Abstract**

MPEG-I Video established the second working draft and the second test model algorithm description during the 127th MPEG meeting (8 to 12 July 2019, Sweden) after evaluating the core experiment results and revisiting the responses to the call for proposal on 3DoF+ [1]. This document serves as a source of general tutorial information on the Metadata for Immersive Video (MIV) design. It defines the terminology used, the process and data flow, the operating modes, the algorithmic components, and the conformance points (i.e. data formats) adopted by the video group for the reference Test Model for Immersive Video (TMIV) at both the encoder & the decoder sides along with the general characteristic of the metadata and bitstream.

# Introduction

The MPEG-I project (ISO/IEC 23090) on coded representation of immersive media supports 3 Degrees of Freedom (3DoF) as described in Figure 1 (left), where a user’s position is static but its head can yaw, pitch and roll. This is available under MPEG-I Part 2 Omnidirectional MediA Format (OMAF) version 1 published in 2018. However, rendering flat 360° video, i.e. supporting head rotations only, may generate visual discomfort especially when objects close to the viewer are rendered. 3DoF+ (which is a 6DoF in a restricted volume) enables head movements for a seated person (adding horizontal, vertical and depth translations) as described in Figure 1 (middle). 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 was issued, to enable such movement of the head within a limited space.

This document describes the second version of the Test Model for Immersive Video (TMIV) that was defined at the 127th meeting of MPEG in July 2019. It is aligned with the TMIV-SW 2.0 reference software. It includes a description of the process flow, the operating modes, the algorithmic components, and the data formats.

Integrated items (TM2.0) (changes compared to N18470):

* No tool adoption.
* Inclusion of operating modes (MIV mode, MIV View mode).
* Editorial improvements.

# Scope

The normative decoding process for Metadata for Immersive Video (MIV) is specified in the working draft specification MPEG/N18576 [2]. The TMIV reference software (TMIV-SW) is provided to demonstrate a reference implementation of non-normative encoding and rendering techniques and the normative decoding process for MIV standard. The software is available on the MPEG Git server as detailed in Section 4.

|  |  |  |
| --- | --- | --- |
|  |  | cid:image003.jpg@01D2725F.2A10A840 |

Figure 1: 3DoF motion yaw, pitch, roll (left), 3DoF+ in a limited space (middle),   
and unconstrained 6DoF (right).

This document provides an algorithm description for the encoder and decoder sides of TMIV reference software. The purpose of this document is to share a common understanding of the coding features and the reference methods supported in the TMIV-SW, in order to facilitate the assessment of the technical impact of new technologies during the standardization process. Common test conditions are provided in MPEG/N18563 [3].

# General Description of the System and Algorithms

## Terminology definitions

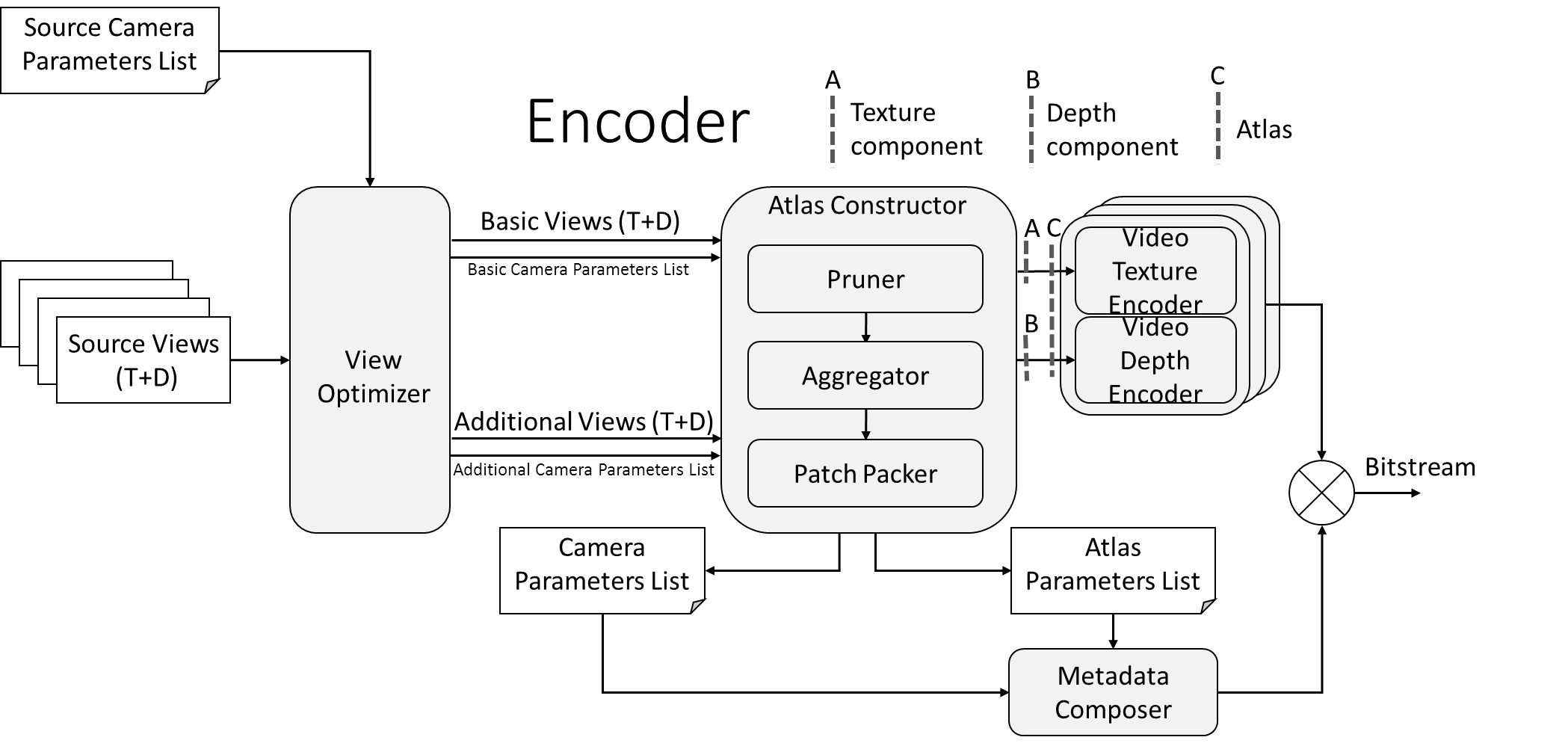
For the purpose of this document, the following terms and definitions apply. The \* superscript next to the term indicates that same definition is shared with the MIV working draft document [2].

Table 1: Terminology definitions used for TMIV**.**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| *3D scene\** | Visual content in the *global reference coordinate system.* |
| *Access unit\** | A set of *NAL units* that are associated with each other according to a specified classification rule, are consecutive in *decoding order* and contain at most one *coded picture* with any specific value of nuh\_layer\_id. [This definition is an exact copy from HEVC Annex F.] |
| *Aggregator* | An embodiment of a process that accumulates pruning masks over an intra-period to account for motion within the scene. |
| *Additional view* | A *view representation* produced by the *view optimizer* operating on *source views* and is pruned and packed in multiple patches*.* |
| *Atlas\** | Aggregation of *patches* from one or more *view representations* after a packing process, into a picture pair which contains a *texture component* *picture* and a corresponding *depth component* *picture*. |
| *Atlas component\** | A texture or depth component of an *atlas*. |
| *Atlas list\** | A list of one or more *atlases* which may be present within the same *access unit*. |
| *Atlas parameters list* | Define how *patches* are packed within the *atlas(es)* and mapped to specific *view representations* in addition to the patches’ size and rotation within the *atlas(es)*. |
| *Atlas patch occupancy map\** | A 2D array corresponding to an *atlas* whose values indicate for each sample position in the *atlas* which *patch* the sample corresponds to, or if the sample is invalid. |
| *Basic view* | A *view representation* produced by the *view optimizer* operating on *source views* and is packed as a whole in a single patch*.* |
| *Camera parameters\** | Defines the projection used to generate a *view representation* from a 3D scene, including intrinsic and extrinsic parameters. |
| *Camera parameters list\** | A list of one or more *camera parameters*. |
| *Inpainter* | An embodiment of a process to fill missing regions prior to outputting a requested *target view*. |
| *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\** | A rectangular region within an *atlas* that corresponds to a rectangular region within a *view representation*. |
| *Patch descriptor\** | A description of the *patch*, containing its size, location within an *atlas*, rotation within an *atlas*, and location within a *view representation*. |
| *Patch packer* | An embodiment of a process to gather *patches* in an *atlas*. |
| *Projection\** | Inverse of the process by which the sample values of a projected *texture component* picture of a *view representation* are mapped to a set of positions in a *3D scene* represented in the *global reference coordinate system* according to the corresponding *depth* sample value and *camera parameter*s *list*. |
| *Pruner* | An embodiment of a process to identify and extract the occluded regions across *basic views* and *additional views* resulting in *patches*. |
| *Renderer\** | An embodiment of a process to create a *viewport* or *omnidirectional view* from a *3D scene* representation corresponding to a viewing position and orientation. |
| *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 *camera parameters*. |
| *Target view* | Indicates either *perspective viewport* or *omnidirectional view* at the desired viewing position and orientation. |
| *View optimizer* | An embodiment of a process in charge of selecting *basic* and *additional* *views.* |
| *View representation\** | 2D sample arrays of a *texture component* and a corresponding *depth component* representing the projection of a *3D scene* onto a surface using *camera parameters*. |
| *Viewport\** | Projection of texture onto a planar surface of a field of view of an omnidirectional or 3D image or video suitable for display and viewing by the user with a particular viewing position and orientation. |

## Process and data flow

A high-level hierarchy of the TMIV encoder and decoder stages is presented in Figure 2. In essence, the TMIV encoder consists of three main steps: the view optimizer, the atlas constructor, and the video encoder & metadata composer. Similarly, the decoder consists of the video decoder & metadata parser, the atlas patch occupancy map generator, and the renderer.



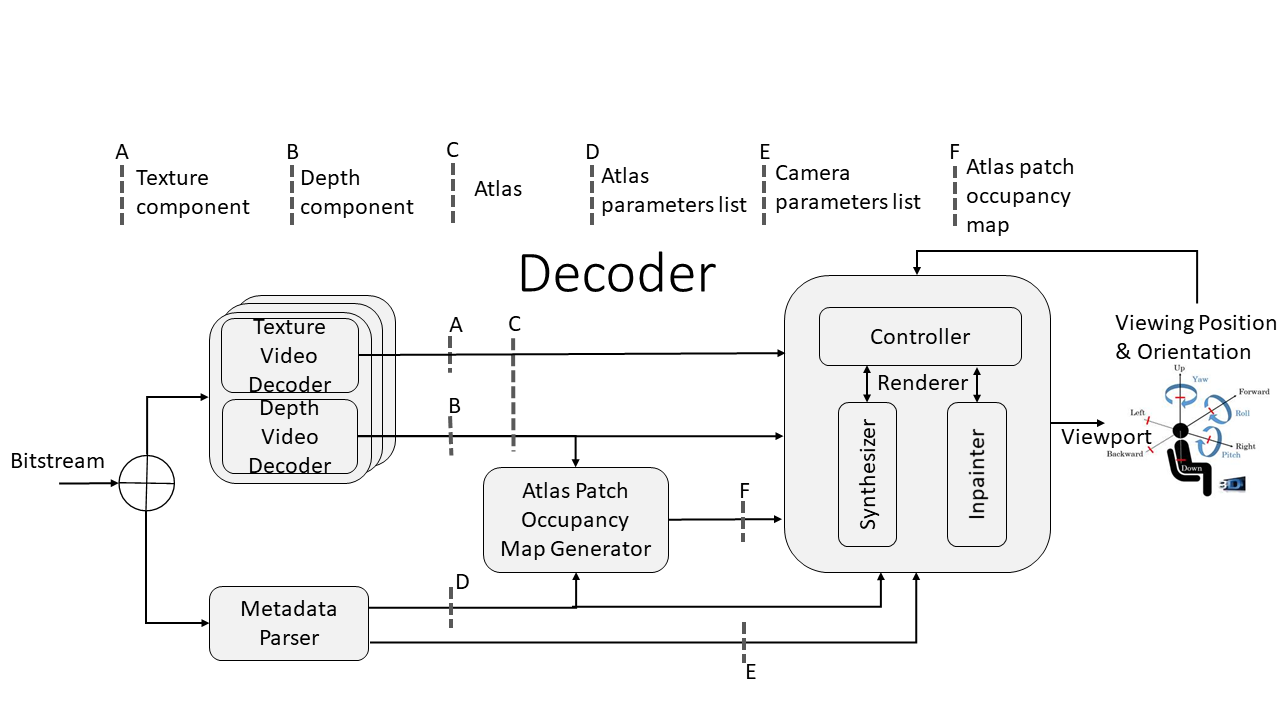
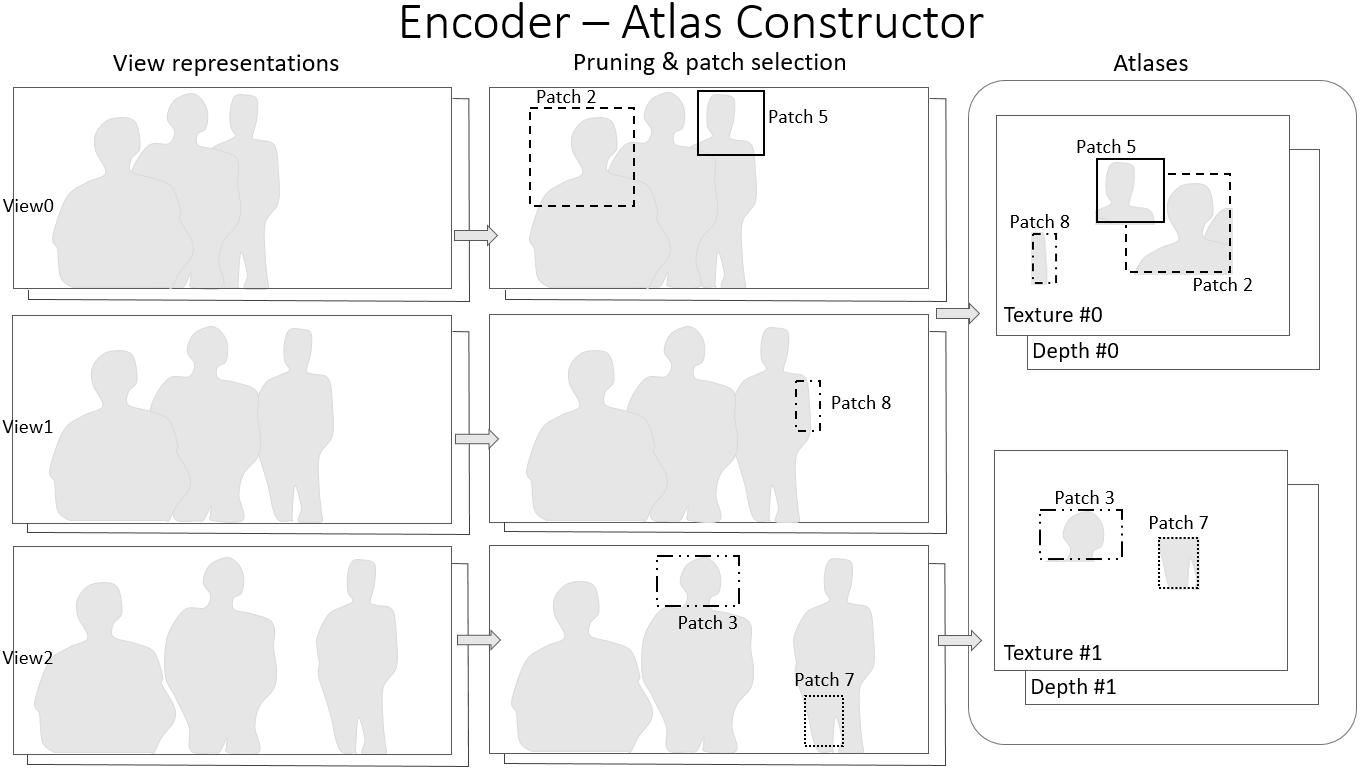


Figure 2: Process flow for the TMIV encoder and decoder.

Figure 3 further illustrates how data is processed across key components; the atlas constructor at the encoder side and the atlas patch occupancy map generator[[1]](#footnote-1) and the renderer at the decoder side. The figure demonstrates an example of how to map between 3 view representations and 2 atlases. More details are given in the related Sections 3.4 and 3.6.



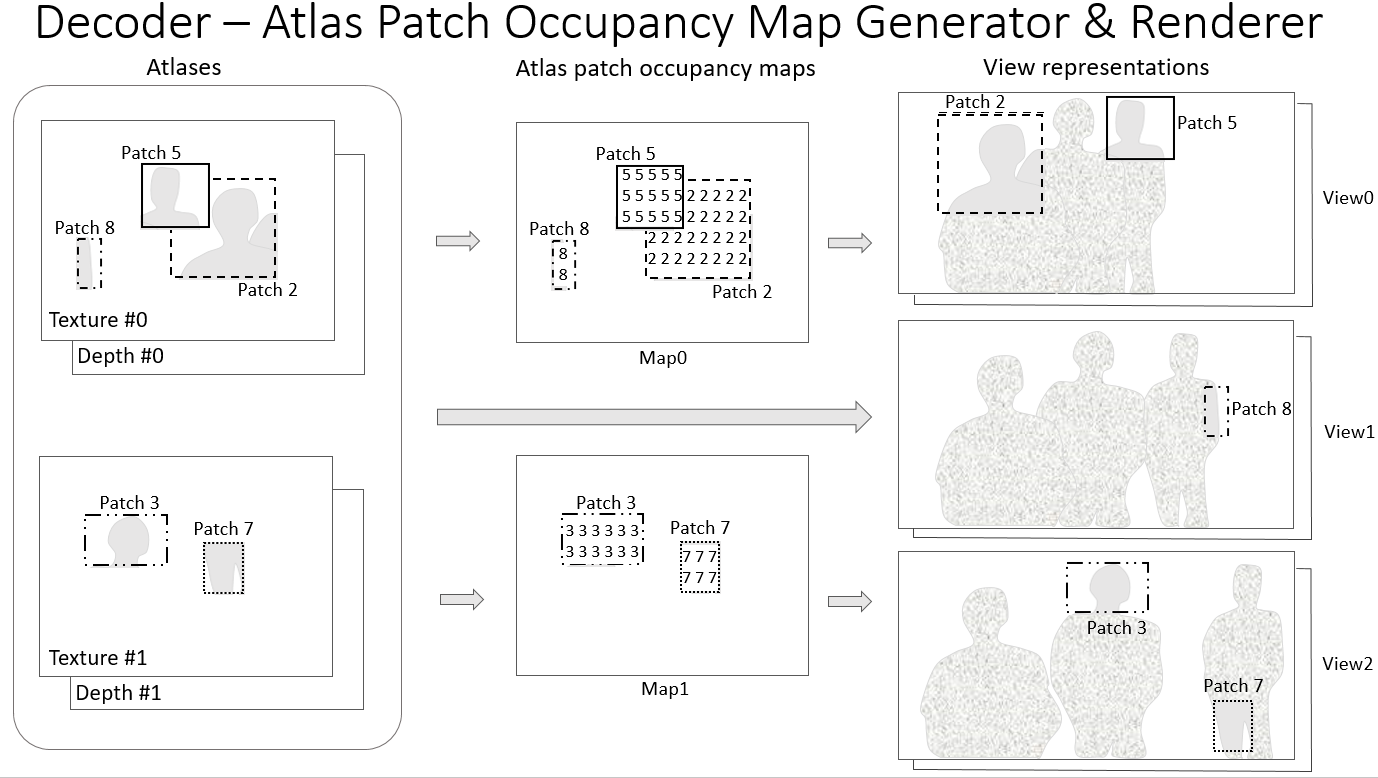


Figure 3: Data flow through key components within the TMIV encoder (top)   
and the TMIV decoder (bottom).

## Inputs, outputs, operating modes, configuration file, and metadata format

### Inputs

Source views (texture & depth) representing projections of 3D real or virtual scene are made available as inputs to the TMIV encoder. The source views can be an equirectangular projection (ERP), or perspective projections. They are provided in luma & chroma ~~4~~:2:0 format with 10 bit for texture and 8, 10, or 16 bit for depth. All the source views have the same resolution (width & height). The source camera parameters list are provided in JSON[[2]](#footnote-2) format and includes the extrinsic parameters (x, y, z positions and yaw, pitch, roll orientations in the format defined by OMAF), the intrinsic parameters (focal lengths, principal points, and distortion coefficients), in addition to the projection type per source view.

### Outputs

The output of TMIV decoder is a perspective viewport or omnidirectional view according to a desired viewing position and orientation, enabling motion parallax cues within a limited space. The rendered output is provided in luma & chroma ~~4~~:2:0 format with 10bit support for texture and depth. It can be displayed on either head mounted display (HMD) or on regular 2D monitor with tracking system feeding the updated viewer’s position and orientation back to the renderer for the next target view.

### Operating modes

The TMIV software can operate in two modes: MIV mode and MIV View mode.

#### MIV mode

In the MIV mode, the source views go through the entire TMIV pipeline where they get optimized, pruned, packed in atlases while keep tracking patches mapping information via occupancy maps before being passed to the renderer.

#### MIV View mode

In MIV View mode, selected complete views are passed from the TMIV encoder’s input to the renderer. The view optimizer here operates in “NoViewOptimizer” mode labeling all selected views as basic views hence no pruning or aggregating is performed and the atlas packer places them in separate atlases ready for video encoding.

At the decoding stage, atlas patch occupancy map generator runs a trivial operation outputting maps of a single patchId per atlas to the renderer at the decoder side. The renderer uses those information along with the metadata and decoded atlases to output the desired viewport or omnidirectional view. This mode helps evaluating the renderer itself in isolation of the other TMIV components.

### Configuration file

The TMIV-SW executable files take a configuration JSON file as an input. To better illustrate the current configuration parameters[[3]](#footnote-3), examples of JSON file for the MIV anchor and the MIV View anchor are given and commented in Annex 1 and Annex 2 respectively.

### Metadata format

The test model uses simple structures for metadata with a fixed length per camera and patch. The structures carry the same information as the working draft. For this first version of the test model, metadata file formats are simple binary serializations.

*// Data type that corresponds to an entry of camera\_params\_list of MPEG/N18*576

*struct CameraParameters {*

*Vec2i size;*

*Vec3f position;*

*Vec3f rotation;*

*ProjectionType type;*

*Vec2f erpPhiRange;*

*Vec2f erpThetaRange;*

*CubicMapType cubicMapType;*

*Vec2f perspectiveFocal;*

*Vec2f perspectiveCenter;*

*Vec2f depthRange;*

*};*

*// Data type that corresponds to camera\_params\_list*

*using CameraParametersList = std::vector<CameraParameters>;*

*// Data type that corresponds to an entry of atlas\_params*

*struct AtlasParameters {*

*uint8\_t atlasId;*

*uint8\_t viewId;*

*Vec2i patchSize;*

*Vec2i posInView;*

*Vec2i posInAtlas;*

*PatchRotation rotation;*

*PatchFlip flip;*

*};*

*// Enumeration of PatchRotation values*

*enum class PatchRotation {*

*upright, // what was up stays up*

*ccw, // what was up goes left, i.e. 90deg*

*ht, // half-turn, i.e. 180deg*

*cw // what was up goes right, i.e. 270deg*

*};*

*// Enumeration of PatchFlip values*

*enum class PatchFlip {*

*none, // what was up stays up*

*vflip // what was up goes down, i.e. vertical flip*

*};*

*// Data type that matches with atlas\_params\_list*

*using AtlasParametersList = std::vector<AtlasParameters>;*

## Encoder

### View optimizer

The view optimizer selects one or several views from the source views and labels them as basic views while the other non-selected source views are labeled as additional views. It includes two steps:

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

The input of the view optimizer is the source views and the source camera parameters list (the position [x, y, z], the orientation (yaw, pitch, roll) [, , ], and the projection type). The output of the view optimizer is the list of basic view(s) and their parameters, and the list of additional views and their parameters.

#### Step1 - Determination of the number of basic views

First, the goal is to find a pair of views (view m, view n) that has the largest direction deviation according to the equation , where *i* and *j* are the indices the source views 0, with , and:

as shown in Figure 4.



Figure 4: Explanation of the directions deviation.

When two pairs provide the same maximum direction deviation, the view pair which has the largest sum of field of views (FOVs) is selected:

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

Second, the overlap between the two views is computed, as illustrated in Figure 5.

Figure 5: Illustration of the overlap and its calculation.

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

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

Finally, the number of basic views is determined:

* If , only one basic view is selected.
* If overlap , multiple basic views including view m and view n are selected.

#### Step 2: Selection of the basic views

#### When only one basic view is needed, according to step 1, the following applies:

The source view that has the largest FOV is selected as a basic view . If several views have the same largest FOV then the following applies:

* Calculate the central camera position of the source capturing system given the source camera parameters list.
* Select as a basic view the source view which camera position is the closest to the central camera position:

#### When several basic views are needed, according to step 1, the following applies:

The views m and n found in step1 are selected as basic views. The view k which has the largest direction deviation with view m and view n is determined:

If view k has less than 50% FOV overlap with the already selected basic views m and n, then view k is selected as a basic view, and the same process is repeated to find the next basic view. Otherwise the process stops.

All other non-selected source views are labeled as additional views and passed along with the basic views to the output of the view optimizer.

### Atlas constructor

The atlas constructor takes as input basic and additional views along with the associated basic and additional camera parameters list and output atlases, with the camera parameters list and with the atlas parameters list, as shown in Figure 2. Each basic view is carried in the atlas as a single, fully occupied patch (assuming the atlas size is equal or larger than the basic view size otherwise the basic view may be split into multiple atlases). The additional views are pruned into multiple patches which may be carried along with a basic view’s patch in the same atlas if the atlas is of larger size or in separate atlas(es).

The atlas constructor is composed of three parts: the pruner, the aggregator, and the patch packer. The pruner and the input of the aggregator operate at the frame level. The output part of the aggregator and the patch packer operate at intra-period level.

Note that the Pruner, the Aggregator and the Patch packer blocks only process depth component, as illustrated in the detailed block diagram drawing of the Atlas Constructor (Figure 6).

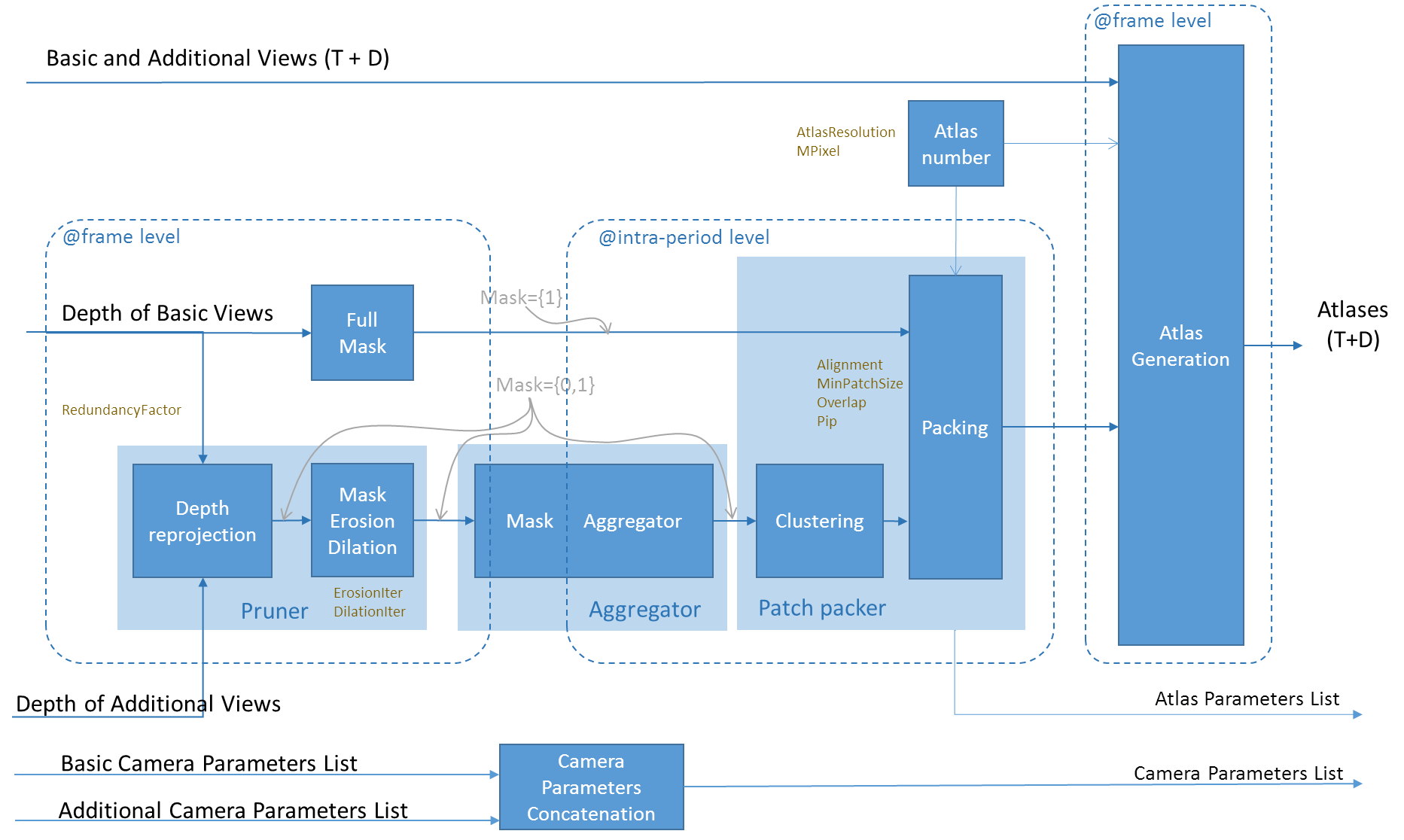


Figure 6: Block diagram of the Atlas Constructor.

#### Pruner

##### Creation of the masks

The created mask indicates the part of the input view to be kept further in the pipeline. The removed samples correspond to parts of the scene identified as redundant. The mask is a binary frame of the same resolution as the view it relates to.

For a basic view, the mask is filled with the value ‘1’, because a basic view is not pruned at all. For an additional view, the mask is filled with the value ‘0’ or ‘1’. It is obtained by re-projecting each depth pixel value of the additional view onto each basic view and validate or invalidate the pixel accordingly. This is done in a ladder type process as illustrated in Figure 7.

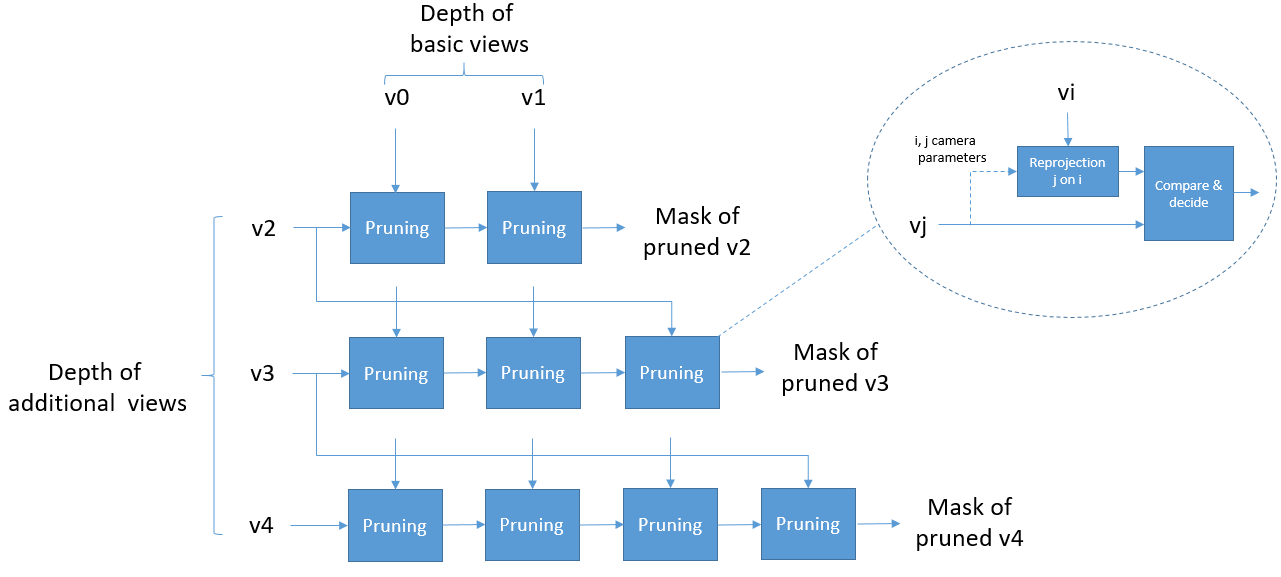


Figure 7: example of pruning with 2 basic views and 3 additional views.

Each elementary “Pruning” block makes use of a reprojection of the view j to be pruned into the reference view i, which can be a basic view or an already pruned view as well. A reprojection consists, first, in de-projecting a pixel from view j to a point in 3D space and, then, in projecting this 3D point onto the reference view i.

In this reprojection, a sample of the additional view is pruned if it is already “covered” by a sample of a previous view (basic or additional) at a small difference which linearly depends of the depth range. More precisely, a sample is said “covered” by the previous depth if its depth z is such that:

with *"RedundancyFactor": 0.05.*

The re-projected pixel generally does not align with the reference view pixel grid. No interpolation is performed for the depth comparison process. Instead, the four nearest pixels of the re-projected pixel are tested, and the mask is set to 0 if at least one among the four satisfied the test.

##### Mask cleaning

A mask has holes and irregularities which are cleaned up by a classical iterative erosion and dilation method on a 3x3 structuring element:

* For the erosion, a pixel that has at least one empty neighbor is discarded (pixel = 0).
* For the dilation, a pixel that has at least one non empty neighbor is filled (pixel =1).
* The related parameters are the number of iterations for the erosion and for the dilation: "ErosionIter": 1, "DilationIter": 5.

#### Aggregator

The mask is reset at the beginning of each intra period. Then, an accumulation is done for each mask’s pixel i with the 1 value across the different frames of the intra period by implementing the logical operation OR as follows:

aggregatedMask[i]@current\_frame =

max(Mask[i]@current\_frame, aggregatedMask[i]@previous\_frame)

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

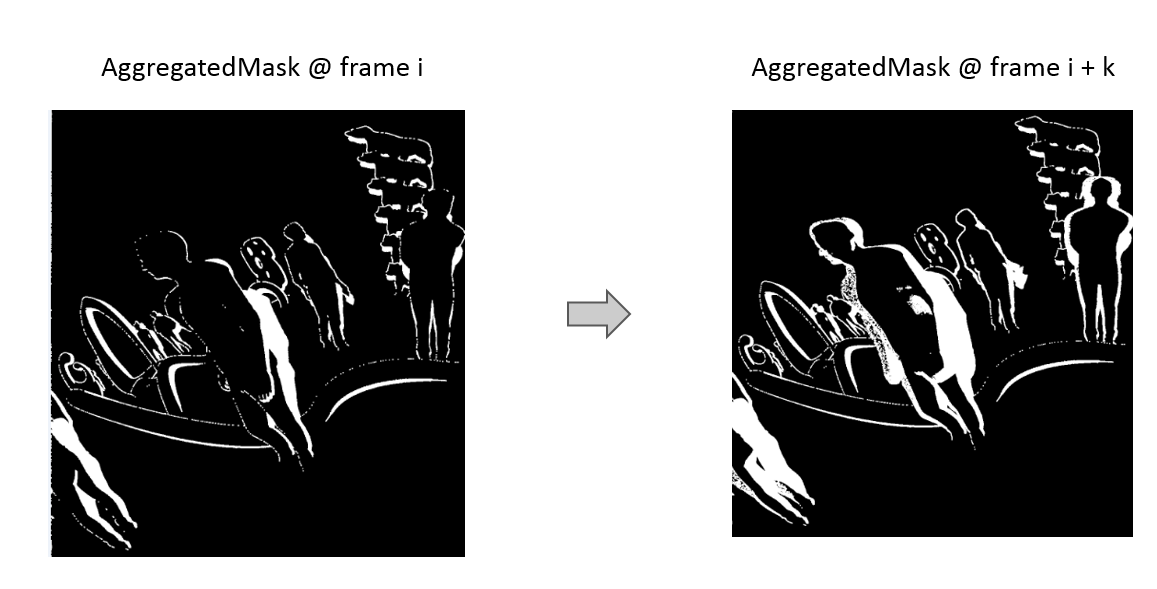


Figure 8. AggregatedMask evolution within an intra period.

#### Patch packer

##### Clustering

This block is in charge of identifying what is called “clusters”. A cluster is a rectangle, containing a set of connected mask pixels of 1s value obtained by a region growing process. The connection criteria of one pixel is the presence of at least one other pixel among the 8 neighbors.

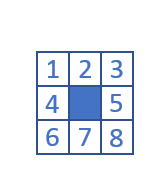


Figure 9: 8-pixel neighborhood for defining the connectivity criteria for region growing.

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

* x and y positions of the top left rectangle corner.
* Width and height of the rectangle.
* The cluster are then sorted by a decreasing size order.

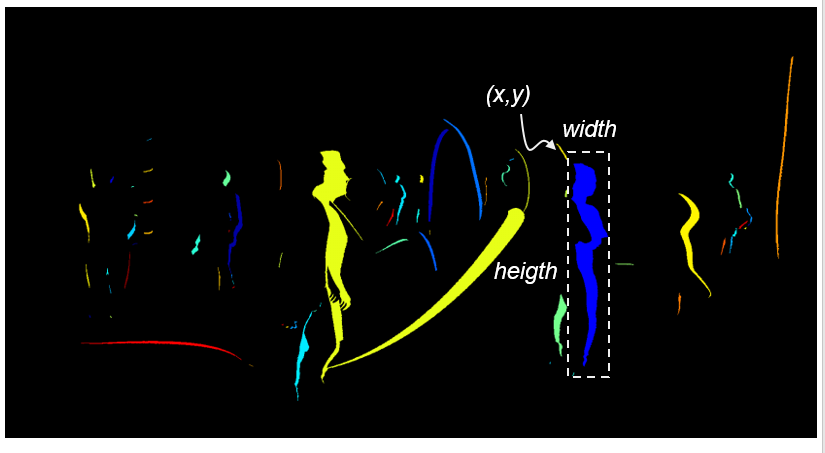


Figure 10: Clusters represented in false color on a pruned view

##### **Selection of the number of atlases**

This function defines the number of atlases by 2 input parameters passed in the configuration file:

* "*AtlasResolution*": [4096, 2048] # [Atlas width, Atlas Height]
* "*MaxLumaSamplesPerFrame*": 33554432 # Maximum size of all atlases combined expressed in luma samples per frame (for texture plus depth)

The number of atlases is given by:

##### **Packing**

The packing process tries to sequentially pack each cluster into the atlases. The input parameters are the following:

* *“Alignment”* is defined as a number of pixels. Patch size and patch position are multiple of alignment. Default value is 8.
* *“MinPatchSize”* is the number of pixels of the smallest edge 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 patches; it prevents seam artefacts. Default value is 1
* *“PiP”* is a flag enabling the Patch-in-Patch feature when equal to 1. Default value is 1.

The packing process is based on a version of MaxRect algorithm [6]. It makes use of the existing “Used Space” first, by examining the space which is efficiently occupied (“Filled space”). It is made of intricated loops which are described by the following pseudo-code:

|  |
| --- |
| *For each cluster*  *For each atlas*  *Try to push the cluster in Used Space*  *Try with 0° rotation first*  *Else with 90° rotation*  *Else*  *Try to push the cluster into free space*  *Try with 0° rotation first*  *Else with 90° rotation*  *Else*  *Split the cluster into 2 parts by its largest border*  *For each resulting 2 parts*  *If smaller than MinPatchSize*  *Discard*  *Else*  *Put the part in the cluster priority list* |

Currently (as described in the pseudo-code), only a possibility of 90o rotation (with no flipping) is implemented in TMIV2.0 encoder although the metadata allows more variations and TMIV2.0 decoder can interpret all possibilities.

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

* The location in the atlas patch\_pos\_in\_atlas\_x and patch\_pos\_in\_atlas\_y along with the AtlasId.
* The location in the original view representation patch\_pos\_in\_view\_x and patch\_pos\_in\_view\_y, and its dimensions patch\_width\_in\_view and patch\_height\_in\_view.
* The related ViewId, which itself refers to the de-projection parameters for that view in the decoder
* A possible rotation by i\*90° where i = 0, 1, 2, 3
* A possible vertical flip

The packing operation from view representation to Atlas is done with rotation (first) then vertical flipping (second). The eight achievable states for the patch packing are covered by the “rotation/vertical flip” configurations.

Note that at the encoding side, the rotation of 90° is here meant to be from view representation to Atlas and is counter-clockwise, i.e. rotates the Y-Axis on the X axis, as illustrated in the following figure (Figure 11).

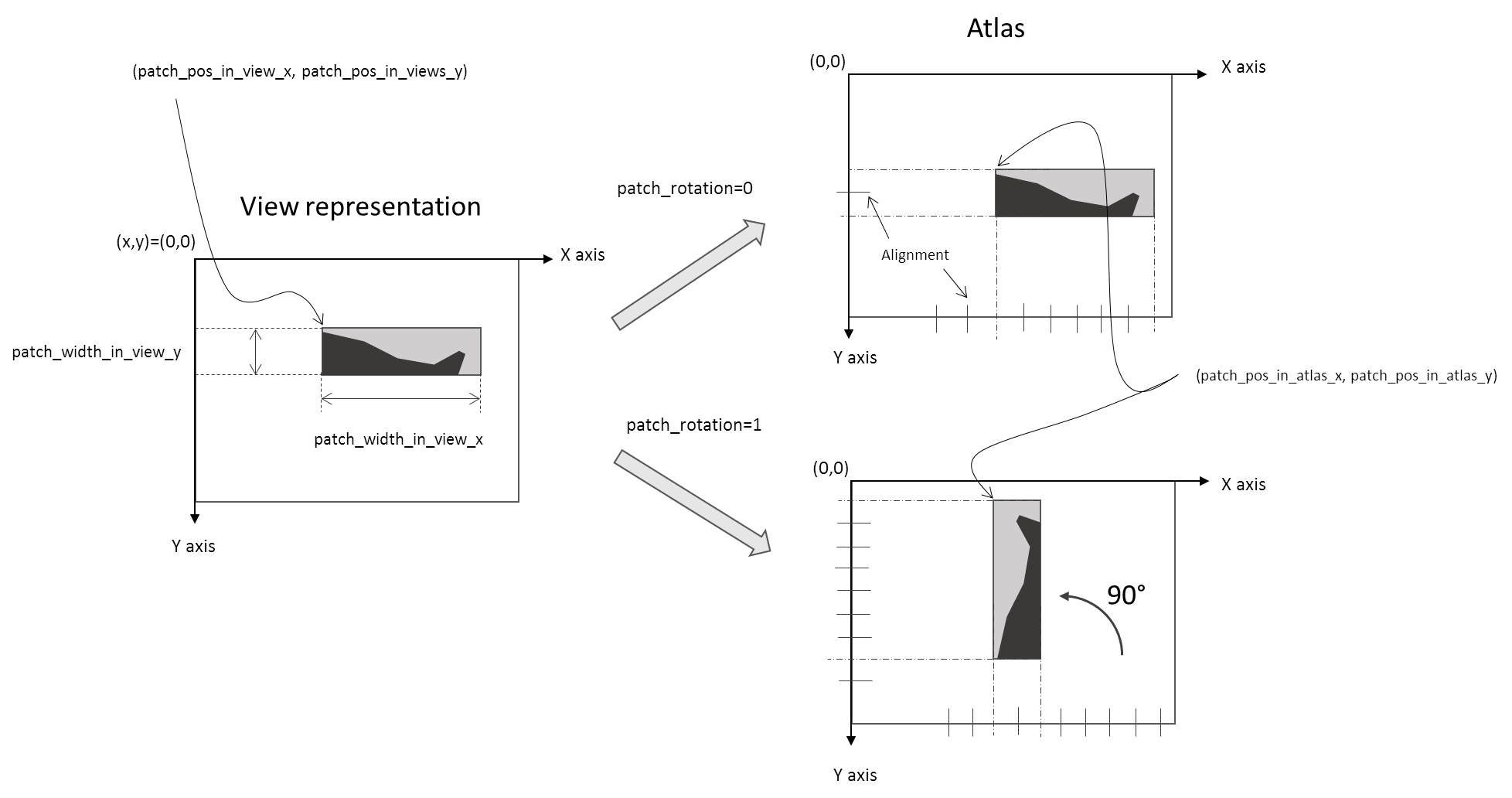


Figure 11: Meaning of patch related parameters.

Although the working draft [2] allows rectangular patches to have unoccupied pixels, in TMIV2.0 the entire rectangular patch is currently set to be occupied. Unoccupied pixels are indicated by setting the corresponding depth pixel luma values to 0.

#### Atlas generation

The final operation within the atlas constructor is writing the patches in the buffer allocated to the atlas (both the depth and the texture components). Figure 12 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.

These are rectangles are fully occupied in general. They may be partially empty when basic and/or additional views contains invalid pixels (it may be the case when these latter views are not source views for instance). In that case, the null value in the depth expresses the invalidity of a sample.

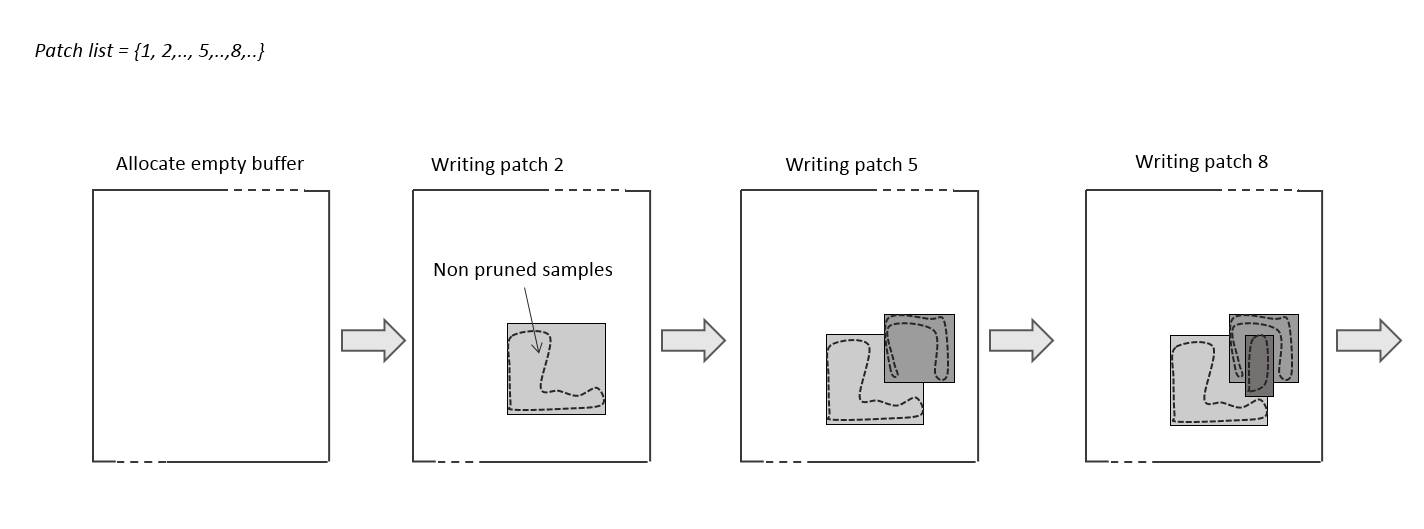


Figure 12: Successive writing of patches into an atlas.

#### Camera parameters concatenation

This block outputs a single camera parameters list by simply concatenating the basic camera parameter list and the additional camera parameters list.

### Video encoder and metadata composer

The HEVC encoder of profile Main10 in the Random Access configuration is used to encode the texture and depth of the atlas(es) video (in separate layers) provided in 4:2:0 10 bit format. The metadata composer merge the camera parameters list and atlas parameters list outputted by the Atlas Constructor, as shown in Figure 2.

## Bitstream

The bitstream contains:

* An HEVC bitstream containing one or more layer pairs (as defined in [2]), each layer pair representing an atlas.
* Metadata composed of:
  + Camera parameters list.
  + Atlas parameters list.

## Decoder

### Video decoder and metadata parser

The video decoder receives HEVC encoded atlases. The texture component and the depth component may be decoded independently by HEVC decoders.

The metadata parser splits the received metadata into camera parameters list and atlas parameters list. The atlas parameters list is used by the Atlas Patch Occupancy Map Generator and the Renderer whereas the camera parameters list is only used by the Renderer to produce the viewport requested (Figure 2).

### Atlas patch occupancy map generator

An occupancy map is generated for each atlas. It has the same size as the atlas. In the TMIV-SW the atlas patch occupancy map generator is only updated at each new intra period, while the signalization of sample invalidity is only used within the renderer.

This map gives for each sample the number of the patch that belongs to, as illustrated in the figure below. The map is simply created by browsing the atlas parameters list from the parsed metadata exactly in the same order as during its creation, to resolve any overlapping. Figure 13 illustrates how three overlapping patches can be resolved by following the right order.

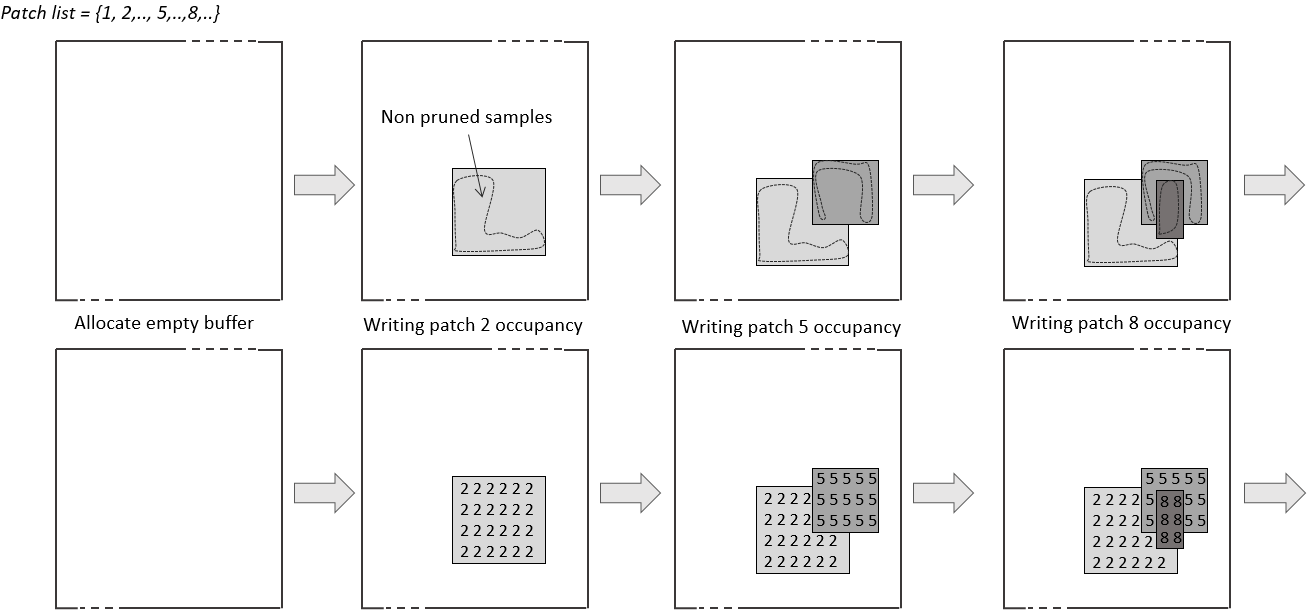


Figure 13: Atlas Occupancy Map Generation in an ordered manner to resolve the overlapping patches.

This map is then used in a loop on the atlas’s samples to get their respective PatchId, which itself enables getting the respective ViewId, hence enabling the de-projection and re-projection on the viewport.

To keep tracking the valid-pixels per patch while accounting for compression artifacts and noise during the streaming, we set to zero all depth values (in the depth component of the atlases) that are below a threshold (set to 64 in TMIV2.0). This results in identifying and excluding the non-valid pixels within patches during the rendering process which helps in suppressing their artifacts.

### Renderer

The TMIV-SW has a flexible rendering engine that is able to render directly from atlases using pipelined reprojection and parallel rasterization of triangles to reduce the wall time for generating viewports. The Renderer is based on RVS with improvements from CfP responses, and it can render directly from multiple atlases and/or multiple views. The output is a view with texture and depth of the same bit depth.

As depicted in Figure 2 the renderer has three parts:

* The Controller (§3.6.3.1) accepts the input data, invokes the Synthesizer and the Inpainter in a multi-pass manner and forwards the output.
* The Synthesizer (§3.6.3.2) reprojects, rasterizes and blends the input data.
* The Inpainter (§3.6.3.3) replaces any missing pixels (indicated by level 0 in the depth map) with interpolated texture and depth data.

#### Controller

The controller invokes the Synthesizer in multiple passes where *“NumberOfPasses”* and *“NumberOfViewsPerPass”* can be tuned as part of the configuration parameters. At first only nearby views (or patches belonging to nearby views) are used for the synthesis to output coherent synthesis results. Then, the view selection is extended to include views further away (or patches belonging to views further away) from the target view to output more complete synthesis results. The process is repeated over the chosen number of passes. When operating on atlases, local occupancy maps are created per pass such that they include only the patchId of patches from the selected views per pass. Then they are passed to the synthesizer to render only these selected patches. Afterward, the synthesis results of individual passes are merged together in a successive manner to output a coherent and complete synthesis results. Finally, the Inpainter is engaged to fill the missing regions prior to outputting the requested target view. A block diagram of the multi-pass operation invoked by the Controller is shown in Figure 14.

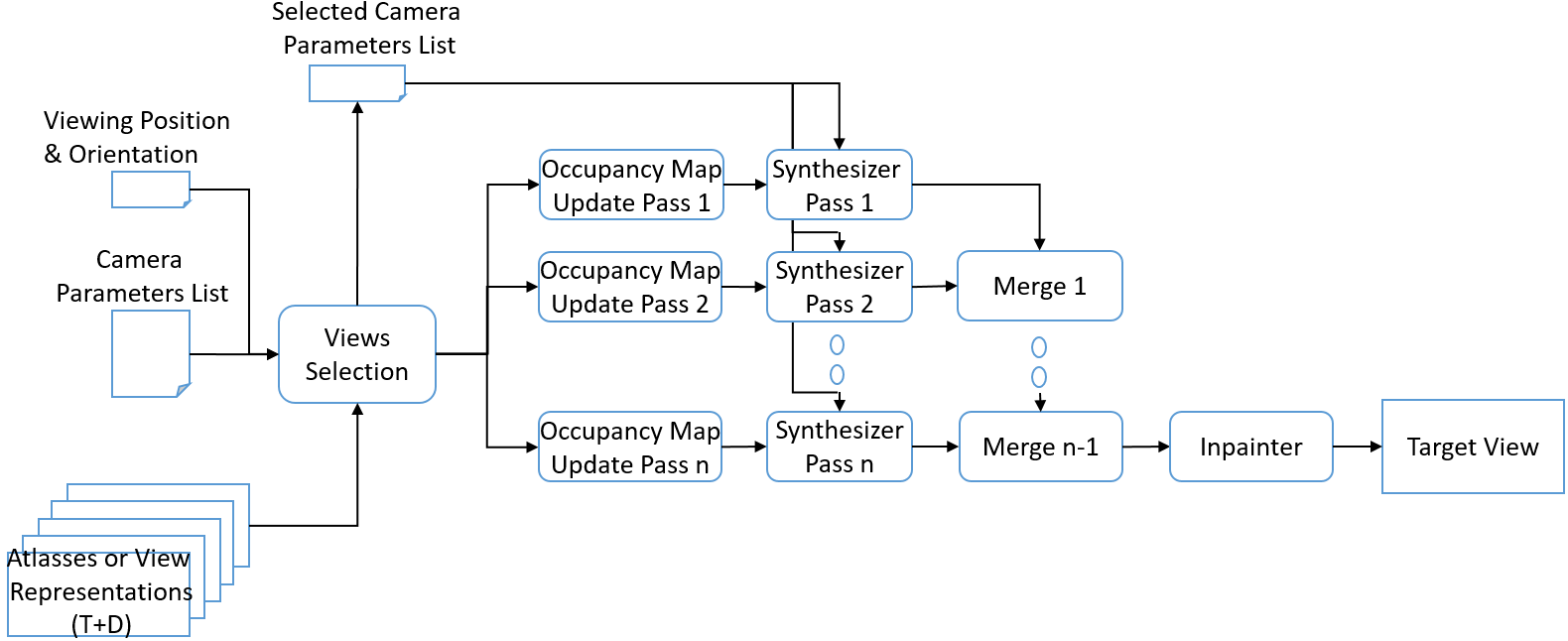


Figure 14: Process flow for the multi-pass Controller.

#### Synthesizer

Like RVS [4], the synthesis is based on:

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

##### Rendering from atlases

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

* Multiple atlases with 10-bit texture and 10-bit depth (normalized disparities),
* An atlas patch occupancy map per atlas with 16-bit values (Figure 13),
* A single atlas parameters list,
* A single 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-bit texture and 10-bit depth components.

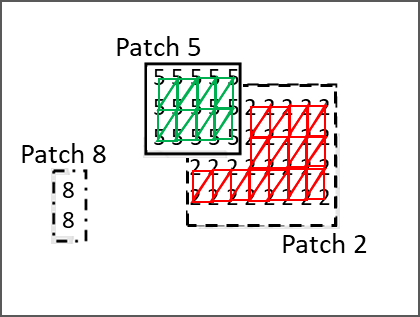


Figure 15: Creating a mesh from an atlas. Triangles between pixels from Patch 5 and 2 are omitted. Note that Patch 8 is not drawn because no triangle can be formed. Unused pixels are skipped too.

The process is to build a mesh (Figure 15) from each of the atlases:

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

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

##### Rendering from multiple views

With core experiments in mind the renderer can also be part of the encoder and render directly from the source format:

* Multiple views with 10-bit texture and 16-bit depth (normalized disparities),
* A single source camera parameters list,
* Target camera parameters for a perspective viewport or an omnidirectional view.

The output of renderer is a single view (viewport or omnidirectional) with same input’s bitdepth.

##### 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).

The weighted sums are normalized by the depth 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 | 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 depth | The reciprocal of the depth value in the target view [diopter]. | Prefer foreground over background (depth ordering). |
| Stretching | The unclipped area of the triangle in the target view relative to the source view. | Penalize triangles that stretch between foreground and background objects. |

#### Inpainter

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 depth of both neighbors, the color of the neighbor with further depth is copied instead of using a 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 the additional step of changing projection type is performed, and the search of the nearest points is performed within transverse ERP images (transverse equirectangular projection – the Cassini projection [5]). In equirectangular projection, a sphere is mapped onto a cylinder that is tangential to points on a sphere having the latitude equal to 0 degrees (Figure 16a). In transverse projection, the cylinder on which the sphere is mapped is rotated by 90 degrees, so it is tangential to points that have longitude equal to 0 degrees (Figure 16b). 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.

|  |  |
| --- | --- |
| cylinders | cylinders |

Figure 16: 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 17a). In a second step, all columns of such image are expanded (Figure 17b), to be of the same length (Figure 17c).

transverse

Figure 17: 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.

# Reference software

The reference software (TMIV-SW) is publicly available on the Gitlab server at:

<https://gitlab.com/mpeg-i-visual/tmiv/>

The software is ISO C++17 conformant and does not require external libraries. Core experiments are expected to include the reference software as a subproject and introduce new components. Alternatively core experiments may fork the test model.

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

* Julien Fleureau, [julien.fleureau@technicolor.com](mailto:julien.fleureau@technicolor.com)
* Bart Kroon, [bart.kroon@philips.com](mailto:bart.kroon@philips.com)
* Bin Wang (王彬), [3130100819@zju.edu.cn](file:///E:\\BS%20Sync\\Projects\\LFStreaming\\Reports\\MPEG\\126%20Geneva%202019\\TestModel\\3130100819@zju.edu.cn)

Contributions should be in the form of git pull requests.

**References**

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

[2] J. Boyce, R. Doré, V. Kumar Malamal Vadakital (Eds.), “Working Draft 2 of Metadata for Immersive Media (Video)”, ISO/IEC JTC1/SC29/WG11 MPEG/w18576, July. 2019, Gothenburg, Sweden.

[3] J. Jung, B. Kroon, J. Boyce, Common Test Conditions for Immersive Video, ISO/IEC JTC1/SC29/WG11 MPEG/N18563, July. 2019, Gothenburg, Sweden.

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

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

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

**Annex 1: MIV configuration**

The example of the *ClassroomVideo* sequence (denoted by SA) of resolution 4096x2048 is considered.

The parameters setting the temporal aspect (i.e. frame parameters) of the TMIV-SW are set as follows:

*"startFrame": 0,*

*"numberOfFrames": 120,*

*"intraPeriod": 32,*

*“extraNumberOfFrames”: 180 (Optional)*

The *extraNumberOfFrames* option mirrors a decoded video a specified length.

The source views and the associated parameters are set as follows:

*"SourceTexturePathFmt": "%s\_texture\_4096x2048\_yuv420p10le.yuv",*

*"SourceDepthPathFmt": "%s\_depth\_4096x2048\_yuv420p16le.yuv",*

*"SourceCameraParameters": "ClassroomVideo.json",*

*"SourceCameraNames": [ "v0", "v1", "v2", "v3", "v4", "v5", "v6", "v7", "v8", "v9", "v10", "v11", "v12", "v13", "v14"]*

The *SourceCameraParameters* are in RVS metadata format [4] and provide camera parameters and source resolutions per source view. There may be more cameras in this JSON, but the ones indicated by *SourceCameraNames* are used in the specified order.

The target view is set as follows:

*"OutputCameraName": ["viewport"],*

*“PoseTracePath”: “Apt2.csv” (Optional)*

Hereby *OutputCameraName* has to be a camera in the *SourceCameraParameters* file. The pose trace is optional and when provided it shall have the RVS pose trace format [4].

When the view optimizer and atlas constructor run as separate executables, then the paths for intermediate data and metadata files have to be specified:

*"BasicTexturePathFmt": “BAS\_SA\_R0\_Tt\_v%02d.yuv",*

*"BasicDepthPathFmt": "BAS\_SA\_R0\_Td\_v%02d.yuv",*

*"BasicMetadataPath": "BAS\_SA\_R0\_Tm\_vxx.bit",*

*"AdditionalTexturePathFmt": "ADD\_SA\_R0\_Tt\_v%02d.yuv",*

*"AdditionalDepthPathFmt": "ADD\_SA\_R0\_Td\_v%02d.yuv",*

*"AdditionalMetadataPath": "ADD\_SA\_R0\_Tm\_vxx.bit"*

The following fields specify the output of the encoder and the input of the decoder:

*"AtlasTexturePathFmt": "ATL\_SA\_R0\_Tt\_c%02d\_4096x2048\_yuv420p10le.yuv",*

*"AtlasDepthPathFmt": "ATL\_SA\_R0\_Td\_c%02d\_4096x2048\_yuv420p10le.yuv","AtlasMetadataPath": "ATL\_SA\_R0\_Tm\_c00.bit"*

When the atlas patch occupancy map generator and the renderer are run as separate executables, then the path format for the occupancy maps has to be specified:

*"AtlasPatchOccupancyMapFmt": "APO\_SA\_R0\_Td\_c%02d.yuv"*

The final paths to be specified are the output paths:

*"OutputTexturePath": "TM1\_SA\_R0\_Tt\_%s\_4096x2048\_yuv420p10le.yuv",*

*"OutputDepthPath": "TM1\_SA\_R0\_Td\_%s\_4096x2048\_yuv420p10le.yuv", (Optional)*

The config file selects one implementation per component, but the reference software may provide some alternatives (no view optimization, single-pass / multi-pass rendering or no inpainting). Core experiments will add more implementations. A component is selected through *Method* parameters and method parameters are specified per component in a section with the name of the method:

*"ViewOptimizerMethod": "ViewReducer",*

*"ViewReducer": {}*

The section has to be present even when there are no method parameters like in above case.

The configuration of components is hierarchical:

*"EncoderMethod": "Encoder",*

*"Encoder": {*

*"ViewOptimizerMethod": "ViewReducer",*

*"ViewReducer": {},*

*"AtlasConstructorMethod": "AtlasConstructor",*

*"AtlasConstructor": {*

*"PrunerMethod": "Pruner",*

*"Pruner": {*

*"RedundancyFactor": 0.05,*

*"ErosionIter": 1,*

*"DilationIter": 5*

*},*

*"AggregatorMethod": "Aggregator",*

*"Aggregator": {},*

*"PackerMethod": "Packer",*

*"Packer": {*

*"Alignment": 8,*

*"MinPatchSize": 16,*

*"Overlap": 1,*

*"PiP": 1*

*},*

*"AtlasResolution": [4096, 2048],*

*"MaxLumaSamplesPerFrame": 33554432*

*}*

*},*

*"DecoderMethod": "Decoder",*

*"Decoder": {*

*"AtlasDeconstructorMethod": "AtlasDeconstructor",*

*"AtlasDeconstructor": {},*

*"RendererMethod": "* *Renderer",*

*"* *Renderer": {*

*"SynthesizerMethod": "Synthesizer",*

*"Synthesizer": {*

*"rayAngleParameter": 20,*

*"depthParameter": 20,*

*"stretchingParameter": 0.6,*

*"maxStretching": 5*

*},*

*"InpainterMethod": "Inpainter",*

*"Inpainter": {},*

*}*

*}*

**Annex 2: MIV View configuration**

The example of the *ClassroomVideo* sequence (denoted by SA) of resolution 4096x2048 is considered.

*{*

*"startFrame": 23,*

*"numberOfFrames": 97,*

*"intraPeriod": 32,*

*"SourceTexturePathFmt": "%s\_texture\_4096x2048\_yuv420p10le.yuv",*

*"SourceDepthPathFmt": "%s\_depth\_4096x2048\_yuv420p16le.yuv",*

*"SourceCameraParameters": "ClassroomVideo.json",*

*"SourceCameraNames": [*

*"v0", "v7", "v8", "v9", "v10", "v11", "v12", "v13", "v14"*

*],*

*"AtlasTexturePathFmt": "ATL\_SA\_R0\_Tt\_c%02d\_4096x2048\_yuv420p10le.yuv",*

*"AtlasDepthPathFmt": "ATL\_SA\_R0\_Td\_c%02d\_4096x2048\_yuv420p10le.yuv",*

*"AtlasMetadataPath": "ATL\_SA\_R0\_Tm\_c00.bit",*

*"OutputTexturePath": "TM1\_SA\_R0\_Tt\_%s\_4096x2048\_yuv420p10le.yuv",*

*"OutputCameraName": "v1",*

*"EncoderMethod": "Encoder",*

*"Encoder": {*

*"ViewOptimizerMethod": "NoViewOptimizer",*

*"NoViewOptimizer": {},*

*"AtlasConstructorMethod": "AtlasConstructor",*

*"AtlasConstructor": {*

*"PrunerMethod": "Pruner",*

*"Pruner": {*

*"RedundancyFactor": 0.02,*

*"ErosionIter": 1,*

*"DilationIter": 5*

*},*

*"AggregatorMethod": "Aggregator",*

*"Aggregator": {},*

*"PackerMethod": "Packer",*

*"Packer": {*

*"Alignment": 8,*

*"MinPatchSize": 16,*

*"Overlap": 1,*

*"PiP": 1*

*},*

*"AtlasResolution": [4096, 2048],*

*"MaxLumaSamplesPerFrame": 150994944*

*}*

*},*

*"DecoderMethod": "Decoder",*

*"Decoder": {*

*"AtlasDeconstructorMethod": "AtlasDeconstructor",*

*"AtlasDeconstructor": {},*

*"RendererMethod": "MultipassRenderer",*

*"MultipassRenderer": {*

*"NumberOfPasses": 3,*

*"NumberOfViewsPerPass": [2, 4, 9],*

*"SynthesizerMethod": "Synthesizer",*

*"Synthesizer": {*

*"rayAngleParameter": 20,*

*"depthParameter": 20,*

*"stretchingParameter": 0.6,*

*"maxStretching": 5*

*},*

*"InpainterMethod": "Inpainter",*

*"Inpainter": {}*

*}*

*}*

*}*

1. Please note that the Atlas Patch Occupancy Map Generator is called TMIV::AtlasDeconstructor in TMIV-SW. The component has the additional (optional) functionality of reconstructing pruned views. [↑](#footnote-ref-1)
2. <http://json.org/> [↑](#footnote-ref-2)
3. A configuration file is part of TMIV-SW as *doc/ExampleConfiguration-SA.json*. [↑](#footnote-ref-3)