INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 4
MPEG VIDEO CODING

ISO/IEC JTC 1/SC 29/WG 4 m66516
January 2024, Virtual

Title: **[MIV] Report on EE 1: rendering and refactoring**

Source: **Adrian Dziembowski (PUT), Bart Kroon (Philips), Jong-Beom Jeong (SKKU)**

Abstract

This document is the report of ISO/IEC 23090-12 *MPEG immersive video (MIV) 2nd edition* exploration experiment (EE) 1 on rendering and refactoring, as described in N 0413. Our recommendations are to integrate proposed changes into the reference software and view the best reference pose trace videos. Besides pure refactorings, the MPI encoder was improved to support frame packing and all codec group IDC's, and for all TMIV executables with JSON configuration file, unused parameters now result in an early error.

1 Introduction

This report provides the results of the exploration experiment per task in the following sections. Objective results are attached, and subjective results (pose trace videos) are available on request.

2 Task 1: VWS vs. AS comparison, no SW changes

The purpose of this experiment is to compare the additive synthesizer (AS) with the view-weighting synthesizer (VWS) with the goal of removing one of them from the reference software. Currently the VWS is used for all anchors and best reference. The AS is used for the server-side inpainter and the rasterizer within it is also used for the hierarchical pruner to perform incremental view synthesis.

To prepare for the experiment a branch `mpeg145-ee1.1` was created which modifies the main anchor, DSDE anchor and best reference configurations to use the AS. The experiment was conducted without noticeable problem. Conditions were N0377 CTC main anchor and best reference with TMIV 17.1 instead of TMIV 17.0. An additional test was performed to verify that these tow versions perform equally.

Objective evaluations are attached to this document and pose trace videos can be made available on request.

It is useful to study the best reference, for instance CBA Basketball, because the subjective difference is large in the opinion of the participants. Occasionally the AS outperforms VWS, at least objectively, but in general AS has more artifacts then VWS. It especially happens often that foreground objects are semi-translucent.

One additional objective metric was calculated: the size of the pose trace video zip-files for AS and VWS are different, indicating that VWS renderings are smoother/less noisy than AS:

- R65 (RP0): 1.3 GB (VWS) vs 1.8 GB (AS)
- A65 (RP0..RP4): 5.9 GB (VWS) vs. 9.1 GB (AS)

The Philips experts have tried to tune AS parameters in the past, and by their judgment it is unlikely that there is a combination of parameters that works good enough for all sequences. No tuning was tried in the context of this EE.

The participants recommend removal of the additive synthesizer (AS) as a viewport renderer based on collected subjective and objective results. Because the AS is also used for pruning and view optimization, the code cannot be entirely removed, however it is possible to narrow down the interface of the AS to support only required functionality:

- There is no need for the AS to render from a decoded access unit.
- The pruner mainly re-uses the rasterizer and reprojection code.
- The server-side inpainter renders full views.

The benefit of this change is that we avoid having a test model with a selectable renderer that does not comply to the intention of the standard, for instance because it does not support patch margin signalling.

If the AHG follows the recommendation, then the participants intend to provide a branch with the described change.

Finally, the participants wonder how much of complexity was added to the initial version of the hierarchical pruner to compensate for rendering errors? Would the pruning result improve when the rendering test itself is improved?

3 Task 2: AS source code modification

This task was skipped in view of the outcome of Task 1.

4 Task 3: VWS vs. AS comparison, SW changes from task 2

This task was skipped in view of the outcome of Task 1.

5 Task 4.1: TMIV Encoder refactoring

The refactorings are introduced by way of simplified code snippets, with enough information to illustrate the concept. The full implementation is available in branch mpeg145-ee1.4.

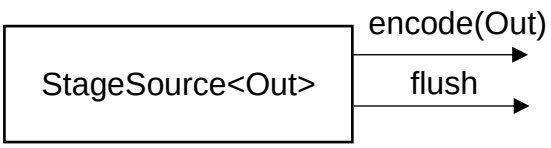
To simplify the encoder multiple sequential refactorings have been performed. These refactorings are introduced in the following sub sections in chronological order. Each of the refactorings have been tested in isolation.

5.1 Introduce encoder stages

The Encoder interface is quite complicated with a combination of methods like pushFrame that follow a *push* design, and methods like popFrame that appear to follow a *pull* design. This refactoring introduces an encoder Stage class that has a *push* design, and applies it to the Encoder as a whole.

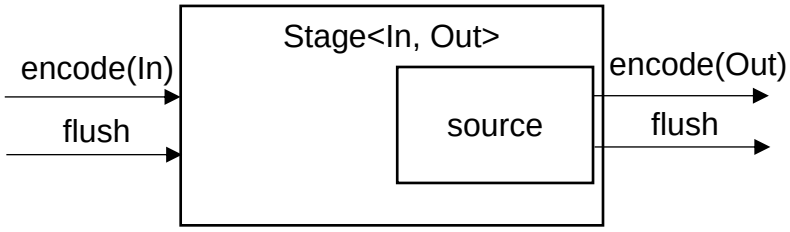
```
// Interface of the input side of an encoding stage
template <typename In> class IStageSink {
public:
  // Input is pushed into this stage using this method
  virtual void encode(In value);

  // End of input is signalled using this method
  virtual void flush() = 0;
};
```


```
// Output side of an encoding stage
template <typename Out> class StageSource {
public:
  // Output is pushed out of this stage into the next one
  void connectTo(IStageSink<Out> &sink);

  // If connected, send a value to the next encoding stage
  void encode(Out value) const;

  // If connected, signal that the last value has been sent
  void flush() const;
};
```



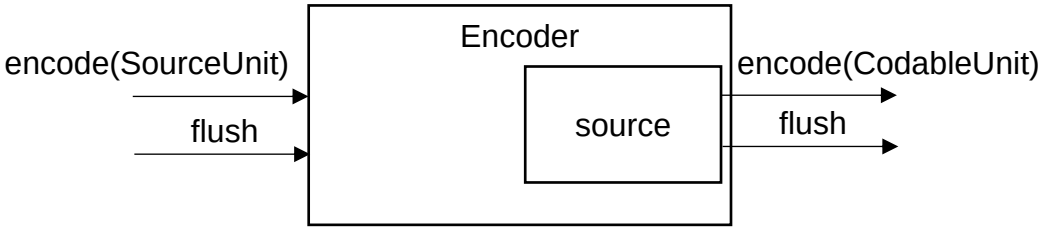

```
// An encoding stage has an input stage interface and an output stage
template <typename In, typename Out> class Stage : public IStageSink<In> {
```

```
public:
  StageSource<Out> source;
};
```



The Encoder is refactored to derive from Stage<SourceUnit, CodableUnit> whereby:

```
struct SourceUnit {
  MivBitstream::SequenceConfig sequenceConfig;
  Common::DeepFrameList deepFrameList;
};

struct CodableUnit {
  EncoderParams encoderParams;
  bool hasAcl{};
  Common::V3cFrameList v3cFrameList;
};
```



The Encoder application no longer has intra period logic. It just sends in all frames:

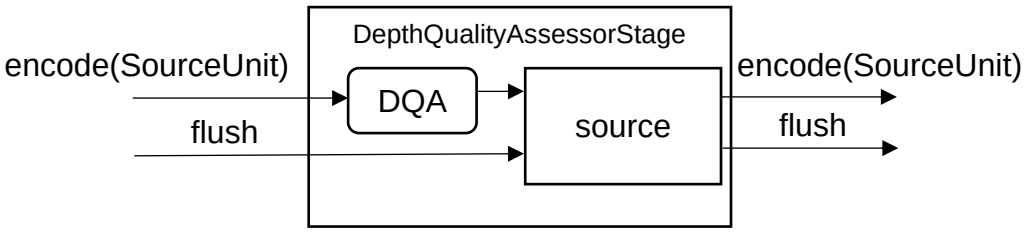
```
  for (int32_t i = 0; i < m_numberOfInputFrames; ++i) {
    m_encoder.encode(
        m_inputSequenceConfig,
        IO::loadMultiviewFrame(..., i));
  }

  m_encoder.flush();
```

The decision to output an IRAP frame (see intraPeriod parameter), a non-IRAP atlas frame (see interPeriod parameter) or only video components is now left to the Encoder.

The Encoder application receives a number of CodableUnit's followed by a call to flush. The hasAcl member, short for "has atlas coded layer", indicates if a codable access unit has common atlas data and atlas data. With MIV it is possible to have frames with only video data, whereby the non-video data is persistent.

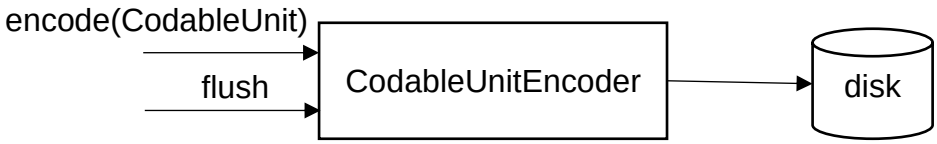
5.2 Extract depth quality assessment

At this point the strategy is to extract stages from the sink-side or source-side of the encoder because this is easier to oversee than splitting the encoder halfway. The first step is to make a stage out of the DepthQualityAssessor. This is easy because it is just a transformation of the SourceUnit to set the depthLowQualityFlag.



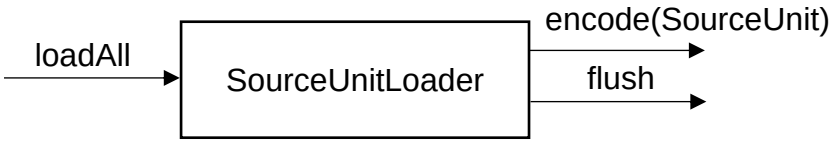
5.3 Codable unit encoder

The encoder main source file (and also the MPI encoder main source file) have logic for outputing the CodableUnit's. The encoder parameters are fed into an MivEncoder to format the MIV bitstream, and the video frames are written to disk as raw YUV files. This part is the same for MVD and MPI and is extracted as a CodableUnitEncoder. This is final stage and thus it is only a sink.



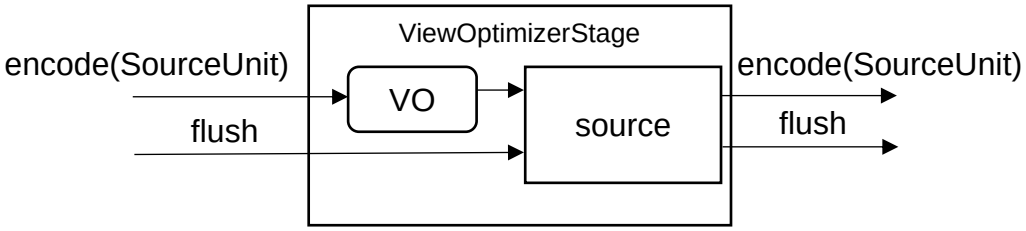
5.4 Source unit loader

The encoder and MPI encoder main source files have logic for loading source units from disk. The input formats are however completely different. In the case of the MVD encoder there was quite some code in the main program, and this was refactored out into a separate class SourceUnitLoader, which is a StageSource. The main purpose of this refactoring is to make the main source file shorter and less detailed.



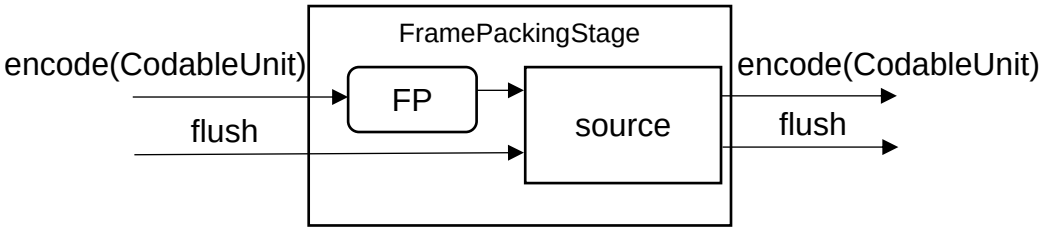
5.5 View optimization stage

When the encoder receives a SourceUnit, one of the first actions was to call the view optimizer to transform the parameters and video frames. This was straightforward to refactor into a separate stage. The main purpose of this refactoring is to make the Encoder smaller: separation of concerns.



5.6 Frame packing stage

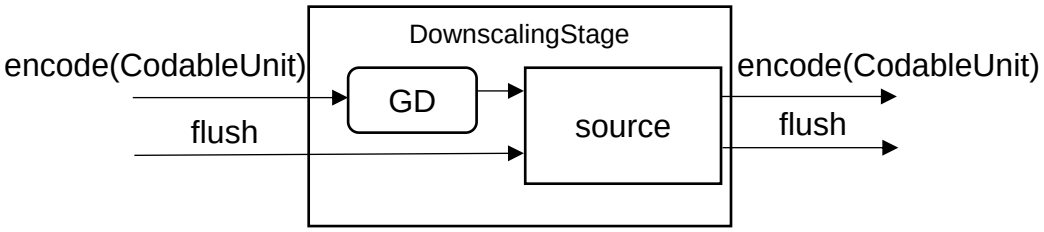
The way that frame packing was implemented in TMIV, first non-packed access units are computed and then the VPS and video frames are converted to frame packed. Because the existing implementation was already written as a per-unit transformation, it was easy to extract this as a stage that transforms a CodableUnit. The main benefit of this refactoring is that this stage can also be used for the MPI encoder which enables MPI + frame packing. This was done and will be described in a section further down.



At this point a design decision was made that for all transformations that are optional, like frame packing but also e.g. geometry quantization is only needed if geometry video is present, the stage itself has the passthrough logic. The MVD and MPI encoder pipelines are fixed: they do not depend on configuration parameters or content.

5.7 Geometry downscaling stage

Just before frame packing geometry downscaling was performed. This is a transformation on individual video frames and was easy to extract as a stage.

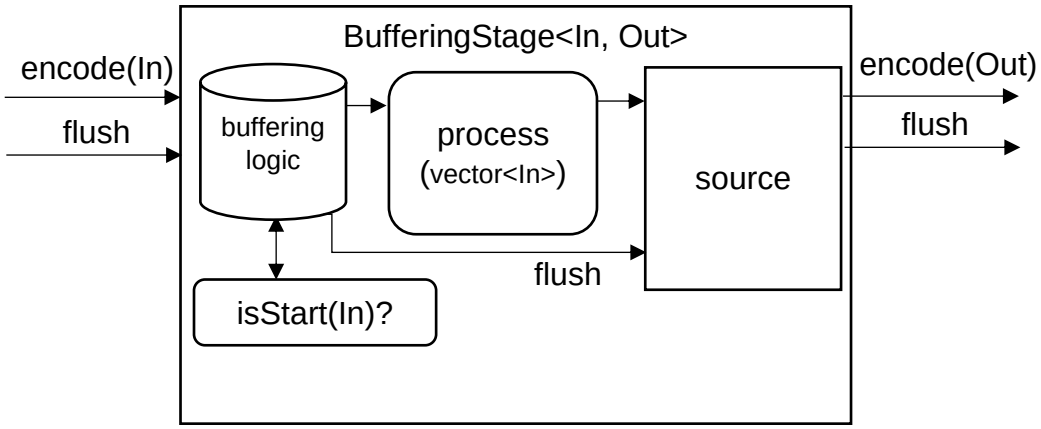


5.8 Quantization stage

Extracting geometry quantization as a separate stage was the first hard refactoring because of multiple details. This fits in with the software development difficulties with CE 2.2..2.4 in the previous cycle. One problem that was identified while working on this stage is that atlas tiling creates duplicate state: patch parameters were present in two locations. Furthermore, geometry quantization takes into account multiple codable units: an IDR or TRIAL followed by multiple SKIP frames. This stage thus needs to buffer units,

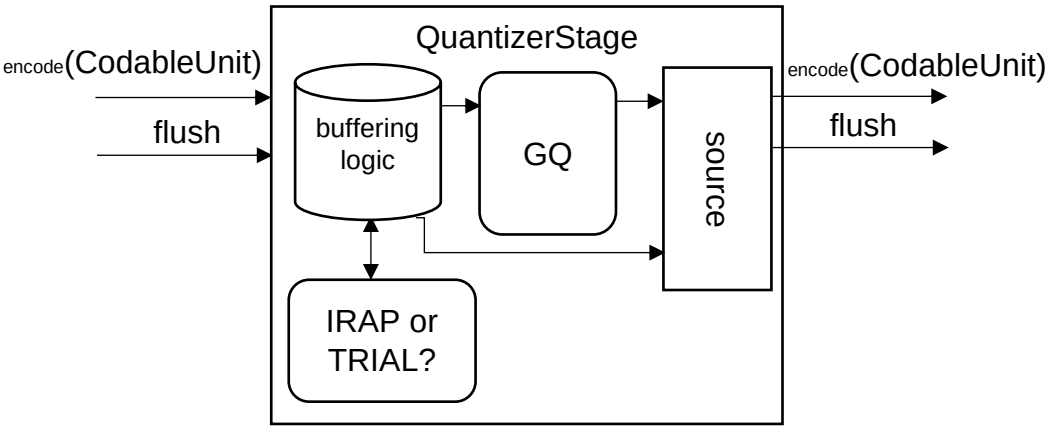
analyse them all at once and then push them to the next stage. All in all, to extract this stage multiple refactorings were performed in order:

1. Introduce CodableUnitType (IDR, TRIAL, SKIP)
2. Unite DeepFrame and V3cFrame
3. Atlas tiling: avoid duplicate state
4. Introduce BufferedStage and apply to Encoder



The buffering stage is a bit more complex. An alternative design was considered whereby the output unit is a vector of codable units. This is conceptually simpler but limited in application. With the current design it is theoretically possible that one stage buffers intra periods (starting with IRAP, followed by TRIAL and SKIP) while another stage buffers "inter" periods (starting with IRAP or TRIAL, followed by SKIP). In other words: each stage has full flexibility in if and how to buffer regardless of preceding and following stages.

Finally, the quantization stage could be extracted. Currently it quantizes occupancy geometry video, hence the more general name was chosen.



5.9 Construct video frames

As part of finding a possible following stage to extract, we refactored the constructVideoFrames and underlying writePatchInAtlas functions because they are long and complicated. The writePatchInAtlas function even exceeded the cognitive complexity metric of clang-tidy. After the refactoring the NOLINT marker could be removed.

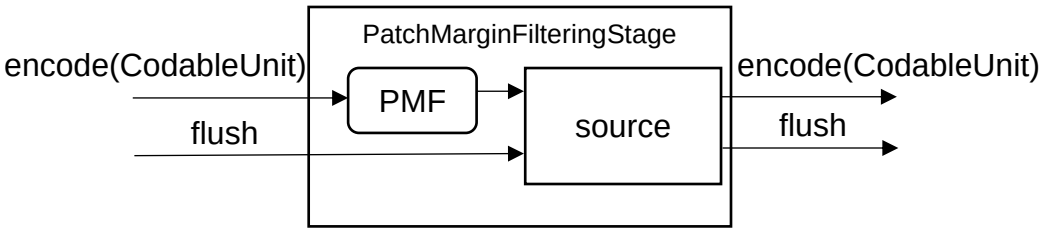
To refactor writePatchInAtlas, the visitor pattern with higher-order functions was applied. A new template function visitPatch accepts a visitor that is calculated by another function. After this the function looks like this:

```
void writePatchInAtlas(const MivBitstream::PatchParams &patchParams,
                       const Common::DeepFrame &view,
                       Common::DeepFrameList &frame,
                       size_t patchIdx) {
  visitPatch(patchParams, writeSampleInPatchIdxMap(patchParams, frame, patchIdx));

  if (m_config.haveGeometry) {
    visitPatch(patchParams, writeOccupancySampleInAtlas(patchParams, view, frame));
    visitPatch(patchParams, writeGeometrySampleInAtlas(patchParams, view, frame));
  }

  if (m_config.haveTexture) {
    visitPatch(patchParams, writeTextureSampleInAtlas(patchParams, view, frame));
  }
}
```

Even now the write.. functions have quite some details, e.g. on when a sample is occupant or not, but at least now the logic for writing patch index map, occupancy, geometry and texture is separated.

5.10 Patch margin filtering stage

At this point it was possible to extract the patch margin filtering stage. The main reason for extracting this stage was to make the Encoder shorter. The patch margin filtering is an operation on individual codable units and was easy to extract.



5.11 Undoing the split of the encoder implementation

At some point in the past the encoder implementation was split over multiple source files because the source file was getting too long. After all the previous refactorings this could be undone. The encoder now has a single header (Encoder.h) and source file (Encoder.cpp). The source file is about 1300 lines long including comments and emtpy

lines. The main motivation for this refactoring is that it is generally easier in IDE's to navigate between a single header and source file.

5.12 MPI encoder

The TMIV MPI encoder is a separate executable with a difference input format (PCS). This refactoring changed the interface of the MpiEncoder to be a BufferingStage and the file output is now handled by the CodableUnitEncoder. This allows for adding the frame packer stage in between, thus enabling testing the combination of the MIV Extended Restricted Geometry profile with frame packing.

As part of this refactoring the participants scanned for differences in configuration parameters between the Encoder and MPI Encoder, and some alignment was performed. Most importantly, it is now possible to set the PTL information to the MPI Encoder as was already possible for the Encoder. This for for instance enables selecting the VVC Main10 codec group IDC.

5.13 Refactorings that could not be performed

At this point the participant decided that enough effort was taken. The following refactorings were tried but are difficult to perform:

5.13.1 Extracting a pruner stage

This is difficult because of entity-based coding. It would require to first extract an entity layer separation stage that outputs something like a SourceLayerUnit. It then requires the Pruner to know about entity layers which violates separation of concerns. A stage may not be good design in this case.

5.13.2 Extracting a packer stage

The interface of the packer has been made more complicated with support for multiple tiles and entity layers. Furthermore, some logic after the packing needs some information that was available prior to packing. A stage may not be a good design in this case.

5.13.3 Extracting texture offsetting stage

Currently the last step of the encoder is to optionally perform texture offsetting. This should be a separate stage, but for some reason the MD5 sums of the E3 and O3 integration tests change when this refactoring is performed.

It is recommended to perform this refactoring anyhow because it removes about 300 lines of code from the Encoder into a new stage.

6 Task 4.2: TMIV Encoder cfg simplification

A single change was made. Unused configuration parameters result in an error before the first frame is loaded. Existing configuration files have been updated to remove unused parameters. This change was made for all TMIV executables that use a JSON configuration file.

7 Task 5: Crosscheck of Task 1 and Task 3

Task 1 was partially crosschecked by PUT (R65 only). The observations are in line with the ones shared by Philips. The difference between R65 results for VWS and AS is significant and favors the view-weighting synthesizer.

Task 3 was skipped, so the crosscheck was skipped too.

8 Recommendations

1. Remove the additive synthesizer (AS) as a viewport renderer
2. Refactor the AS for use in server-side inpainter and pruner
3. Merge mpeg145-ee1.4 into the main branch (after the first AHG call)
4. Extract a texture offsetting stage despite MD5 sum changes
5. Watch best reference pose trace videos

Attachments

tmiv-17.1-main-anchor-vs-17.0.xlsm

Comparison of TMIV 17.0 and TMIV 17.1 with N0377 CTC main anchor conditions, x64 Linux + Clang 17.0.2 toolchain, provided by Philips.

tmiv-17.1-main-anchor-vs-N0377.xlsm

Comparison of the above TMIV 17.1 result against the N0377 reported main anchor results, provided by PUT.

mpeg145-ee1.1-main-anchor.xlsm

Comparison of additive synthesizer (AS) vs. anchor: view-weighted synthesizer (VWS) on TMIV 17.1, with N0377 CTC main anchor conditions, x64 Linux + Clang 17.0.2 toolchain, provided by Philips.

mpeg145-ee1.1-best-reference.xlsx

Comparison of additive synthesizer (AS) vs. reference: view-weighted synthesizer (VWS) run times on N0377 CTC best reference conditions, x64 Linux + Clang 17.0.2 toolchain, provided by Philips.