# Introduction

This document describes test conditions and explains how to run a test to evaluate objectively V-PCC for 3GPP Beyond\_2D studies.

# Sources and prerequisites

## Point cloud generation

In a first step, the user shall generate all point cloud frames using the procedure described in document “[3GPP BEYOND\_2D] documentation on ply\_generation.docx”.

## Prerequisites

To use this framework, the user will need to install the following softwares:

* Python 3.11
* CMAKE 3.26 or above (but under 4.0.)
* A bash command line tool to run the scripts (on Windows, GitBash or MobaXTerm for example)

Please use a [python virtual environment](https://docs.python.org/3/library/venv.html#creating-virtual-environments) to install dependencies and run the scripts. A requirements.txt file is provided such that a suitable virtual environment can be set-up as follows:

python3 -m venv venv

venv\Scripts\activate # on Windows

. venv/bin/activate # on Linux

python -m pip install –upgrade pip

pip install -r requirements.txt

# Framework description

The aim of the framework is to encode, decode, extract objective metrics for test configuration.

The MPEG V-PCC test model [1] will be used to encode and decode test sequences. The MPEG mmetric software [2] will be used to compute metrics. These tools are automatically downloaded and built by the script (see section IV).

Python scripts are provided to be able to run a complete evaluation and are described in section IV.

Below are examples of typical command lines for a vox11 sequence:

## Encode a test sequence

The command in Table 1 encodes one stream named “${test\_sequence}.bin”:

|  |
| --- |
| mpeg-pcc-TMC2/bin/PccAppEncoder \ |
| --config=mpeg-pcc-TMC2/cfg/common/ctc-common.cfg \ |
| --config=mpeg-pcc-TMC2/cfg/condition/ctc-random-access.cfg \ |
| --config=mpeg-pcc-TMC2/cfg/sequence/${test\_sequence}.cfg \ |
| --configurationFolder=mpeg-pcc-TMC2/cfg/ \ |
| --uncompressedDataFolder=${source\_sequence}/ \ |
| --normalDataPath=${source\_sequence}/${source\_sequence}\_%04d.ply \ |
| --compressedStreamPath=${test\_sequence}.bin \ |
| --frameCount=$frame\_number \ |
| --resolution=2047 \ |
| + additional\_parameters |

Table 1 Typical encoding command

*Configuration files*

TMC2 requires input configuration files. Notably the condition configuration file for using TMC2 in Random Access (RA) mode is provided by the reference software as with ctc-random-access.cfg.

For each tested sequence, a configuration file (test\_sequence.cfg in Table 1) is provided to the TMC2 software. This configuration files are installed during the installation test process described in section IV and are located in TMC2 folder under external\_data/sequence\_cfg. Table 2 lists the configuration files identified for 3GPP sequences.

|  |  |  |
| --- | --- | --- |
| Sequence | Seq Id | Sequence CFG file |
| Mitch | S01 | mitch\_vox11.cfg |
| Juggle Soccer | S02 | jugglefootrouge\_vox11.cfg |
| Henry | S03 | henry\_vox11.cfg |
| Nathalie | S04 | nathalie\_vox11.cfg |
| Aliyah | S05 | aliyah\_vox11.cfg |

Table 2 List of sequence configuration files

*Uncompressed Data Folder*

The uncompressedDataFolder specifies the path to the PLY sources, named ${source\_sequence} in the example. This path is generated by the script using a JSON file (see section IV).

*Normal Data Path*

The NormalDataPath specifies the path to the normal information of the source sequence, needed for metrics computation by TMC2. It is named ${source\_sequence}/${source\_sequence}\_%04d.ply in the example. This path is generated by the script using a JSON file (see section IV).

*Compressed stream Path*

The CompressedStreamPath specifies the path to the bitstream generated by TMC2, named ${test\_sequence}.bin in the example. This path is generated by the script using a JSON file (see section IV).

*Frame number information*

For all sequences, the first ten seconds of content are processed, except for the Juggle Soccer sequence, which has only 125 frames as shown in Table 3. This defines the frameCount needed for TMC2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sequence | Seq Id | Content Provider | FPS | Number of frames |
| Mitch | S01 | Volucap | 25 | 250 |
| Juggle Soccer | S02 | XD Productions | 25 | 125 |
| Henry | S03 | Render People | 30 | 300 |
| Nathalie | S04 | Volucap | 30 | 300 |
| Aliyah | S05 | Render People | 30 | 300 |

Table 3 V-PCC frame number parameters set for each sequence

*Resolution*

The resolution is set to 2047 for vox11.

*Additional parameters*

Additional parameters are described as the concatenation of two pieces of information:

* **Profile information**: In line with the V-PCC verification test [3], the following profile is evaluated “Basic which corresponds to V-PCC profile HEVC Main10 V-PCC Basic Rec0. This profile uses the basic V-PCC toolset and a simple reconstruction of points in 3D space.”

Options to add for profile configuration are described in Table 4.

|  |  |  |
| --- | --- | --- |
| V-PCC Profile | Name | Options |
| V-PCC HEVC Main10 V-PCC Basic Rec0 | Basic | "--profileToolsetIdc=0",  "--profileReconstructionIdc=0",  "--mapCountMinus1=0" |

Table 4 V-PCC profile options

* **Bitrates information**: In line with the V-PCC verification test [3], the following targeted rates are evaluated [5,10,20,30,50] Mps.

For each sequence and profile, encoder parameters have been established and summarized in Table 5 below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rate | Target Bitrate (mbps) | S01 Mitch | | | S02 Juggle Soccer | | | S03 Henry | | | S04 Nathalie | | | S05 Aliyah | | |
|  |
|  |  | QP Geo | QP Att | Occ Prec | QP Geo | QP Att | Occ Prec | QP Geo | QP Att | Occ Prec | QP Geo | QP Att | Occ Prec | QP Geo | QP Att | Occ Prec |  |
| R01 | 5 | 29 | 33 | 4 | 30 | 39 | 4 | 23 | 34 | 4 | 25 | 39 | 4 |  |  |  |  |
| R02 | 10 | 23 | 29 | 2 | 19 | 35 | 2 | 15 | 30 | 2 | 24 | 30 | 4 |  |  |  |  |
| R03 | 20 | 19 | 25 | 2 | 11 | 28 | 2 | 8 | 26 | 2 | 20 | 26 | 4 |  |  |  |  |
| R04 | 30 | 15 | 23 | 2 | 9 | 24 | 2 | 7 | 23 | 2 | 18 | 24 | 2 |  |  |  |  |
| R05 | 50 | 11 | 21 | 2 | 5 | 21 | 2 | 6 | 20 | 2 | 17 | 21 | 2 |  |  |  |  |

Table 5 V-PCC encoder parameters (qp geometry, qp attribute, occupancy precision) per sequence per rate per profile

## Decode of a V-PCC bitstream

The command in Table 6 decodes a V-PCC stream to generate PLY files named “${test\_sequence}\_dec\_%04d.ply”:

|  |
| --- |
| mpeg-pcc-TMC2/bin/PccAppDecoder \  --startFrameNumber=0 \  --compressedStreamPath=${test\_sequence}.bin \  --reconstructedDataPath=${test\_sequence}\_dec\_%04d.ply \  --inverseColorSpaceConversionConfig=mpeg-pcc-TMC2/cfg/hdrconvert/yuv420toyuv444\_16bit.cfg |

Table 6 Typical decoding command

*Start Frame Number*

The startFrameNumber specifies the initial frame index for decoding. This value is generated by the script using a JSON file (see section IV).

*Compressed stream Path*

The CompressedStreamPath corresponds to the path of the bitstream produced by TMC2, named ${test\_sequence}.bin. This path will be generated by the script thanks to a JSON file (see section IV).

*Reconstructed Data Path*

The reconstructedDataPath denotes the location of the PLY output files, named ${test\_sequence}\_dec\_%04d.ply. This path is generated by the script thanks to a JSON file (see section IV).

*Configuration file for inverse color space conversion*

To perform colour conversion, TMC2 requires the configuration file yuv420toyuv444\_16bit.cfg, provided by the reference software.

## Compute objective metrics for a test sequence

The command in Table 7 calculates metric data using the source and decoded PLY files with the MPEG mmetric software [2]. For a given sequence, it first computes PCC metrics, followed by PCQM metrics. The results are stored in a log file that contains all the metric information.

|  |
| --- |
| mpeg-pcc-mmetric/build/Release/bin/mm sequence \  --firstFrame 0 \  --lastFrame (frame\_number -1) END \  compare --mode pcc \  --inputModelA source\_sequence\_%04d.ply \  --inputModelB test\_sequence\_dec\_%04d.ply END \  compare --mode pcqm \  --inputModelA source\_sequence\_%04d.ply \  --inputModelB test\_sequence\_dec\_%04d.ply |

Table 7 Typical metric computation command

*First Frame Number*

The firstFrame specifies the initial frame index for metric computation. This value is generated by the script using a JSON file (see section IV).

*Last Frame Number*

The lastFrame specifies the final frame index for metric computation. This value is generated by the script using a JSON file (see section IV).

*Model A*

The inputModelA indicates the path to the PLY source path, named ${test\_sequence}\_%04d.ply. This path will be generated by the script using a JSON file (see section IV).

*Model B*

The inputModelB indicates the path to the PLY output path, named {test\_sequence}\_dec\_%04d.ply. This path will be generated by the script using a JSON file (see section IV).

## Generate CSV file and graph workbook

Based on the process description in section E, the output log containing all metrics information is used to extract metrics and a build CSV file. Each CSV file concates metrics information for each condition and one profile is generated for all sequences and rates.

The following information will be stored in the CSV file:

* SeqId: identifier of the sequence
* CondId: tested condition (RA)
* RateId: tested rate number [R1..R5]
* nbFrame: number of tested frames
* NbInputPoints: number of points in the source sequence
* NbOutputPoints: number of points in the candidate test sequence
* MeanOutputPoints: mean number of points in the candidate test sequence
* MeanDuplicatePoints: mean number of duplicated points (with same geometry) in the candidate test sequence
* TotalBitstreamBits: size of the bistream in bits
* geometryBits: size of the geometry stream in bits
* metadataBits: size of the metadata stream in bits
* attributeBits: size of the attribute stream in bits
* D1Mean: mseF,PSNR (p2point)
* D2Mean: mseF,PSNR (p2plane)
* LumaMean: c[0],PSNRF
* CbMean: c[1],PSNRF
* CrMean: c[2],PSNRF
* PCQM: PCQM PSNR
* SelfEncoderRuntime: encoder time for current process
* ChildEncoderRuntime: encoder time for child processes
* SelfDecoderRuntime: decoder time for current process
* ChildDecoderRuntime: decoder time for child processes

From the CSV file, an excel spreadsheet is generated from the template xlsm sheet (in the “templates” directory) to get tables and graphs for interpretation of the results.

CSV and workbook files are automatically generated by the script (see section IV).

# Perform the evaluation

Python scripts are provided to:

* Build the test environment under the output “dependencies” directory
* Perform tests, including:
  + Encode each sequence for each condition, rate and profile
  + Decode the corresponding sequence
  + Compute metrics
  + Generate CSV tables and graph worksheets

To proceed with the test, the user should navigateto the “ply\_to\_bin/” directory, which contains:

* \*.py : Python scripts to encode, decode, compute metrics and generate CSV and XLSM workbooks
* templates/ : Directory with template XLSM sheet used for graph generation
* jsons/ : Directory with configuration examples
  + sequences.json: Describes the list of input sequences to test. It contains information on the location of point clouds and has to be updated by the user. It also has information on the name of the configuration file used for the encoding step (${test\_sequence}.cfg)
  + 3gpp\_test\_configuration.json : Describes the test lists to perform. For each profile, it defines the encoding parameters as described in Table 4, the number of frames to test (typically 300) and the list of sequences to be tested. This list includes:
    - The “id” corresponding to the one set in the sequences.json file
    - The condition to test, here, random access
    - A list of 5 rates as defined in Table 5

A script “exec\_binGenerator.py” is provided to automate all steps described in Section III. It can be launched from your Python environment with the following command:

python exec\_binGenerator.py -o $YOUR\_OUTPUT\_DIR -i jsons/sequences.json -t jsons/3gpp\_test\_configuration.json

for help on the script, use python3 exec\_binGenerator.py –-help

usage: exec\_binGenerator.py [-h] -i SEQUENCEJSON -o OUTPUTDIR -t TESTCONFJSON

This script generates point cloud frames (PLY) from meshes (OBJ+TXT)

options:

-h, --help show this help message and exit

-i SEQUENCEJSON, --sequenceJson SEQUENCEJSON

Json that contains the sequence to be done

-o OUTPUTDIR, --outputDir OUTPUTDIR

Output BIN directory

-t TESTCONFJSON, --testConfJson TESTCONFJSON

Json that contains the test configuration

The output directory structure is:

* cmd: Directory with job command and logs
* dependencies: Compilation of TMC2 and mmetric software used to perform the test
* A list “Fyy\_ProfileName” directories with Fyy corresponds to the number of tested frames, ProfileName corresponds to the tested profile and includes generated bitstreams
* A list of CSV files with extracted metric information per profile for a given number of frames
* Excel worksheets with graphs per profile for a given number of frames

# Verification / crosschecks

All produced bitstreams, metric results and videos produced will be cross-checked by at least one other SA member to ensure that results are correct.

# References

[1] MPEG V-PCC Test Model v25, [GitHub - MPEGGroup/mpeg-pcc-TMC2: Video codec based point cloud compression (V-PCC) test model](https://github.com/MPEGGroup/mpeg-pcc-tmc2), tag “release-v25.0”

[2] MPEG MM Metric Tool 1.1.7, [MPEGGroup/mpeg-pcc-mmetric: MPEG PCC Mesh metric](https://github.com/MPEGGroup/mpeg-pcc-mmetric) , tag “1.1.7”

[3] MPEG, Subjective verification test report for V-PCC, <https://www.mpeg.org/wp-content/uploads/mpeg_meetings/136_OnLine/w20992.zip>