# Introduction

This document explains how to generate videos from raw point clouds or encoded V-PCC bitstreams using the “bin\_to\_video” framework.

# Prerequisites

## Dependencies

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

* Python 3.11 and the gitpython package
* 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)

This framework will download, build and use the following publicly available repositories:

* [MPEG 3DG Renderer](https://github.com/MPEGGroup/mpeg-3dg-renderer) (commit “c1e09f8”)
* [MPEG V-PCC Test Model (TMC2)](https://github.com/MPEGGroup/mpeg-pcc-tmc2) (branch “release-v25.0”)

## Input generation

This framework takes either a folder containing point clouds .ply files or a V-PCC bitstream as input. Please refer to readme\_ply\_generation.docx and readme\_ply\_to\_bin.docx for information on how to generate those inputs.

## Input files

This framework uses .json files to locate the data needed for the video generation among other things.

### Configuration file

The configuration JSON file is structured as follows:

|  |
| --- |
| {  "nb\_th\_dec" : 8, <- number of decoding threads  "nb\_th\_renderer" : 8, <- number of render threads  "width": 1920, <- output video resolution (width)  "height": 1080, <- output video resolution (height)  "render\_args": "-n 300 --play=1 --playBackward=1 --overlay=0 --visible=0", <- renderer arguments common to all render jobs  "render\_jobs": [ <- list of render jobs (1 job = 1 video / stream)  {  "name": "cube\_size1", <- job name (used in the final file name)  "args": "--floor=1 --type=0 --size=1" <- renderer arguments specific to this job  },  {  "name": "blend\_size2.4\_alpha1.8\_linear",  "args": "--floor=1 --type=3 --alphaFalloff=1.8 --size=2.4 --blendMode=1"  },  {  "name": "bck\_blend\_size2.4\_alpha1.8\_linear",  "args": "--type=3 --alphaFalloff=1.8 --size=2.4 --blendMode=1",  "use\_background": 1 <- show background (optional, default to 0)  }  ]  } |

Please see the MPEG 3DG Renderer documentation for the list of all available arguments for the renderer

### Test file

The test JSON file is structured as follows:

|  |
| --- |
| {  "TestList": [  {  "Name": "Basic\_S1C2RAR0001\_mitch11", <- name of the test (output video filename prefix)  "PathEnc": "encoded/S1C2RAR0001\_mitch11\_enc.bin", <- v-pcc bitstream path  "PathDec": "Mitch/Basic\_S1C2RAR0001\_mitch11", <- decoded ply path  "PathVid": "Videos/Mitch", <- video path  "CameraPath": "camerapath/Mitch.txt", <- mpeg renderer camerapath file  "Config": "background/skatepark\_linux.txt", <- mpeg renderer background config file  "FPS": 25 <- video/sequence FPS  },  {  "Name": "Basic\_S3C2RAR0001\_henry11",  "PathEnc": "encoded/S3C2RAR0001\_henry11\_enc.bin",  "PathDec": "Henry/Basic\_S3C2RAR0001\_henry11",  "PathVid": "Videos/Henry",  "CameraPath": "camerapath/Henry.txt",  "Config": "background/station\_henry\_linux.txt",  "FPS": 30  }  ]  } |

Notes:

The **PathEnc, PathDec** and **PathVid** can be defined either as **absolute paths**, or paths **relative to the output directory** provided to the python script (-o argument)

The **CameraPath** and **Config** can be defined as either **absolute paths** or paths **relative to the directory of this test JSON file.**

# Framework

## Overview

This framework generates 2D videos of point cloud sequences with predefined camera paths, either directly from point cloud files (.ply) or from a V-PCC bitstream.   
  
The framework uses the MPEG V-PCC test model to decode and extract the individual point cloud frames from a V-PCC bitstream and uses the mpeg-3dg-renderer to generate videos from the .ply files.

The MPEG renderer generates uncompressed .rgb videos.

The MPEG test model and MPEG renderer are automatically cloned and built when running the framework for the first time.

## Main executable script

### Overview

This framework provides the executable script bin\_to\_video/exec\_binToVideo.py, which decode V-PCC bitstreams and generate videos.

This script takes the following arguments:

-h, --help

show the help message and exit

-c [CONFIGJSON], --configJson [CONFIGJSON]

(REQUIRED) Path to the .json file containing the general configuration, including the rendering jobs

-i [INPUTJSON], --inputJson [INPUTJSON]

(REQUIRED) Path to the .json file containing the input bitstreams and output videos informations

-o [OUTPUTDIR], --outputDir [OUTPUTDIR]

(REQUIRED) Path to the output directory

-d, --decodeOnly

Run vpcc decoding only (mutually exclusive with --videoOnly)

-v, --videoOnly

Run video generation only (mutually exclusive with --decodeOnly)

-f, --force

Force decoding and/or video generation, even if files already exist

--noRun

Parse the input files, but don't execute the commands (typ. to generate the scripts)

-s, --scripts

Export the generated commands to bash scripts in the output directory.

--scriptsMode {full,test,job}

Set the script export mode:

full : Export all decoding commands in a dec.sh file and

all the video generation commands in a vid.sh files.

test (default) : Export all the commands for individual streams in

dec\_<stream\_id>.sh and vid\_<stream\_id>.sh.

job : Same as TEST for decoding commands, export the

video generation commands for each render job in

vid\_<stream\_id>\_<job\_id>.sh

### Usage

This framework comes with example configuration files, available in the bin\_to\_video/jsons folder. Some information in those files need to be edited by the user:

* the path to the source ply files in S1S2S3S4S5\_src.json
* the path to the backgrounds meshes in all the .txt files in the background folder

Once the user has updated the necessary path, the following command can be executed:

To generate videos for the source:

|  |
| --- |
| python **<path\_to\_the\_repo>**/bin\_to\_video/exec\_binToVideo.py \  -c **<path\_to\_the\_repo>**/bin\_to\_video/jsons/3gpp\_test\_configuration.json  -i **<path\_to\_the\_repo>**/bin\_to\_video/jsons/3gpp\_selection\_src.json  -o **<path\_to\_your\_output\_dir>**  -v |

To decode the bitstreams and generate the associated videos :

|  |
| --- |
| python **<path\_to\_the\_repo>**/bin\_to\_video/exec\_binToVideo.py \  -c **<path\_to\_the\_repo>**/bin\_to\_video/jsons/3gpp\_test\_configuration.json  -i **<path\_to\_the\_repo>**/bin\_to\_video/jsons/3gpp\_selection\_dec.json  -o **<path\_to\_your\_output\_dir>** |

## Custom scripting

### Overview

This framework provides python classes needed to decode V-PCC bitstreams and generate videos which can be used in custom scripts in cases where the user would want to create a custom pipeline (to use grid computing for example).

The architecture is based around the ConfigManger class, responsible for parsing the configuration files and storing the configuration information, and functional classes using that information to generate the bash commands needed for the various steps, with API to either directly execute those commands or export them in files for later execution.

The exec\_binToVideo.py script leverage this architecture and can serve as a baseline for custom scripts.

ConfigManager

Class to parse the configuration and test files and store the relevant information.

Usage:

|  |
| --- |
| #!/usr/bin/python3  from ConfigManager import ConfigManager  cm = ConfigManager(confJsonPath, testJsonPath, outputDir) |

Note: The constructor will clone and build the needed dependencies if not present in the output directory.

### VpccDecoder

Class elaborating the TMC2 commands to decode the bitstreams

Usage for running all tests sequentially

|  |
| --- |
| #!/usr/bin/python3  from ConfigManager import ConfigManager  from VpccDecoder import VpccDecoder  cm = ConfigManager(confJsonPath, testJsonPath, outputDir)  dec = VpccDecoder(cm)  #Export all the commands in a single bash file (optional)  dec.toFile("pathToExportTheDecoderCommandsTo.sh")  #Execute all the commands sequentially in a bash command prompt  dec.run() |

Usage for exporting each test commands in individual \*.sh files

|  |
| --- |
| #!/usr/bin/python3  from ConfigManager import ConfigManager  from VpccDecoder import VpccDecoder  cm = ConfigManager(confJsonPath, testJsonPath, outputDir)  #in path, generate PREFIX\_0.sh, PREFIX\_1.sh… dec.toSeparateFiles("path/PREFIX") |

### VideoGenerator

Elaborate the 3DG Renderer commands to generate the videos

Usage for running all tests sequentially

|  |
| --- |
| #!/usr/bin/python3  from ConfigManager import ConfigManager  from VideoGenerator import VideoGenerator  cm = ConfigManager(confJsonPath, testJsonPath, outputDir)  vid = videoGenerator(cm)  #Export all the commands in a single bash file  #(one per test per render job)  vid.toFile("pathToExportTheRenderingCommandsTo.sh")  #Execute all the commands sequentially in a bash command prompt  vid.run |

Usage for exporting each test or job commands in individual \*.sh files

|  |
| --- |
| #!/usr/bin/python3  from ConfigManager import ConfigManager  from VideoGenerator import VideoGenerator  cm = ConfigManager(confJsonPath, testJsonPath, outputDir)  #export each test command (multiple jobs) as PREFIX\_0.sh PREFIX\_1.sh…  dec.toSeparateFilesTest("path/PREFIX")  #export each jobs as PREFIX\_0\_0.sh PREFIX\_0\_1.sh… following a  # \_{test\_id}\_{job\_id} naming convention  dec.toSeparateFilesJobs("path/PREFIX") |