

# Prerequisite

---

## Requirements

---

This lists the software requirements for executing the experiments. Note that the instruction are given for an Ubuntu (or linux based) system.

- Octave
  - `sudo apt-get install octave`
- Python

## Setting up

---

### Octave scripts

The spatial decomposition of the yuv files is performed as a pre encoding step using Octave scripts and functions in the folders `scripts/` and `functions/` respectively.

The Octave scripts may call Octave functions. To this end, each script includes the path `~/Octave/video-tools`. As a result, the following commands needs to be executed. Note that this must be done with the UNIX user who is going to execute the experiment scripts later on.

```
mkdir ~/Octave
ln -s /path/to/experiments/functions ~/Octave/video-tools
```

# Temporal Multiplexing

---

## Experiment A

---

### Test set preparation

The experiment requires 3 test sets, namely :

- A : The original JVET test set
- B : The half-downscaled version of the original JVET test set
- C : The temporally framepacked version of the JVET test set

## Half-downscaled JVET test set (B)

1. The JVET test set is stored in `JVET_YUV_FOLDER`.
2. Execute `batch_half_downscaled.py` in its own folder with as arguments the path to the JVET test set folder followed by the output folder as shown below:

```
cd scripts
./batch_half_downscaled.py JVET_YUV_FOLDER OUTPUT_FOLDER
```

3. Collect all the `*_half.yuv`, they constitute the test set B.

## Temporal framepacked test set (C)

1. The yuv files of the JVET test set A are stored in `JVET_YUV_FOLDER`.
2. Execute `batch_temporal_framepacking.py` in its own folder with as arguments the path to the JVET test set folder followed by the output folder as shown below:

```
cd scripts
./batch_temporal_framepacking.py JVET_YUV_FOLDER OUTPUT_FOLDER
```

3. Collect all the `*_tfp.yuv`, they constitute the test set C.
4. Prepare the CTC config files of each sequence, by dividing the width and height by half and by multiplying the frame rate by 4.

## Encoding runs

1. Encode A according to CTC
2. Encode B according to CTC + `ConformanceWindowMode=1`
3. Encode C according to CTC + `ConformanceWindowMode=1`

Notes:

- `ConformanceWindowMode=1` is to accommodate resolutions which are not multiple of the min CU size (8), e.g. 540 pixels padded to 544 pixels.
- The "all intra" are modified as explained below.

## All intra

According to the CTC, a temporal subsampling for the "all intra" encodings is performed, i.e. only every 8th frame is coded. However, this temporal subsampling collides with the temporal multiplexing of this experiment. Such a value would lead to encode every 2nd frame of the first resolution component, leaving out the other resolution components.

As a result, the parameter `TemporalSubsampleRatio` is removed from the CTC config files for "all intra", hence disabling the subsampling.

# Post processing

Once C is encoded, the high resolution output of each encoded sequences needs to be generated, especially to compute the PSNRs.

1. The decoded yuv files of the test set C are stored in `JVET_TFP_YUV_FOLDER` .
2. Execute `batch_temporal_framedepacking.py` in its own folder with as arguments the path to `JVET_TFP_YUV_FOLDER` followed by the output folder as shown below:

```
cd scripts
./batch_temporal_framedepacking.py JVET_YUV_FOLDER OUTPUT_FOLDER
```

3. Collect all the `*_highres.yuv` , they constitute the high resolution output of the temporally frame-packed sequences.
4. Compute the PSNR for the `*_highres.yuv` against the original yuv files of A.

*TODO:* Provide tools to compute the PSNR values.

## BD-rate comparisons

In the following section, bitrate and PSNR variables are indexed by 1, 2 or 3, respectively for encodings 1, 2 and 3.

In addition, encoding 3 is sub indexed by `h` or `l` , respectively for the high resolution output and the low resolution output of the sequence.

### Commparation 1 : Against JEM baseline

This comparison shows the impact of temporally packing frames in terms of BD-rate cost. It does not aim at proving the relevancy of the technique but providing some insight based on quantitative results.

`bitrate_ref = bitrate_1`

`bitrate_test = bitrate_3`

`PSNR_ref = PSNR_1`

`PSNR_test = PSNR_3h`

### Commparation 2 : Against simulcast

The reference for this comparison is the combination of encodings 1 and 2, i.e., what is known as simulcasting. Since the technique aims at providing two output resolutions in a single bitstream, the technique must be more efficient than sending two bitstreams, one for each resolution.

`bitrate_ref = bitrate_1 + bitrate_2`

bitrate\_test = bitrate\_3

PSNR\_ref = PSNR\_1

PSNR\_test = PSNR\_3h