Video Processing

software engineering exercise

Introduction

Goal

- to give you basic knowledge in video processing
- the task is simple and can be solved even by the beginners
- at the same time, developing efficient solution is quite complicated and that provides great opportunity for experienced developers to show their skills

What you will know

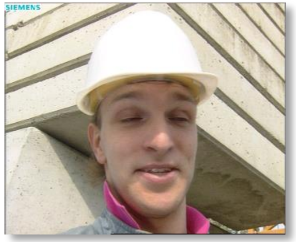
- in the beginning there were raw pixels, so you will study color formats
- then you will change them a bit - resolution change (aka resize, up/down scale, resampling and so on)
- then you will study the most basic principle of video coding that is used in all contemporary standard to remove temporal redundancy - motion estimation

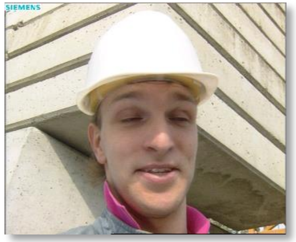
Schedule

- first meeting - problem description, overview of motion estimation tools, general algorithm for solving the problem
- then one week for algorithm development
- second meeting - QA session, draft of SAS should be ready
- one more week to finalize SAS
- four weeks for development, on demand meetings are welcome
- benchmarking
- final meeting, winner announcement

Task

Shuffler



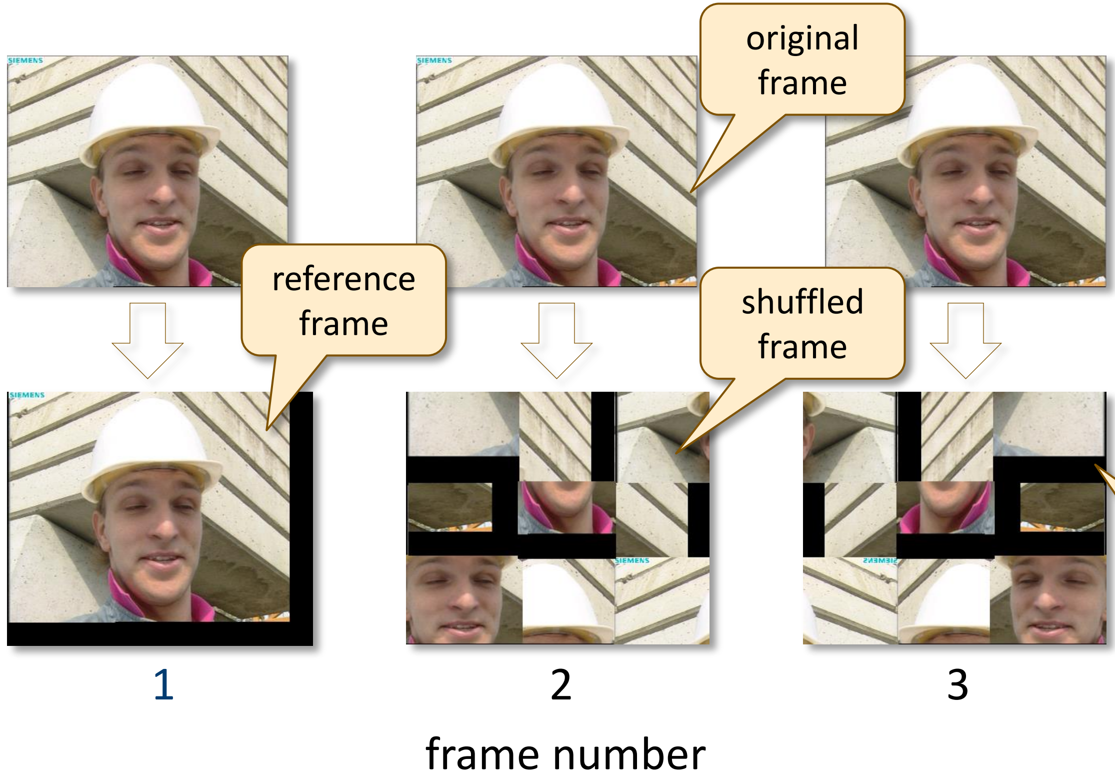


- common YUV stream is used as shuffler input

1 2 3

frame number

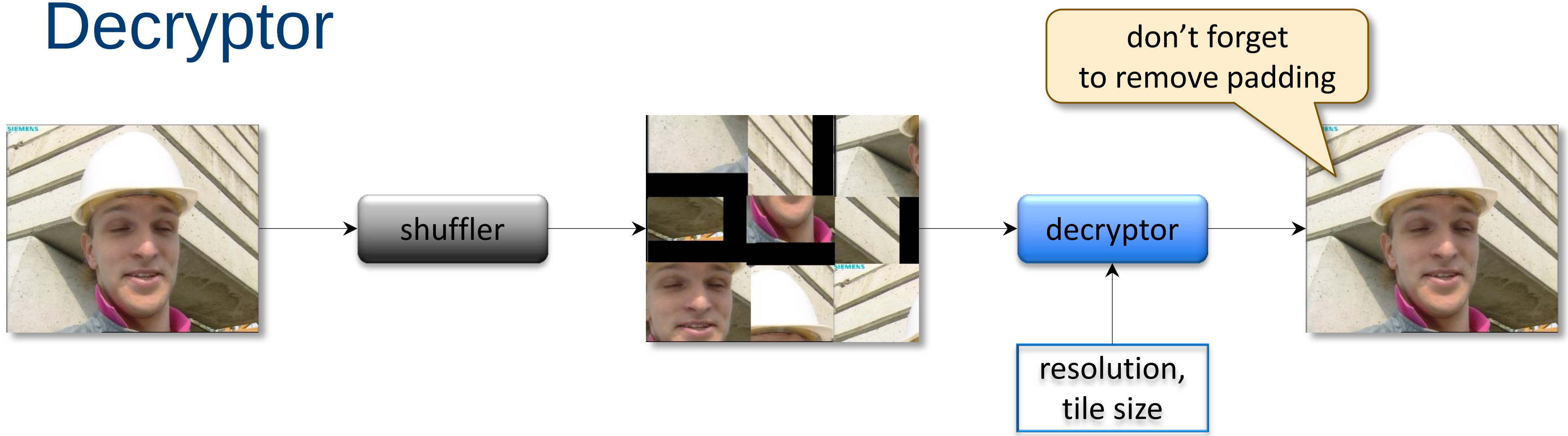
Shuffler



- it leaves first frame as is
- then breaks all other frames starting from the second one into rectangular tiles and shuffle these tiles inside the frame

note black areas, they are caused by padding to align frame size to tile size

Decryptor



- you have to restore original stream from shuffled one, you have to write “the decryptor”
- shuffler will be provided as binary
- output stream should be bit exact to input
- author of the fastest decryptor will be the winner of this exercise

Constrains

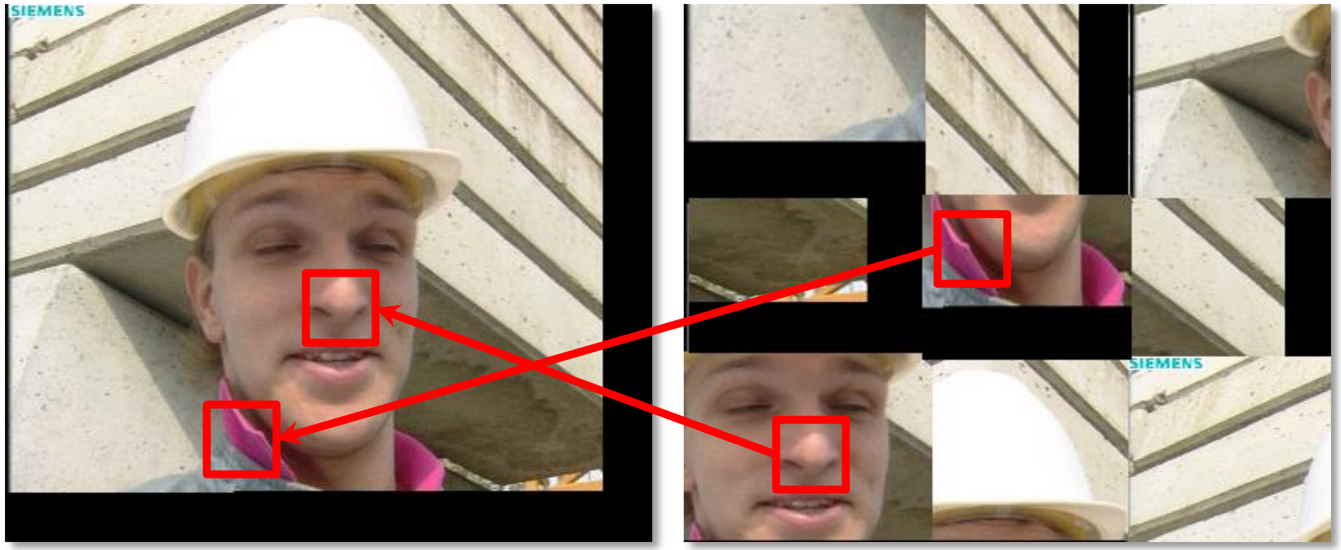
- use current Linux environment
- base solution on motion estimation
- use sample YUV reader/writer, don’t invent your own
- use FEI PreENC for ME
- use VPP for scaling
- don’t touch data in video memory except for reshuffling

Solution

Definitions

- macroblock (MB) – block of 16x16 pixels, used in motion estimation
- tile – block of 64x64 pixels, these blocks are shuffled
- MVP – motion vector predictor, location from which to start motion estimation

First step, motion estimation



```
MV[0] = {260, 24}; D[0] = 323;
MV[1] = {120, 72}; D[1] = 21;
MV[2] = {132, 68}; D[2] = 18;
. . .
```

- use first frame as reference
- use PreENC for motion estimation
- estimate motion for each 16x16 MB
- you will get lots of MVs and distortion for each MB

Second step, calculate tile MV

```
MV[0] = {260, 24}; D[0] = 323;
MV[1] = {120, 72}; D[1] = 21;
MV[2] = {132, 68}; D[2] = 18;
MV[3] = {116, 76}; D[3] = 43;
```

- we will use real life streams, so don’t expect zero distortions and perfect MVs match, you have to invent algorithm to calculate tile MV

Calculate tile MV, cont.

```
MV[0] = {260, 24}; D[0] = 323;
MV[1] = {120, 72}; D[1] = 21;
MV[2] = {132, 68}; D[2] = 18;
MV[3] = {116, 76}; D[3] = 43;
```

bad MV

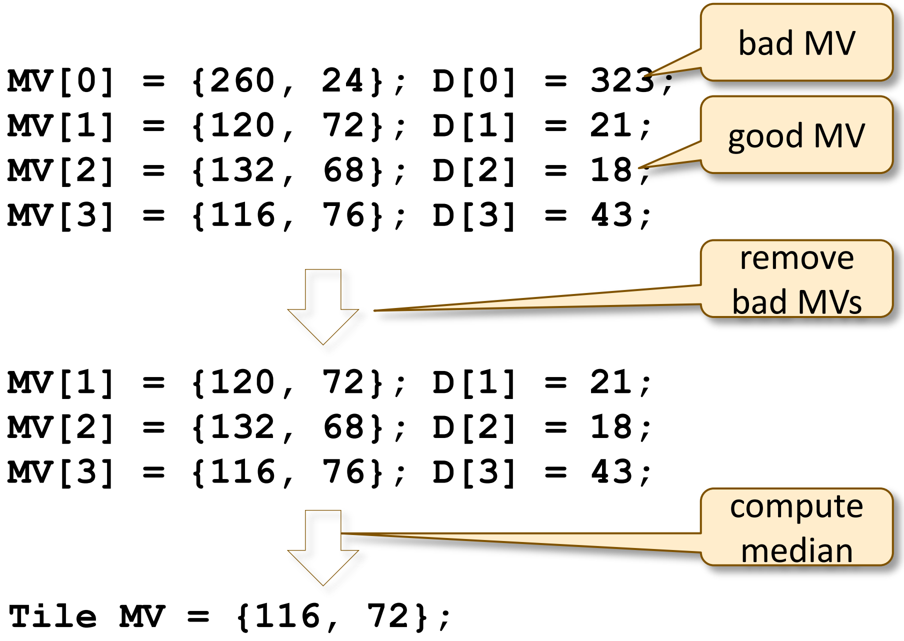
good MV

remove bad MVs

```
MV[1] = {120, 72}; D[1] = 21;
MV[2] = {132, 68}; D[2] = 18;
MV[3] = {116, 76}; D[3] = 43;
```

- we will use real life streams, so don't expect zero distortions and perfect MVs match, you have to invent algorithm to calculate tile MV
- start with selecting threshold and remove all "wrong MVs", you can use dynamic threshold or base it on results of experiments

Calculate tile MV, cont.



```
MV[0] = {260, 24}; D[0] = 323;
MV[1] = {120, 72}; D[1] = 21;
MV[2] = {132, 68}; D[2] = 18;
MV[3] = {116, 76}; D[3] = 43;
```

```
MV[1] = {120, 72}; D[1] = 21;
MV[2] = {132, 68}; D[2] = 18;
MV[3] = {116, 76}; D[3] = 43;
```

```
Tile MV = {116, 72};
```

- we will use real life streams, so don’t expect zero distortions and perfect MVs match, you have to invent algorithm to calculate tile MV
- start with selecting threshold and remove all “wrong MVs”, you can use dynamic threshold or base it on results of experiments
- from remaining vectors calculate tile MV, for example, use median
- tile MV shows you where tile has been moved by shuffler relative to the reference frame, you have to move it back

Calculate tile offset

```
MV[0] = {260, 24}; D[0] = 323;
MV[1] = {120, 72}; D[1] = 21;
MV[2] = {132, 68}; D[2] = 18;
MV[3] = {116, 76}; D[3] = 43;
```

```
MV[1] = {120, 72}; D[1] = 21;
MV[2] = {132, 68}; D[2] = 18;
MV[3] = {116, 76}; D[3] = 43;
```

```
Tile MV = {116, 72};
```

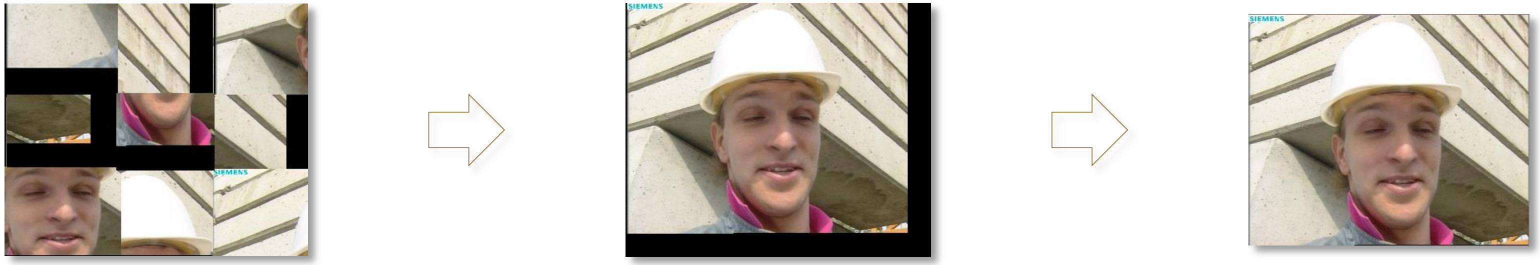
round

```
Tile Offset = {128, 64};
```

- calculate tile offset by rounding tile MV
- watch for global motion!
  no ideas how :-)
- example on the left is for integer MVs,
  PreENC outputs MVs in quarter pixel units

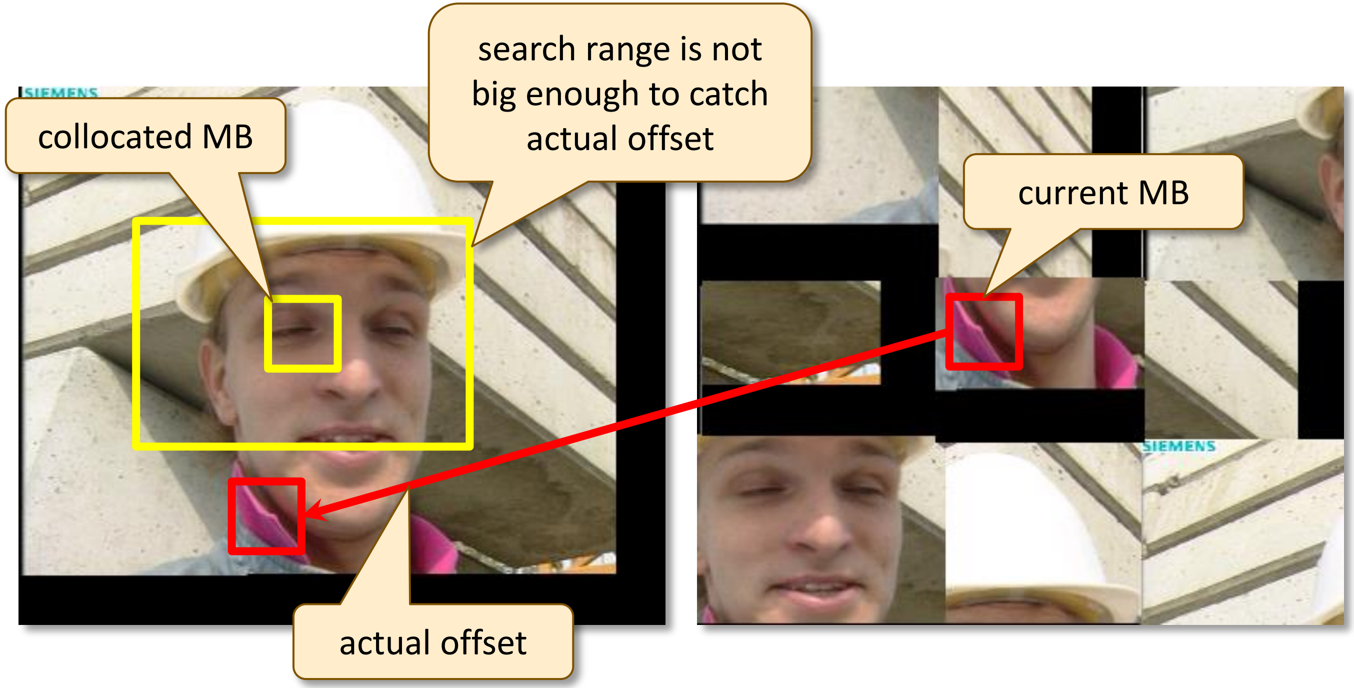
Reshuffle

- now you know original tile location, move it back to right place
- you can use VPP in composition mode
  or
  you can lock surface and move data directly in video memory
- we have no ideas what will be faster, you can misuse both methods :-)
- don’t forget to crop



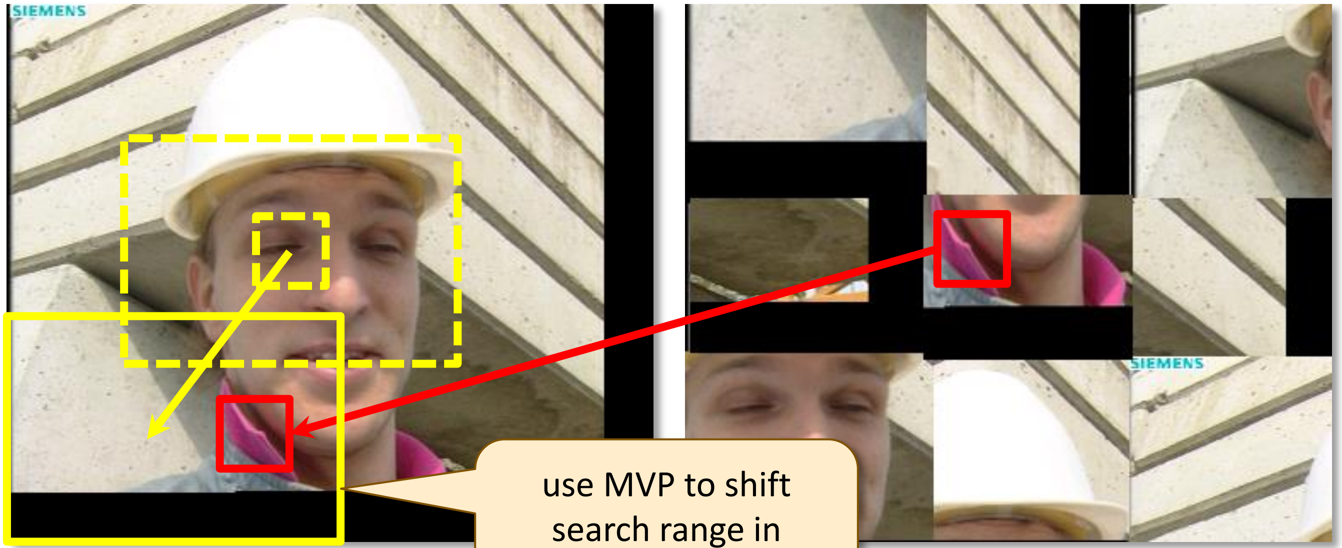
PreENC

PreENC



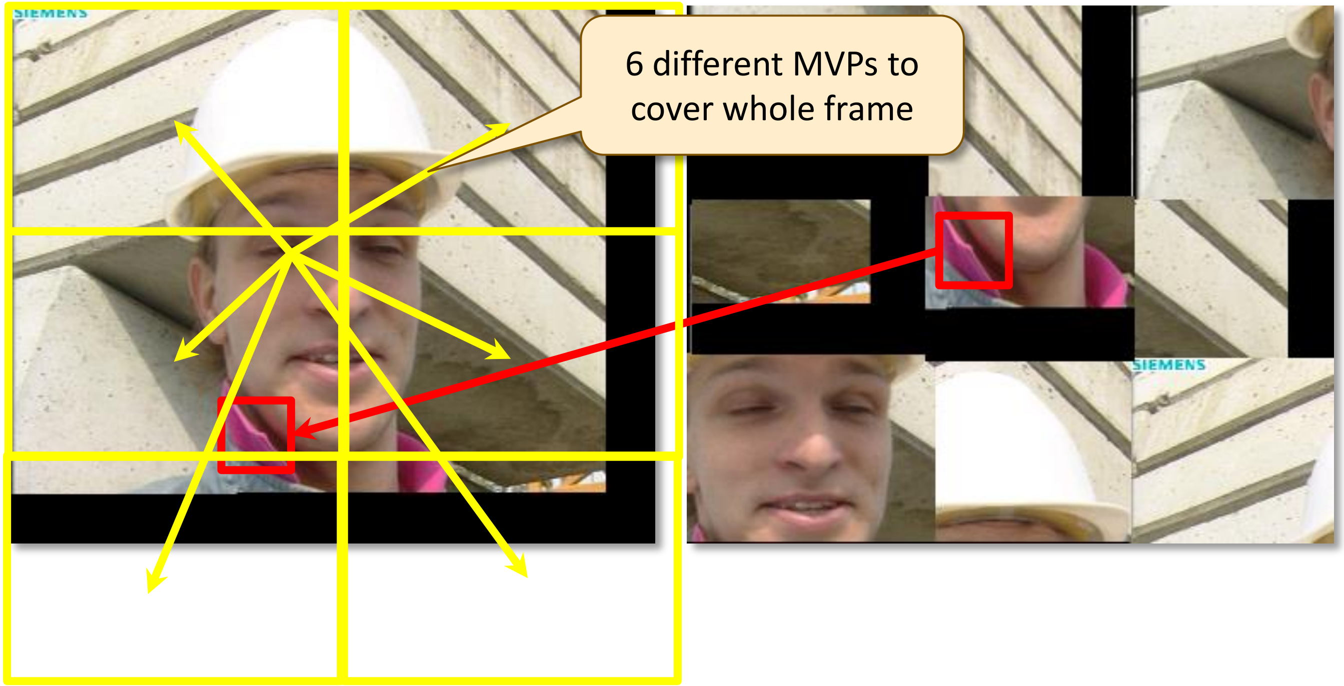
- PreENC works on 16x16 blocks
- relatively small search range +-64px horizontal +-48px vertical
- could not cover whole frame in one call

How to use PreENC with MVP



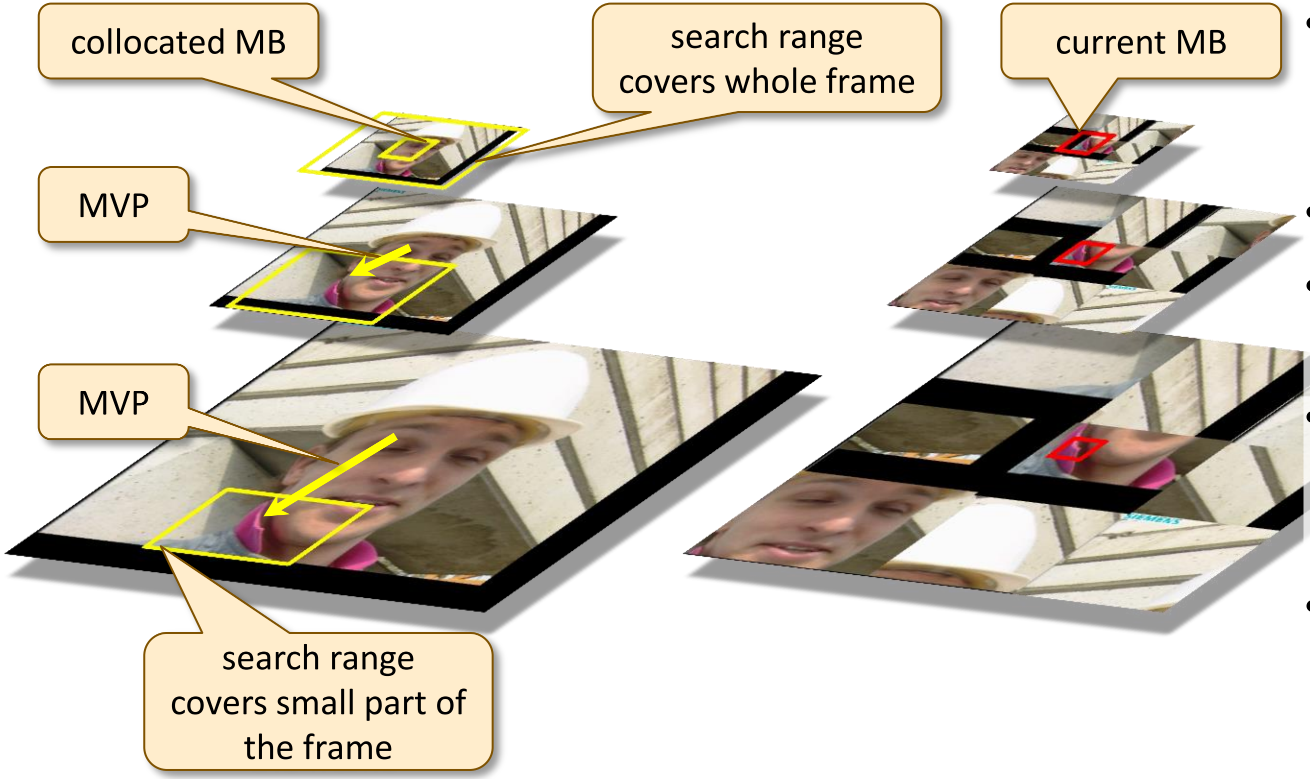
use MVP to shift search range in desired direction

How to use PreENC with MVP, cont.



- use several calls with different MVPs to cover whole reference frame

Hierarchical motion estimation


collocated MB
search range covers whole frame
current MB
MVP
MVP
search range covers small part of the frame

- create several layers, each one smaller than previous one
- do ME on top layer
- get coarse MVs, may be several for single MB
- decent one layer, use MVs from previous layer as MVP, repeat ME
- repeat for all layers

Performance

for experienced developers only

Performance optimization tips

- functionality firstly, performance later, but architect for performance from the beginning.
- keep GPU busy, try to separate frame into several partitions and process them in parallel.
- try to process several frames in parallel, it is possible but a bit tricky, guess why :-)
- you don’t need to estimate motion for all MBs in tile, in ideal case 1 is enough. You will have 16 times speed up for 64x64 tile and 64 times speed up for 128x128 tile. PreENC estimates all MBs at once but you can use different MVPs for each MB in tile.
- reshuffle – experiment(!), try in place and copy, use intrinsics, use VPP in composition mode, do it in separate thread.

Q & A