Current meta of video compression

...or how to hammer things until they work better

FOSDEM 2018
Rostislav Pehlivanov
atomnuker@gmail.com

Preamble

A few weeks ago the MPEG chairman expressed his concerns that the open model of development of AV1 is killing innovation and research.

http://blog.chiariglione.org/a-crisis-the-causes-and-a-solution/

Current state

- Motion vector search
- Qindex for the frame
- RDO for each block
- DCT
- Scalar quantization
- Loop filter
- Some form of entropy coding

Not much has changed since MPEG-1 days
Despite bandwidth and storage advances compression still as important.

The attempts of “next gen of video encoding”

Dirac:

- Used frame-level wavelet transforms instead of per-block DCTs
- OBMC
- Arithmetic coding instead of VLCs or Golomb codes
- Better resilience against errors

The result: not much better compression than H.264
Why: limitations of wavelets

The attempts of “next gen of image encoding”

JPEG2000:

- Newer
- Used frame-level wavelet transforms instead of per-block DCTs
- Arithmetic coding instead of VLCs or Golomb codes
- Better resilience against errors
- More advanced frequency management than Dirac
- Hugely more complex than JPEG

The result: worse compression than JPEG
Why: limitations of wavelets

The attempts of “next gen of video encoding”

SVQ3:

- More advanced than H.264
- Had vector quantization

The result: no major adoption
Why: bad business model, not enough coding gains over H.264

The attempts of next gen of video encoding

Daala:

- Completely different
- Lapping instead of Loop filtering
- Perfect reconstruction DCT
- Vector quantization
- Multi-symbol range coding instead of arithmetic coding

The result: surpassed HEVC in terms of quality, abandoned in favour of porting to AV1
Why: no support or help from any company
Outcome: 2 major tools ported to AV1 and 1 rewritten for AV1

The attempts of “next gen of video encoding”

MPEG-4:

- Hugely complex
- Massive
- Had specified various coding tools which couldn’t be used in any way

The result: few vendors if any implemented it, made kind of popular through open source
Why: H.264 wasn’t out yet

Yet it brought something to the table - global motion.

The conflicts when developing a new codec

- Hardware complexity
- Software complexity
- Compression

Can’t have universal adoption of a standard unless all 3 are somewhat satisfied.

Eventually a part of the process will bottleneck compression.

Replacing parts

Eventually a part of the basis of the compression process will bottleneck coding gains and will need to be replaced.

- Can’t change method significantly because hardware complexity will increase.
- Must change method significantly to resolve the bottleneck.
- Must know what to change the method to from somewhere.

Conclusion

There’s no motivation for any involved party to cease development of new compression techniques.
If you want to find out what’s likely to be in a new codec, just look at the bin of scrapped ideas of AV1 or old ideas from other codecs.

Encore: potential Vulkan use in multimedia

Why Vulkan?   Other APIs are too high level or proprietary.

First issue with using the GPU for encoding

Uploading takes time. Vulkan gives you many ways to upload.

- Create a host-visible linear image, map it and memcpy (uses the CPU, accessing linearly tiled images is slow, so only useful for short pipelines)
- Create a device-local optimally tiled image, a host-visible buffer and map it, and use a transfer queue to upload asynchronously (still uses the GPU, but accessing the resulting image is faster).
- Create a host-visible image like 1.) but use an extension to import from **host memory.** Then copy it asynchronously to an optimally tiled image. (in theory won’t use the GPU and accessing the image will be as fast as it could be).

Second issue with using the GPU to encode

Very large number of threads. Very large number of registers. Very large registers. Impractical to use for entropy coding or any other parts full of branches.

Perfect for psychovisual optimization weight calculation (x264’s OpenCL did just that on the lookahead).

Good for MV searching, loads of research done. Unfortunately little actual code exists.

Third issue with using the GPU to encode

Memory management. Allocating can be expensive and memory is dedicated for a specific purpose and can’t be reclaimed until its freed.

For temporary or intra-pipeline memory you can lazily allocate memory.

For multiplane images you should use the new extension.

Fourth issue with using the GPU to encode

Getting the data back from the GPU.

Still multiple ways to go about it:

- Put output in a host-visible buffer and map it
- Export memory via an FD and just read it
- Do it in-place

Work in progress status

https://github.com/atomnuker/FFmpeg/tree/exp_vulkan

- Doesn’t yet use CPU-less uploading
- Able to map DRM frames to Vulkan (so it can import from KMS or Vulkan) but waiting on an extension to do something useful with them.
- Can ingest arbitrary SPIR-V and perform filtering.