Lesson learnt from WebP.

What’s next?

Pascal Massimino
skal@google.com
Plan

- lessons learnt from VP8 -> WebP codec
- research direction and experiments for “WebP v2”
- results (+demo?)
Motivation

WebP, HEIF, AVIF ...
Motivation

WebP, HEIF, AVIF ...

most recent Image codecs originate from Video codec.
Motivation

WebP, HEIF, AVIF ...

most recent *Image* codecs originate from *Video* codec.

Is it always a good choice?
Lessons learnt from VP8 -> WebP
Lessons learnt from VP8 -> WebP

Two main use-cases for image compression:

- “Capture” [device -> storage / CDN]
Lessons learnt from VP8 -> WebP

Two main use-cases for image compression:

- “Capture” [device -> storage / CDN]
- “Web consumption” [CDN -> mobile device]
Lessons learnt from VP8 -> WebP

Two main use-cases for image compression:

- "Capture" [device -> storage / CDN]
- "Web consumption" [CDN -> mobile device]
Web image format

important peculiarities
Web image format important peculiarities

- incremental decoding
- memory consumption
- small format overhead
- interleaved chunk data for early display
- efficient lossy/lossless transparency
- efficient lossless coding
- preview
- light ‘animation’ format (!= video)
- efficient in software, more than hardware
Web image format important peculiarities

- incremental decoding
- memory consumption
- small format overhead
- interleaved chunk data for early display
- efficient lossy/lossless transparency
- efficient lossless coding
- preview
- light ‘animation’ format (!= video)
- efficient in software, more than hardware
WebP v2: experimentations

Goal:

\[ v2 = \text{like v1} \ldots \]

“Web-consumption”, not “Capture”.
WebP v2: experimentations

Goal:

v2 = like v1 ...

... but ‘more’.

“Web-consumption”, not “Capture”.
WebP v2: experimentations

Goal:

v2 = like v1 ...

... but ‘more’. And speed.

“Web-consumption”, not “Capture”.
WebP v2: experimentations

Goal:

v2 = like v1 ...

... but ‘more’. And speed.

And HDR.

“Web-consumption”, not “Capture”.
WebP v2: how do we improve upon v1?

What can we do differently than AV1?
WebP v2: how do we improve upon v1?

- floating partitioning
- small-context residual coding
- non-classic residuals
- custom predictors
- CfL
- lossy/lossless alpha
- more filters
- more predictors
- interruptibility
- custom CSP transform
- ANS + adaptive multi-symbol dictionaries
- tiles
WebP v2: how do we improve upon v1?

- floating partitioning  [wip]
- small-context residual coding  [go]
- non-classic residuals  [failed so far]
- custom predictors  [failed so far]
- CfL  [go]
- lossy/lossless alpha  [go]
- more filters  [wip]
- more predictors  [failed so far]
- interruptibility  [go]
- custom CSP transform  [go]
- ANS + adaptive SIMD multi-symbol dictionaries  [go]
- tiles  [go]
WebP v2: how do we improve upon v1?

- floating partitioning [wip]
- small-context residual coding [go]
- non-classic residuals [fail]
- custom predictors [fail so far]
- CfL [go]
- lossy/lossless alpha [go]
- more filters [wip]
- more predictors
- interruptibility [go]
- custom CSP transform [go]
- ANS + adaptive multi-symbol dictionaries [go]
- tiles
classic AV1 block partitioning
floating block-partitioning
floating block-partitioning

Parsing order = lexicographic order

X-Y sorted
Buffer = 32 px-high rolling cache (max block = 32x32)
Memory = $O(32 \times \text{tile\_width})$
floating block-partitioning

Parsing order != decoding order

Strategy: try to maximize the left-sample availability
floating block-partitioning

Parsing order != decoding order

Strategy: try to maximize the left-sample availability
floating block-partitioning

Parsing order != decoding order

Strategy: try to maximize the left-sample availability
floating block-partitioning

Parsing order != decoding order

Strategy: try to maximize the left-sample availability
floating block-partitioning

Parsing order != decoding order

Strategy: try to maximize the left-sample availability
floating block-partitioning

Parsing order != decoding order

Strategy: try to maximize the left-sample availability
floating block-partitioning

Problem:

the search space is HUGE
How to do RD-Opt with this vast search space??
Floating partitioning algo

Algo for finding a partitioning of a 32x32 section:

- use variance to label 4x4 blocks with four buckets.

Variance of input 4x4 blocks:

<table>
<thead>
<tr>
<th>14.0</th>
<th>12.5</th>
<th>12.0</th>
<th>11.8</th>
<th>11.3</th>
<th>8.1</th>
<th>11.1</th>
<th>10.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.6</td>
<td>12.0</td>
<td>13.3</td>
<td>12.6</td>
<td>11.9</td>
<td>9.9</td>
<td>13.3</td>
<td>8.7</td>
</tr>
<tr>
<td>12.2</td>
<td>14.6</td>
<td>12.6</td>
<td>15.0</td>
<td>10.3</td>
<td>9.2</td>
<td>11.5</td>
<td>11.2</td>
</tr>
<tr>
<td>74.7</td>
<td>80.8</td>
<td>103.0</td>
<td>118.5</td>
<td>80.1</td>
<td>16.6</td>
<td>13.2</td>
<td>20.5</td>
</tr>
<tr>
<td>37.4</td>
<td>33.4</td>
<td>39.2</td>
<td>35.6</td>
<td>34.6</td>
<td>59.8</td>
<td>114.7</td>
<td>93.4</td>
</tr>
<tr>
<td>34.5</td>
<td>29.9</td>
<td>33.1</td>
<td>30.2</td>
<td>33.4</td>
<td>30.0</td>
<td>32.4</td>
<td>25.2</td>
</tr>
<tr>
<td>32.1</td>
<td>29.9</td>
<td>37.1</td>
<td>34.5</td>
<td>34.7</td>
<td>33.7</td>
<td>29.9</td>
<td>21.7</td>
</tr>
<tr>
<td>32.9</td>
<td>31.5</td>
<td>29.6</td>
<td>36.1</td>
<td>35.9</td>
<td>28.7</td>
<td>33.3</td>
<td>29.4</td>
</tr>
</tbody>
</table>
Floating partitioning algo

Algo for finding a partitioning:

- use variance to label 4x4 blocks with four buckets.
  
  
  | 0 0 0 0 0 0 0 0 |
  | 0 0 0 0 0 0 0 0 |
  | 0 0 0 0 0 0 0 0 |
  | 2 2 3 3 2 0 0 0 |
  | 1 1 1 1 1 2 3 2 |
  | 1 1 1 1 1 1 1 1 |
  | 1 1 1 1 1 1 1 0 |
  | 1 1 1 1 1 1 1 1 |

- lay down boxes with same labels,
- starting from the largest down to the smallest (finishing fill with 4x4 boxes).
Floating partitioning algo

Algo for finding a partitioning:

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
2 2 3 3 2 0 0 0
1 1 1 1 1 2 3 2
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1
```

Diagram of partitioning.
Floating partition algo

Algo for finding a partitioning:

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
2 2 3 3 2 0 0 0
1 1 1 1 1 2 3 2
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1
```
Floating partitioning algo

- Variance isn’t necessary a good metric
- too many ‘small’ blocks for filling gaps
- so many other algos to try!
Floating partitioning algo

-> Still a lot of potential

trading geometry vs residuals
Residual coding
Residual coding

Bounds: use Adaptive Bit to say if the residuals are bounded in X/Y. If bounded, store bounds as range.

Residual: parse as zigzag but skip anything that is outside the box:
Residual coding

EOB: Adaptive Bit, but only if we have already touched both sides of the bounding box.

Only 1s after When finding a 1, ABit that indicates whether all elements after are 1s.
Custom CSP transform
Custom CSP transform

Use PCA to tight-fit the color transform matrix.
Lossy-lossless alpha mix
Lossy-lossless alpha mix
Lossy-lossless alpha mix
Triangle-based preview

218 bytes.

In the header.
Triangle-based preview

ICIP 2018 Paper.

[ grid = 64 x 64, nb_colors = 5 nb_pts = 187]  

Colormap (w/ use counts):  

- (50)  
- (39)  
- (33)  
- (34)  
- (31)
WebP v2:

results so far
WebP v2: results so far. The Good.
WebP v2: results so far. The Bad.
WebP v2: results so far. The Ugly.
Syntactic decomposition

AV1
WP2

block size coding seems more efficient!

at the detriment of block header

trading geometry vs residuals!
## Enc Speed comparison

```bash
g./examples/rd_curve kodim19.png -nomt -av1 -jpeg -webp -ssim
```

<table>
<thead>
<tr>
<th># Q</th>
<th>size (bytes)</th>
<th>bpp</th>
<th>psnr (dB)</th>
<th>SSIM*</th>
<th>enc-time (sec)</th>
<th>dec-time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>5874 0.10</td>
<td>27.07</td>
<td>6.56</td>
<td>1.79</td>
<td>0.10</td>
<td>5028 0.10</td>
</tr>
<tr>
<td>12.1</td>
<td>5776 0.12</td>
<td>27.50</td>
<td>6.65</td>
<td>1.86</td>
<td>0.10</td>
<td>13826 0.27</td>
</tr>
<tr>
<td>24.3</td>
<td>6834 0.14</td>
<td>28.24</td>
<td>6.95</td>
<td>1.81</td>
<td>0.09</td>
<td>18850 0.38</td>
</tr>
<tr>
<td>36.4</td>
<td>8308 0.17</td>
<td>29.04</td>
<td>7.32</td>
<td>1.83</td>
<td>0.09</td>
<td>24882 0.51</td>
</tr>
<tr>
<td>48.6</td>
<td>11780 0.24</td>
<td>30.17</td>
<td>7.96</td>
<td>1.70</td>
<td>0.11</td>
<td>31518 0.64</td>
</tr>
<tr>
<td>60.7</td>
<td>17264 0.35</td>
<td>31.79</td>
<td>9.04</td>
<td>1.79</td>
<td>0.11</td>
<td>37818 0.77</td>
</tr>
<tr>
<td>72.9</td>
<td>28386 0.58</td>
<td>34.12</td>
<td>10.86</td>
<td>1.92</td>
<td>0.10</td>
<td>44738 0.91</td>
</tr>
<tr>
<td>85.0</td>
<td>65536 1.33</td>
<td>39.15</td>
<td>14.45</td>
<td>2.28</td>
<td>0.11</td>
<td>73180 1.49</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WP2</th>
<th>WebP</th>
<th>AV1</th>
<th>JPEG</th>
</tr>
</thead>
<tbody>
<tr>
<td>120x</td>
<td>3x</td>
<td>1200x</td>
<td></td>
</tr>
</tbody>
</table>

jpeg = ref
WebP v2: demo

[video]
Conclusion

Plan for 2020:

- finalize the decoding tools for experiments
- release the code base as starting point
Thanks!

Questions?
Extra material
incremental decoding

using fiber / coroutines to pass control around between codec and network.
Bitstream

Codec::Read(data) (main context)

Codec::Decode() (local context)

User (calling site)

CreateLocalContext()

Available chunk

Not yet available data

WaitForNewPacket()

New data chunk

Give execution control

Yield()

WaitForNewPacket()

return Status::Suspended;

Successful

ANSDec::ReadNextWord()

Successful

ANSDec::ReadNextWord()

Successful

ANSDec::ReadNextWord()

Blocking

ANSDec::ReadNextWord()

Output buffer

Time / CPU usage
Bitstream

Still not there

Available chunk

Still not there

User
(calling site)

Codec::Read(data)
(main context)

Codec::Decode()
(local context)

Discarded data

New data chunk

WaitForNewPacket()

return Status::Suspended;

Resume()

Yield()

User
(calling site)

Output buffer

Time / CPU usage

Was blocking, now successful
ANSDec::ReadNextWord()

Successful
ANSDec::ReadNextWord()

Blocking
ANSDec::ReadNextWord()
Bitstream

User (calling site)

Codec::Read(data) (main context)

Codec::Decode() (local context)

Available chunk

Discarded data

New data chunk

OnDecodedImage()

Output buffer

Resume()

Close()

return Status::Decoded;

Time / CPU usage
Incremental decoding

Don’t assume you have the complete data for the whole frame

one must be able to quickly suspend / resume the decoding with as few work as possible

- check points
- coroutines in the bit-reader’s TryReadNext()

Corollary: good decoding error trapping and reporting is critical
Memory consumption

Video decoding = several buffers (Ref, Alt-ref, etc.)

WebP = $O(\text{width})$ memory consumption

Blit to screen ASAP

animation = 1 buffer only
Hardware = difficult for images

Hardware decoding is:

- per-frame oriented, non-interruptible
- tricky to re-configure
- non-parallelizable
- unstable, sandboxed
- has transfer overhead
Hardware = difficult for images

WebP experiment with Android vp8 hardware:

only 50% faster, but a lot of extra system complexity

-> Let’s target software decoding!