Overview:

HTJ2K Compression WG-04

Bill Wallace, Radical Imaging

Chris Hafey, Amazon Web Services

Challenges with Existing Compression

- **JPEG 2000 has advanced features but has seen low adoption**
  - Significantly slower than other transfer syntax (at least 5x slower than JPEG-LS to decode)
  - Advanced features require image to be encoded using specific parameters (e.g. scalable resolution requires RPLC progression order, certain number of decompositions)
  - Very complex to implement correctly (several interoperability issues early on)
  - Lack of availability of open source implementations for many years (paid commercial libraries was the only option)
  - Concerns over patent conflicts (increased risk of litigation)

- **JPEG Lossless and JPEG LS**
  - Do not support scalable resolution access
  - Little to no gain from parallelization (not designed for this)

HTJ2K Compression

JPEG2000 was extended in 2019 with a new block coder that improves decode and encode speed by an order of magnitude. This variant of JPEG2000 is referred to as High Throughput JPEG2000 (HTJ2K)

- ISO/JPEG standard (https://jpeg.org/jpeg2000/htj2k.html)
- Free of patent conflicts, royalty free
- Strong Open Source support (OpenJPH, OpenJPEG, GROK, ffmpeg)
- Slightly lower compression ratios than JPEG2000 (~5%)
- Significantly faster at decoding than all compressed transfer syntaxes by at least 50% -> improved image display speed
- Supports most of the advanced JPEG2000 features (scalable resolution access in particular)
- Lower implementation complexity

Learn more: https://github.com/chafey/HTJ2KResources

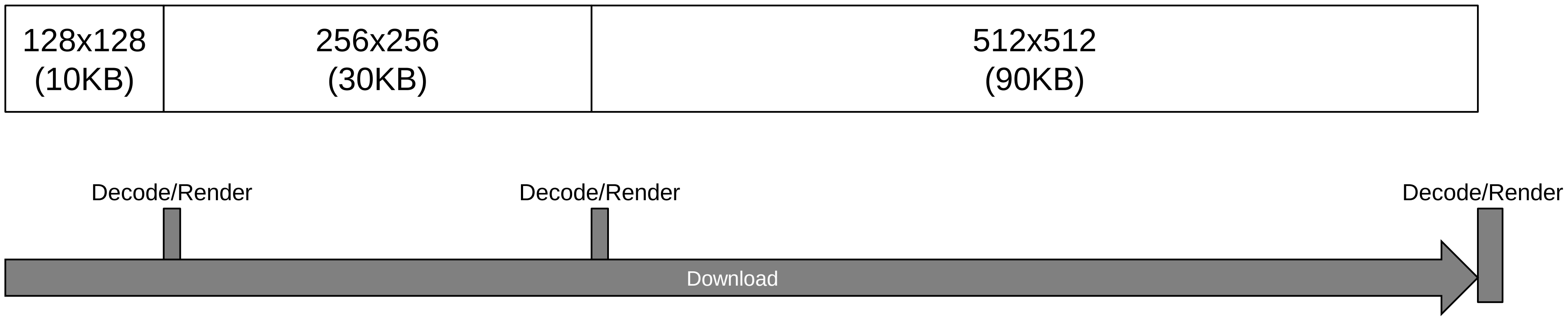
Implementation Results

- OSS Implementation in Cornerstone3D
- https://deploy-preview-779--cornerstone-3d-docs.netlify.app/docs/concepts/progressive-loading/stackProgressive

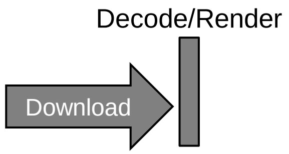
Type	Network	Size	First Render	Final Render
JLS	4g	10.6 M		4586 ms
JLS Reduced	4g	766 K	359 ms	4903 ms
HTJ2K	4g	11.1 M	66 ms	5053 ms
HTJ2K Range	4g	128 K	45 ms	4610 ms

Resolution Scalability

HTJ2K Lossless CT Image - 140KB Total



Viewer can dynamically improve the displayed image quality by decoding and rendering the image bitstream as it is received. This results in better user experiences, especially over lower and variable bandwidth network connections



Viewer can display a 128x128 thumbnail of an image (e.g. series browser) by reading a small number of bytes (10K) of the bitstream. This results in faster initial loading, less bandwidth consumption and lower implementation complexity

Try it out: https://chafey.github.io/openjphjs/test/browser/index.html
Learn more: https://docs.google.com/document/d/1hMv75h8g5a8EvlopucdhElKdu7QGF1PgpAK4J1Ahvp4/edit?usp=sharing

Changes Overview

- Add lossy and lossless Transfer Syntaxes for HTJ2K
- Add constrained lossless Transfer Syntax to ensure applications can leverage scalable resolution features
- Add JPIP transfer syntax for sub-resolution fetches
- Add HTJ2K as a DICOMweb rendered media type for faster decoding and smaller image sizes

Conclusions

- **HTJ2K brings significantly faster image decoding for all image types -> faster image access**
- **The new constrained transfer syntax ensures HTJ2K images are encoded properly for scalable resolution access -> applications can confidently use this powerful feature**