Advancements in HDR Images & Textures

Stephanie Hurlburt, *Binomial*
Richard Geldreich, Jr., *Binomial*
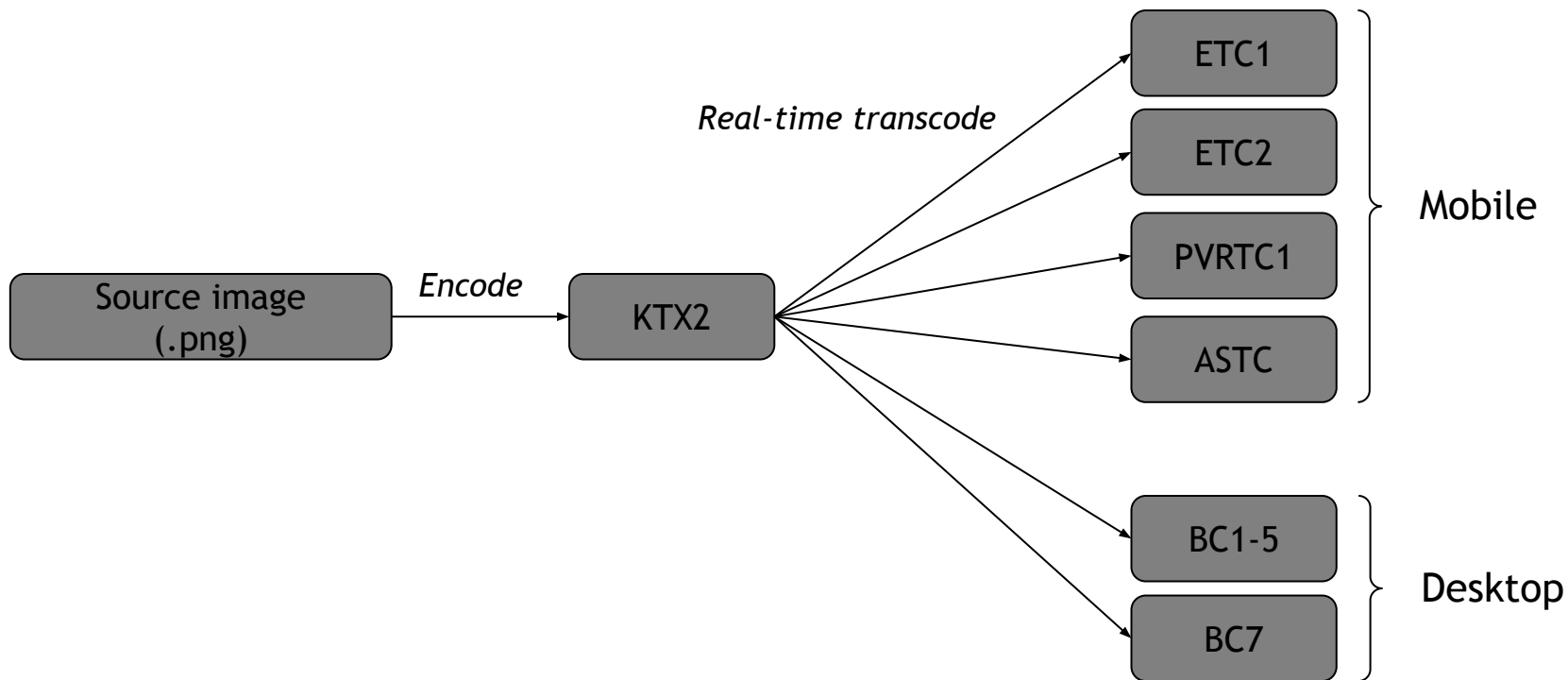
What do we do?

- We develop Basis Universal, an open source image/texture compression library
- We take advantage of GPU hardware acceleration to allow image-intensive applications to run more performantly
- When textures use compressed GPU formats, they take up a lot less space in GPU memory
- In real time, we transcode from KTX2 to the correct format that GPU hardware needs
- We contribute to the KTX2 standard so we can design the best format for everyone's needs
- We partnered with Google to develop this HDR addition
- For HDR, we're calling our spec revision "UASTC HDR", building off our high quality UASTC spec for SDR images
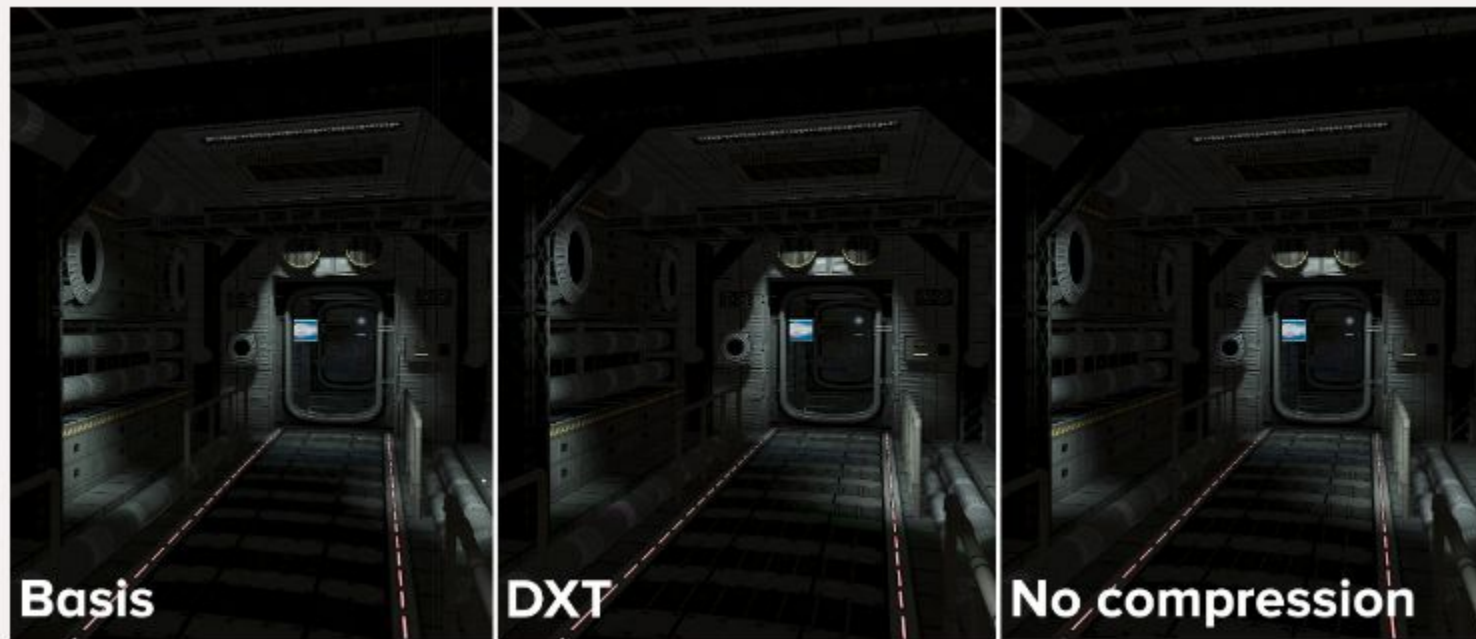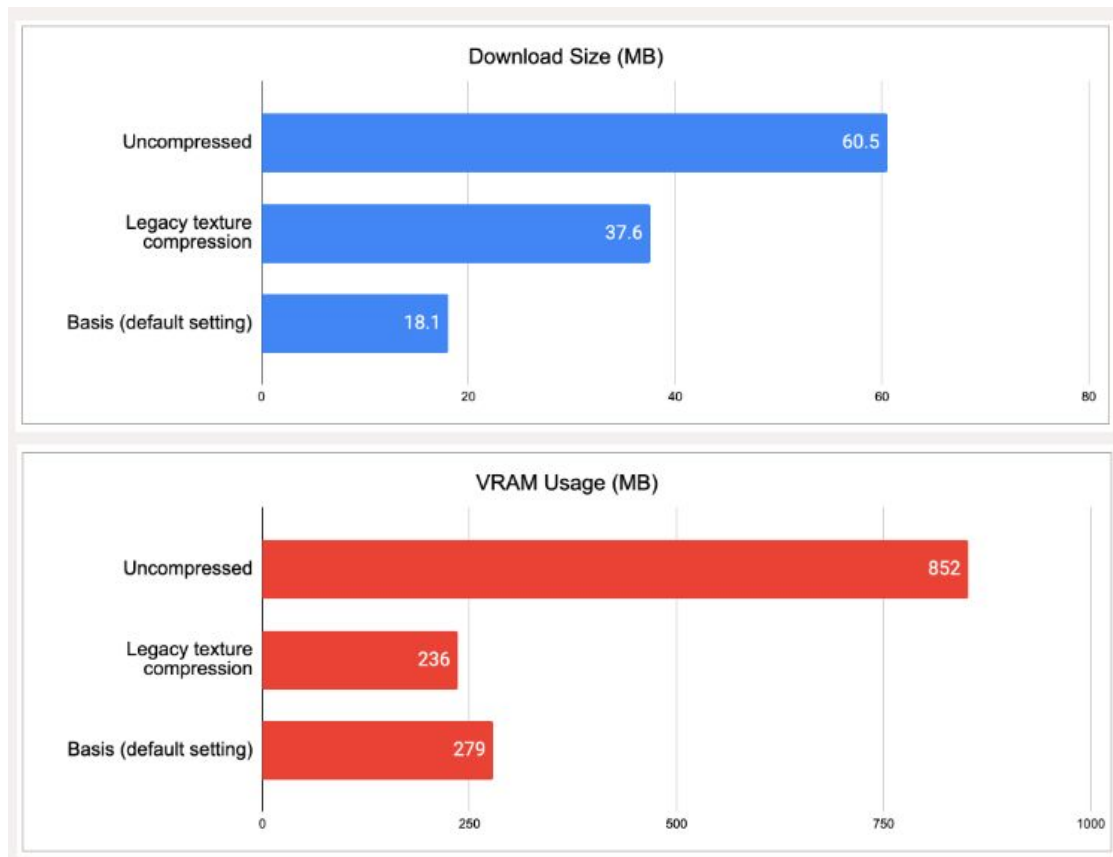
SDR Codec Examples

Credit: *https://x.com/abwood/status/1384901778173538313*

SDR Workflow

SDR Codec Examples

Credit: https://blog.playcanvas.com/basis-texture-compression-arrives-in-playcanvas/

SDR Codec Examples

Download Size (MB)

Uncompressed	60.5
Legacy texture compression	37.6
Basis (default setting)	18.1

VRAM Usage (MB)

Uncompressed	852
Legacy texture compression	236
Basis (default setting)	279

*Credit: https://blog.playcanvas.com/basis-texture-compression-arrives-in-playcanvas/*
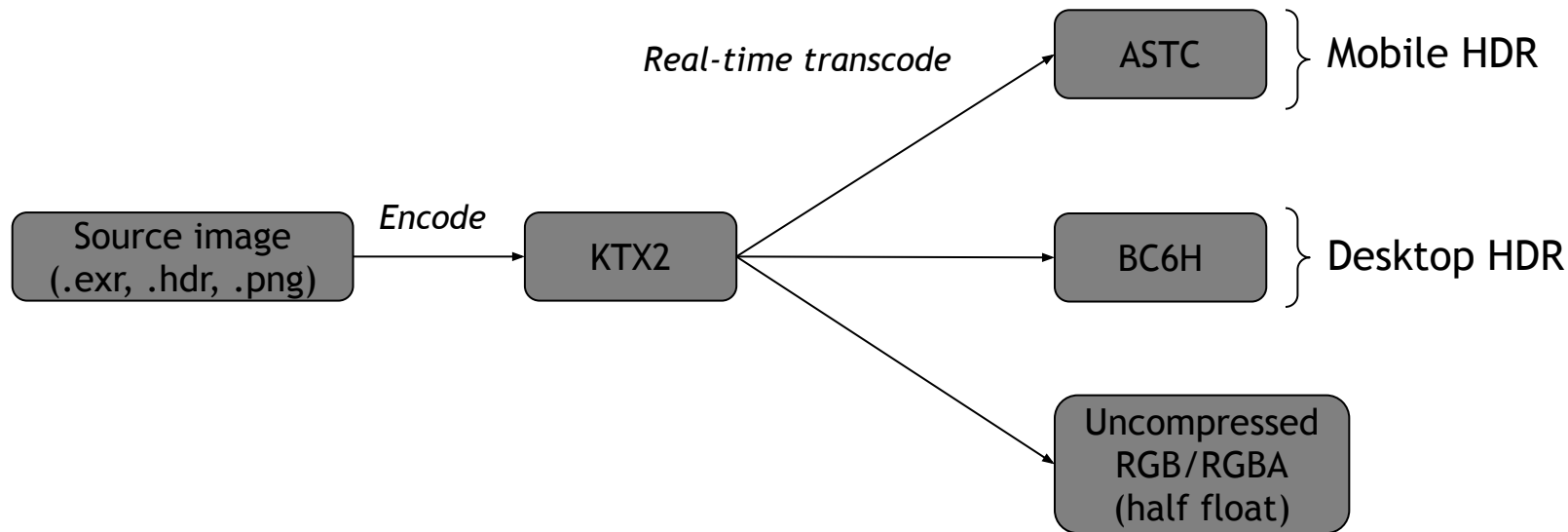
What is HDR?

- HDR stands for "high dynamic range"
- When we go into a dark room, it takes a while for our eyes to adjust. In a light room, it feels too bright to see for a second. There is such a wide range of visible light!
- The real world has all kinds of ranges of colors and light that SDR (standard dynamic range, the most common range we use now) technology doesn't pick up on
- HDR images are more realistic on HDR monitors, but even on SDR displays using them can expand graphics quality with unique photo editing and graphics engine capabilities
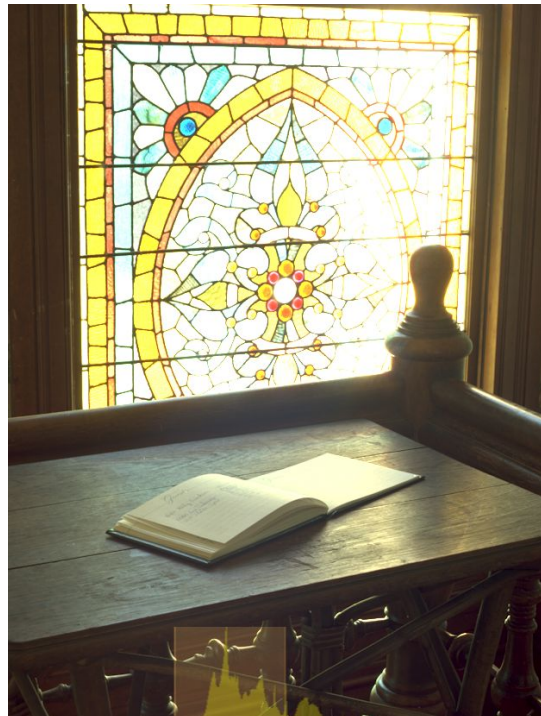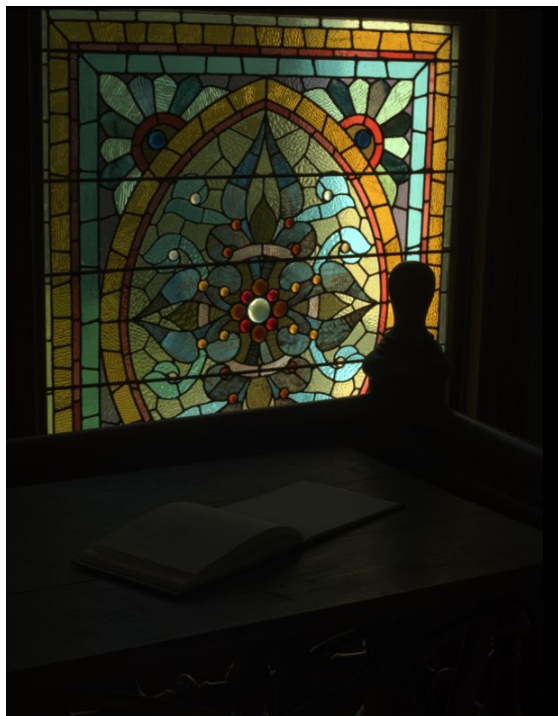
Desk.exr (https://openexr.com/en/latest/test_images/ScanLines/Desk.html) viewed on OpenHDR Viewer (https://viewer.openhdr.org/) at different exposure levels

HDR Workflow

Source image (.exr, .hdr, .png) → *Encode* → KTX2

KTX2 → *Real-time transcode* → ASTC } Mobile HDR

KTX2 → BC6H } Desktop HDR

KTX2 → Uncompressed RGB/RGBA (half float)

Overview of UASTC HDR

- 6x compression in both storage size and GPU memory (48 -> 8 bits/pixel)
- One format that works on both HDR desktop and mobile GPUs
- Format is a simple subset of ASTC
- Real time transcoding to BC6H and uncompressed, no transcode process needed for ASTC
- WASM transcoding
- Quality is almost identical to ASTC/BC6H (which are very high quality) in terms of human perception

Next we'll do a deep-dive into some of the technical details behind the project, and end with benchmarks and images of the codec
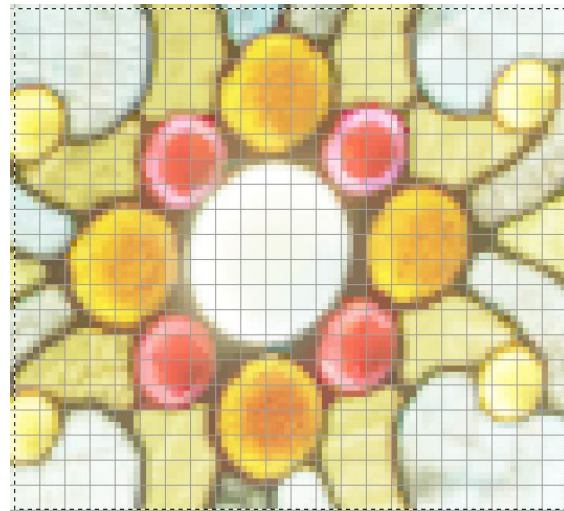
Example of minimal quality difference
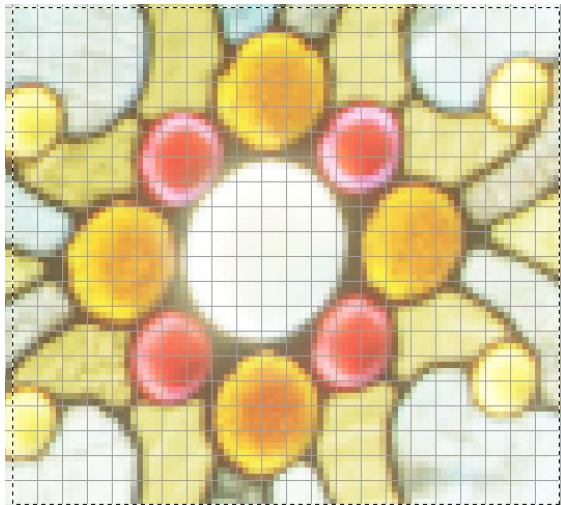
Original image
(desk: Reinhard tonemap, 1.0
intensity scale)
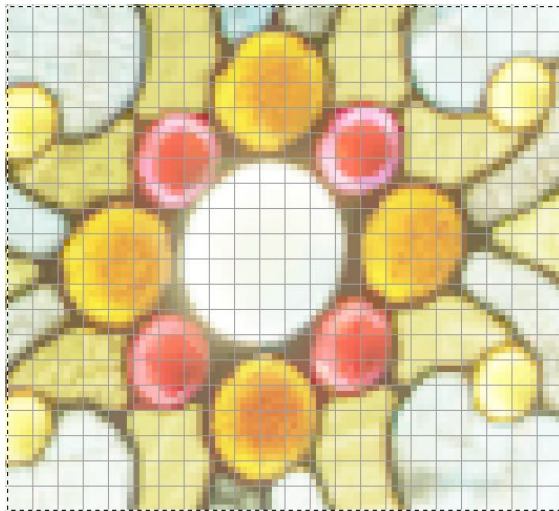
Compressed for mobile GPU
(UASTC HDR)

Compressed for desktop GPU
(UASTC HDR->BC6H)

More examples/explanation coming later!

UASTC HDR - Technical Details

UASTC HDR: Quick Background

- ASTC=**Adaptive Scalable Texture Compression**: A flexible, GPU-friendly, fixed-bitrate, fixed block size lossy texture format standard from the Khronos Group.
  - **ASTC was developed by visionary engineers at ARM and AMD**, supported by many mostly mobile GPU's.
  - Supports LDR and HDR (half float/FP16/IEEE 754-2008), 4x4-12x12 block sizes, upsampled weight grids, many other features. Notable limitation: unsigned only.
    - *See Adaptive Scalable Texture Compression, J. Nystad, A. Lassen, A. Pomianowski, S. Ellis and T. Olson*
- On desktop (and many game consoles) the simpler "good enough" **alternative from Microsoft is BC6H**: "Block Compression" 6H.
  - Always 4x4 block size, 8 bits/texel. Notably: BC6H supports both signed or unsigned FP16 texels.
- To distribute HDR texture content, **one of the common solutions is to offline encode twice** to ASTC and BC6H, using two different encoders from different authors. Two encoders=~2x as many bugs, API's, optimization effort etc. Also distribution complexities, 2x storage cost.
- An **alternative solution is to real-time encode on the target** (using SIMD or compute), requiring extra power, complexity, potential bugs due to driver interaction (drivers/shader compilers=bugs), pipeline bubbles, quality tradeoffs, encoder licensing/dev costs, SIMD porting cost, etc. (Sometimes this solution is needed.)
- The **compressed GPU texture formats are the "black sheep" of the image/texture compression approaches**. Few work in this space, but they are **crucial to exploit the power of modern GPU's**.

UASTC HDR: A New Alternative

- We encode to a common subset and feature union of both ASTC HDR and BC6H.
- "UASTC" HDR="Universal" HDR ASTC, **a standardized constrained subset** of full HDR ASTC.
- **Key feature: UASTC HDR is 100% standard ASTC texture data.**
  - **UASTC HDR data can be provided to the driver or GPU as-is because it follows the ASTC spec** (i.e. it's just ASTC HDR)
  - **Existing ASTC decoders (CPU, hardware or compute) already support it right now.**
  - **Existing ASTC encoders (CPU or compute) can be enhanced to support UASTC HDR.**
- UASTC HDR always uses 4x4 blocks and is always 8-bits pixel/texel.
  - Input: 48 bits/pixel (unsigned FP16 RGB), so **fixed 6:1 compression**.
  - Supports lossless solid color blocks ("void extent"), or lossy 1 subset, or 2 subsets with 27 possible common partition patterns (out of 32 possible for BC6H)
- **To transcode to BC6H, the ASTC block configuration is directly converted to BC6H's block configuration.**
  - **No expensive per-texel work needed, no expensive partition pattern searching, etc.**
  - UASTC HDR's RGB endpoint pairs (1 or 2 per block) are directly converted to BC6H's RGB endpoints.
  - UASTC HDR's 4x4 texel weights are directly translated to BC6H's - no searching, just simple weight index remapping.
  - **ASTC->BC6H is typically lossy**, however the error is within a fraction of a dB PSNR in 16-bit half-float space (details/graphs in a bit).

UASTC HDR: Pros & Cons

- **Pros:**
  - It's just constrained ASTC, so UASTC HDR data is still "standard".
  - We can leverage the existing ASTC spec, tooling, infrastructure, knowledge, etc.
  - Simpler than targeting full ASTC HDR, relatively easy compute shader encoding.
  - No transcoding on mobile devices (so no extra power/compute cost); mostly desktop pays the ASTC->BC6H transcode cost. (Desktop has plenty of power, mobile doesn't.)
  - Encoder can be improved over time without changing spec.
- **Cons:**
  - Lower quality than max. achievable BC6H/ASTC. (Visually it's typically imperceptible in our testing, but this is obviously subjective.)
  - No alpha: BC6H doesn't support alpha, ASTC HDR does.
  - No signed values: ASTC doesn't support signed values, BC6H does.
  - Our reference encoder is not as fast as ARM's, but we can catch up.

UASTC HDR: Risks

- **Top risk: There may be "outlier blocks" we haven't found that transcode particularly badly to BC6H.**
  - So far the worse ASTC->BC6H artifacts we've seen are not objectionable.
  - The ref. encoder tries 100's or 1,000's of block configurations and weight utilization constraints in order to maximize BC6H transcoding quality.
- **Once we have a spec the BC6H transcoding algorithm is locked in stone.**
  - If we discover higher quality ways of converting the endpoints or weights we'll need an updated spec.
  - However, there is flexibility here. **We may only suggest an algorithm**, and it can be improved/tuned over time without invalidating existing data/the spec.
- **Another risk: The ASTC HDR subset tradeoff may not be strong enough for all users.**
  - Do we add upsampled weight grids? (Extra complexity.)
  - Do we support luma-only CEM's?

UASTC HDR: ASTC Features Supported

- **"Void Extent" solid color blocks**
  - 3 half float colors (lossless to BC6H). Must be unsigned, not NaN/Inf.
- **ASTC (Color Endpoint Mode) CEM 7: 1 or 2 partitions**
  - Endpoint ISE ranges 4-20, all endpoint encoding submodes (0-5)
- **ASTC CEM 11: 1 or 2 partitions**
  - Endpoint ISE ranges 4-20, submodes 0-7 or direct
- **ISE texel weight ranges supported:**
  - 1 (3 levels), 2 (4 levels), 3 (5 levels), 4 (6 levels), 5 (8 levels), 6 (10 levels), 7 (12 levels), 8 (16 levels)
- **2 partitions: Only supports the 27 partition patterns in common between BC6H and ASTC (out of BC6H's 32 supported patterns).**
- **Weight grid must always be 4x4, i.e. no weight grid upsampling.**
- **Anything else is not UASTC HDR (i.e. no dual plane, other CEM's, etc.)**

UASTC HDR: BC6H Features Supported

- **ASTC HDR->BC6H transcoder supports all BC6H modes:**
  - Modes 11-14: 1 subset, 4-bit weights
  - Modes 1-10: 2 subsets, 3-bit weights
- **Transcoder uses the highest precision endpoint mode that can delta encode the ASTC HDR endpoints**
  - Requires a very simple search.
  - Current transcoder always falls back to trying a lower precision mode if delta encoding fails, may change to a search that minimizes overall error.
- **Weight values are translated using fixed 1D lookup tables.**
  - We're going to experiment with many different tables.
  - Some ASTC and BC6H interpolation weights slightly differ, but the encoder can try encodings that don't utilize these "more lossy" weights and choose the best overall encoding.

How we develop/test for quality

- **HDR imaging introduces many new complexities**
- **BasisU lib now accepts float images (but input must map to pos. FP16)**
- **Command line tool now accepts .EXR/.HDR files as input**
- **Main work monitor is a 1000 nits ASUS ProArt, PA27UCX-K 27", 4K HDR Mini LED, 97% DCI-P3 99.5%, Adobe RGB 100% sRGB**
  - This monitor still requires tonemapping to display most .EXR/.HDR files
  - Windows 10 HDR setup and finding proper/working HDR image viewers is very tricky
- **For debugging/devel, lib can unpack the ASTC/BC6H encodings at dozens of different exposures with a simple Reinhard tone mapper and write PNG's**
- **We also use a lossless compressive tonemapper for FP16->16-bit PNG's**
  - Tone mapping is only used for testing/development. The codec itself is full FP16 HDR.

Problem: Which HDR quality metric?

- We need a simple quality metric that works across all supported positive half-float input values and doesn't assume that the output will be tonemapped or exposed (because: it may not be, we don't know)
- ARM's standard "astcenc" encoder reports mPSNR, or "Multi-Exposure Peak-Signal-to-Noise Ratio"
    - *[MCHAM06] MUNKBERG J., CLARBERG P., HASSELGREN J., AKENINE-MÖLLER T.: High dynamic range texture compression for graphics hardware. ACM Trans. Graph. 25, 3 (July 2006), 698-706*
- This approach requires many passes through the image, each pass assumes the output is converted to 2.2 gamma (not sRGB!) and scaled/clamped to [0,255]
- Our needs are different: our encoder works internally in FP16 space (using approx. RMSLE for error), we accept FP16 input, and FP16 output.
- We need/want FP16 PSNR (for positive values).

Half-float PSNR Details

- **Jim Blinn's Corner's: Notation, Notation, Notation, page 125:**
  - "If you only deal with positive numbers, the bit pattern of a floating-point number, interpreted as an integer, gives a piecewise linear approximation to the logarithm function."
- **We interpret the FP16 bits as uint16_t and generate stock avg. RGB PSNR from that**

Updated PSNR Formula for Color Images:

When calculating the MSE for RGB images, each pixel has three components. The formula for MSE should account for the three components per pixel.

MSE Calculation for RGB Images:

$$MSE = \frac{1}{M \times N \times 3} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \sum_{c=0}^{2} [I(i,j,c) - K(i,j,c)]^2$$

Where:

- $I(i,j,c)$ is the value of the color component c (Red, Green, Blue) of the original image at pixel (i,j).
- $K(i,j,c)$ is the value of the color component c (Red, Green, Blue) of the compressed image at pixel (i,j).
- M and N are the dimensions of the images.

RMSE Calculation:

$$RMSE = \sqrt{MSE}$$

PSNR Calculation:

$$PSNR = 20 \cdot \log_{10} \left(\frac{MAX_I}{RMSE} \right)$$

Where:

- MAX_I is the maximum possible pixel value of the image (e.g., 255 for an 8-bit image).

Some viewers used for testing

- **https://viewer.openhdr.org/**
  - Web-based, supports .EXR/.HDR, exposure control, basic
- **Windows: "HDR + WCG Image Viewer" by Simon Tao**
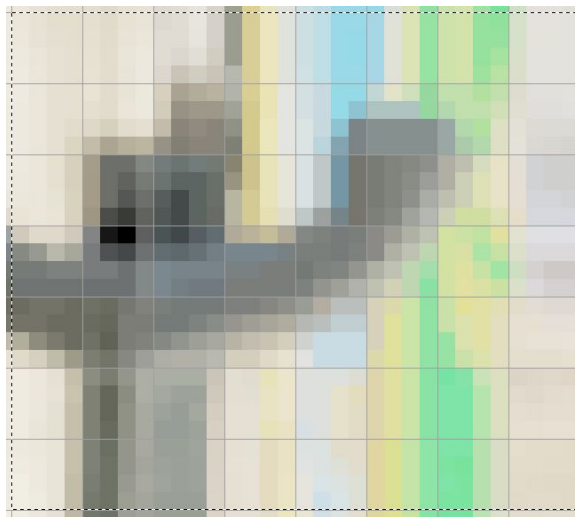 - Supports HDR mode in Windows (HDR10) with a proper config/monitor/cable
  - https://github.com/13thsymphony/HDRImageViewer
  - Windows Store: https://apps.microsoft.com/detail/9pgn3nwpbwl9?hl=en-us&gl=US

FP16 PSNR on six .EXR/.HDR photos: ARM's astcenc v4.8.0 -thorough vs. UASTC HDR vs. UASTC HDR transcoded to BC6H
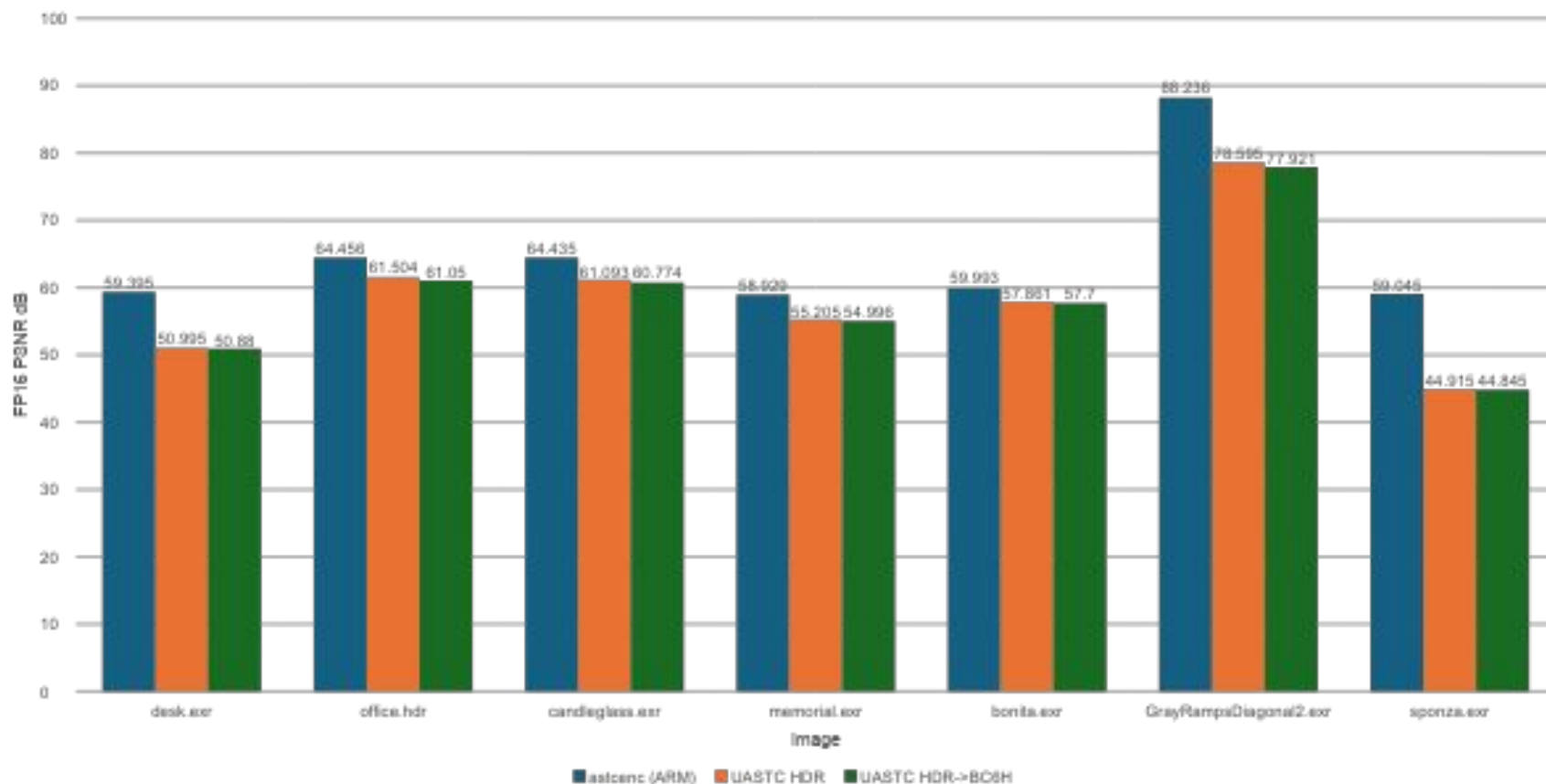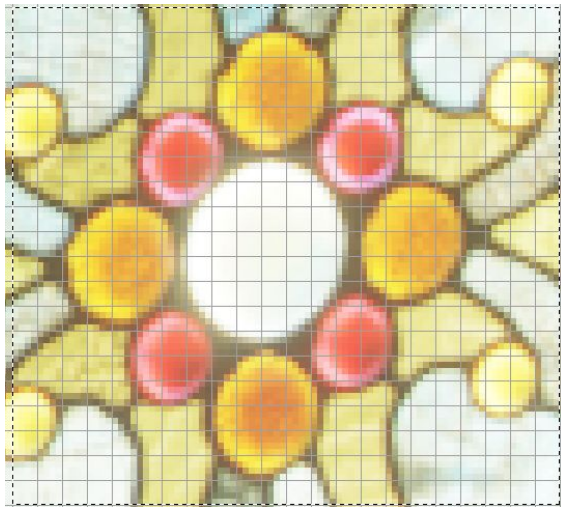
sponza: Reinhard tonemap, 2^-5 intensity scale
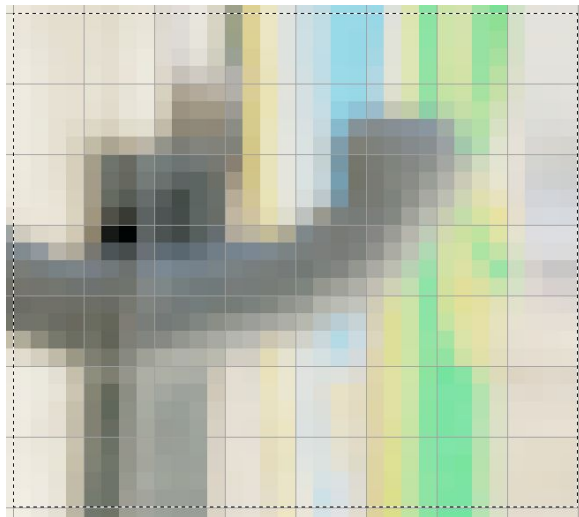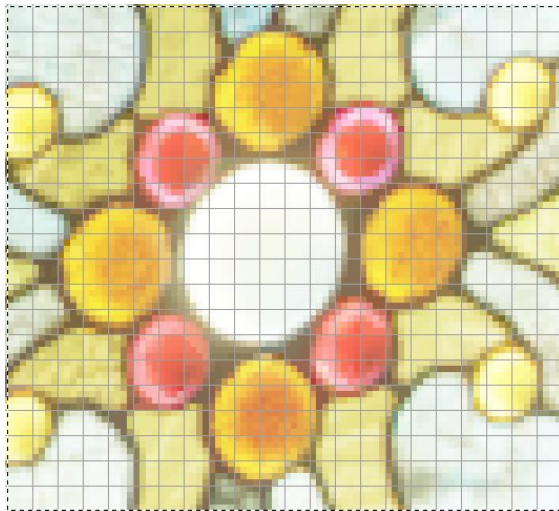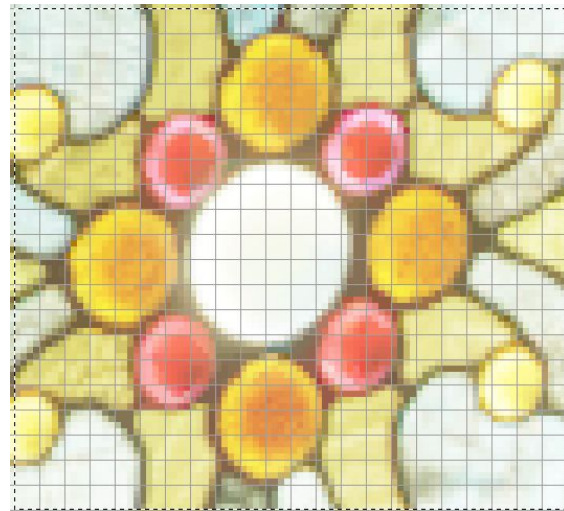

Source
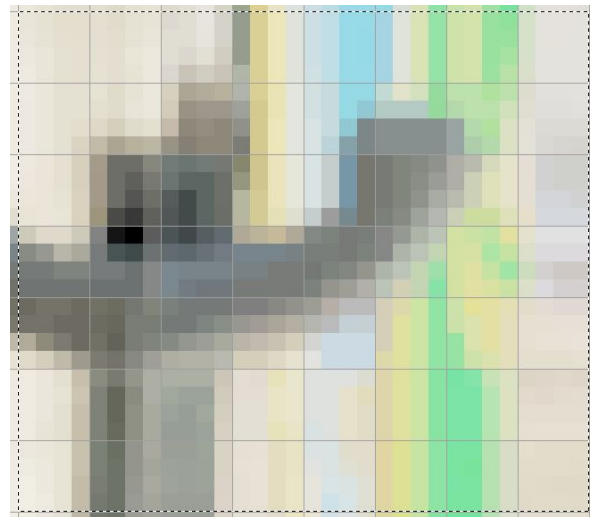
UASTC HDR

UASTC HDR->BC6H

desk: Reinhard tonemap, 1.0 intensity scale

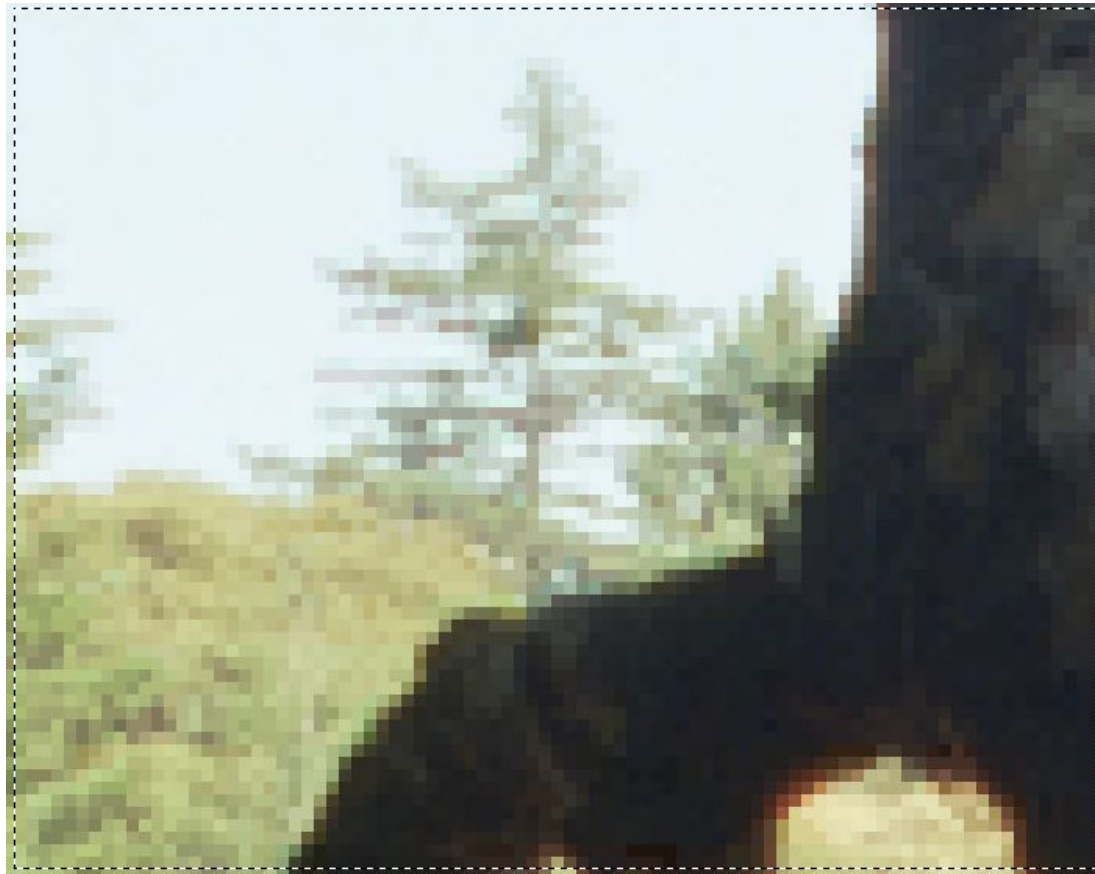Source

UASTC HDR

UASTC HDR->BC6H

Tree, source, Reinhard tonemap

Tree, UASTC HDR, Reinhard tonemap

Tree, UASTC HDR->BC6H, Reinhard tonemap

Thank you!

We are excited to introduce GPU-accelerated, cross platform HDR images to you all!

- Feel free to check out our SDR project here: https://github.com/binomialLLC/basis_universal
- Reach out to us directly: https://binomial.info/contact
- Contribute to the new HDR standard at 3DFormats Khronos meetings
- Public repo coming soon!