High Dynamic Range Rendering on the GeForce 6800

Simon Green / Cem Cebenoyan

Overview

- What is HDR?
- File formats
  - OpenEXR
- Surface formats and color spaces
- New hardware features to accelerate HDR
- Tone mapping
- HDR post-processing effects
- Problems
  - Floating-point specials!

What is HDR?

- HDR = high dynamic range
- Dynamic range is defined as the ratio of the largest value of a signal to the lowest measurable value
- Dynamic range of luminance in real-world scenes can be 100,000 : 1
- With HDR rendering, luminance and radiance (pixel intensity) are allowed to extend beyond [0..1] range
  - Nature isn’t clamped to [0..1], neither should CG
- Computations done in floating point where possible
- In lay terms:
  - Bright things can be really bright
  - Dark things can be really dark
  - And the details can be seen in both

Fiat Lux – Paul Debevec et al.

- **HDR rendering at work: Light through windows is 10,000s of times brighter than obelisks – but both are easily perceptible in the same 8-bit/component image.**

OpenEXR

- Extended range image file format developed by Industrial Light & Magic for movie production
- Supports both 32-bit and 16-bit formats
- Includes zlib and wavelet-based file compression
- OpenEXR 1.1 supports tiling, mip-maps and environment maps
- OpenEXR 16-bit format is compatible with NVIDIA fp16 (half) format
- 16-bit is s10e5 (analogous to IEEE-754)
  - Supports denorms, fp specials
  - range of 6.0e-8 to 6.5e4
- www.openexr.com

What does HDR require?

- "True" HDR requires FP *everywhere*
  - Floating-point arithmetic
  - Floating-point render targets
  - Floating-point blending
  - Floating-point textures
  - Floating-point filtering
  - Floating-point display?
- We have almost all of these today
  - With performance too

Floating-point arithmetic

- All math in the pixel shader is done in floating point today
- IEEE 32-bit (s23e8)
  - This is fast now!
- OpenEXR 16-bit (s10e5)
  - In HLSL, used with half datatype
  - Only used when _pp is specified in asm

Floating-point frame buffers

- Once you’ve done your lighting computations with HDR lights, you need to store these somewhere

- fp16 surfaces are the best solution
  - High precision
  - Linear format
  - High dynamic range

- fp32 per-component would be overkill
  - Too much space, bandwidth
  - Plus, doesn’t support blending

Floating-point Blending

- True HDR rendering was hampered in the previous generation of graphics hw by the lack of blending support – GeForce 6800 supports this
- Blending is crucial for:
  - Adding lights into the framebuffer
  - Transparency
- Many algorithms work better with one pass per light
  - Stencil shadow volumes
- Without fp blending this is painful
  - Involves ping-ponging, copying

Floating-point textures

- With GeForce 6-series we orthogonally support:
  - A32R32G32B32F
  - A16R16G16B16F
  - R32F
  - For all formats (cube maps, volume textures), power-of-2, np2
- But is this what you really want?

Floating-point textures

- Even the “low” precision texture format (4xfp16) is 64-bits per texel
  - 2x the space / bandwidth of 32-bit ARGB
  - 16x the space / bandwidth of DXT1 !
- Space is the biggest killer here
  - Hasn’t scaled at the same rate as computational power and puts a limit on visual complexity
- Surface textures don’t usually require the added range of floating point
  - Color textures just represent the percentage of light reflected (albedo)

Floating-point filtering

- We fully support fp16 filtering on GeForce 6800
- Many algorithms rely on post-processing effects after lighting
  - With HDR rendering, these lighting results will be in floating point
- Canonical example is glow / blur
  - Almost all blur kernels can be accelerated with native hw filtering support

Tone Mapping

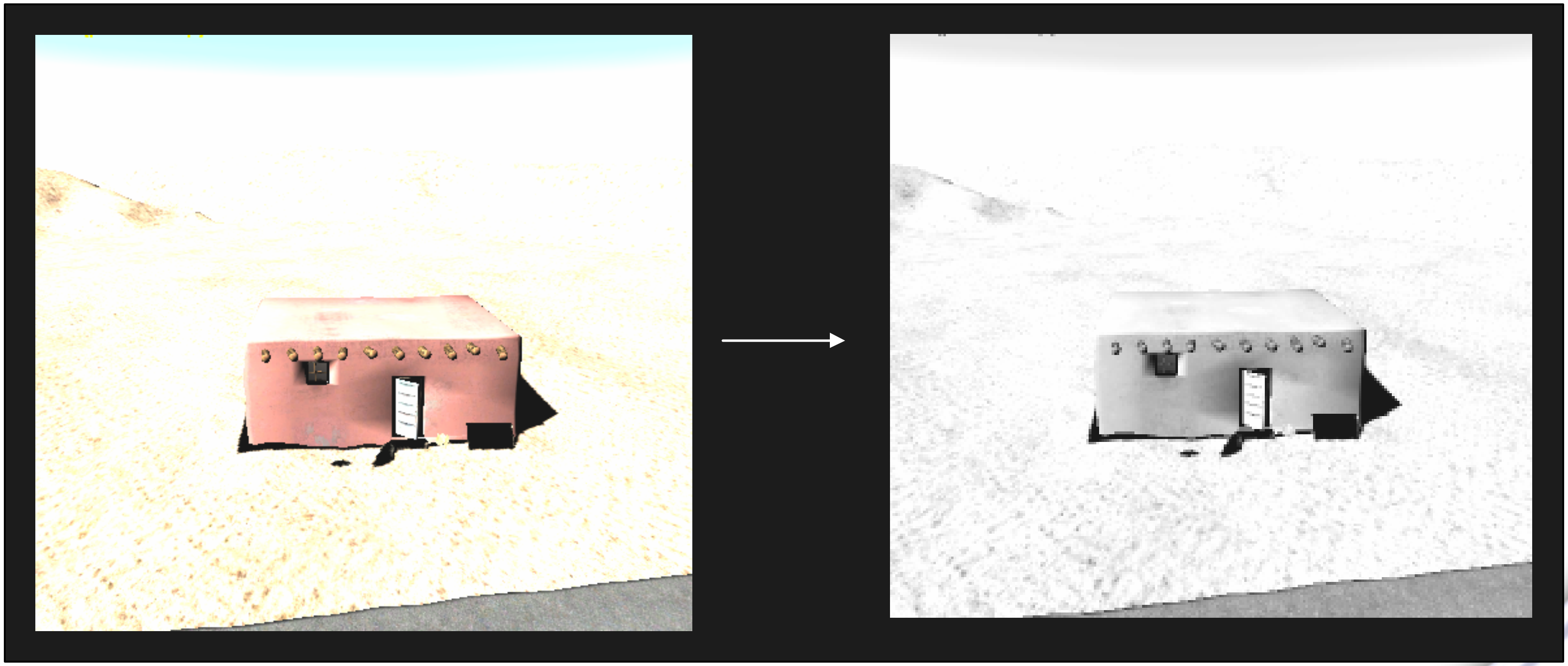
- HDR rendering produces floating point color values with unlimited range
- Most displays today are 8-bits per color component
- Tone mapping is the process of converting fp luminance values to a final displayable value
  - Analogy to film photography: set aperture, exposure based on scene, develop film
- One such mapping function is

$$Lum_{scaled}(x,y) = \frac{\alpha}{Lum_{avg}} Lum(x,y)$$

  - From “Photographic Tone Reproduction for Digital Images”, Reinhard et al.
- Note the reliance on Lum_{avg}!

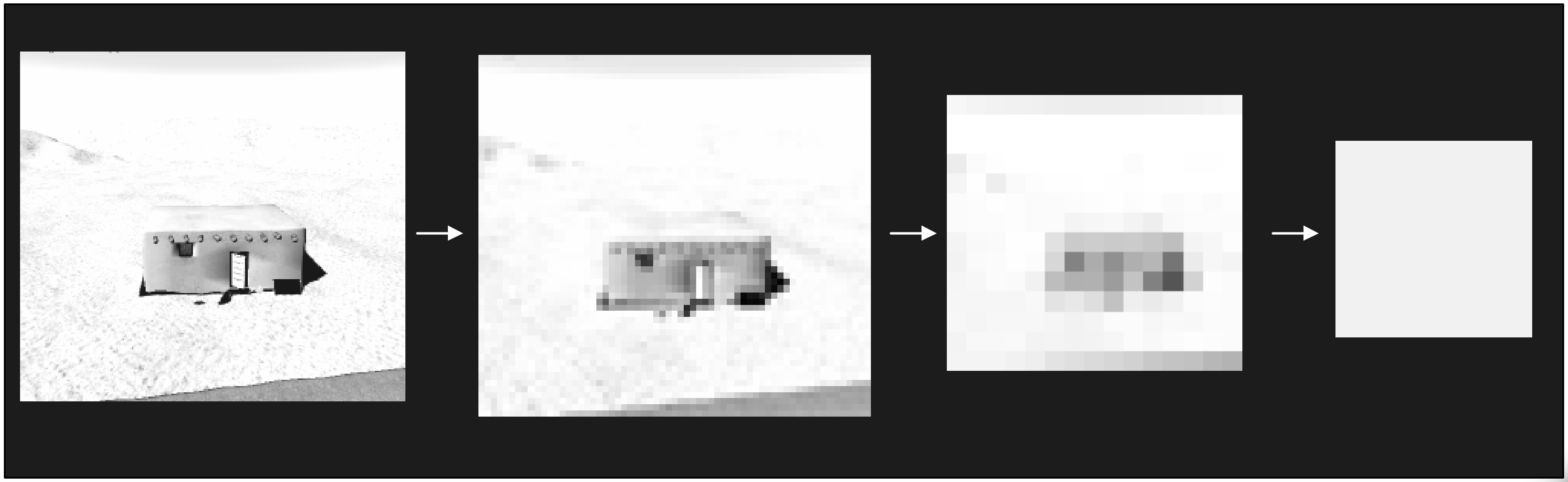
Tone Mapping

- Given an HDR scene, first convert to luminance



Tone Mapping

- Now create down-filtered results all the way down to 1x1
  - This is trivial and fast with native hardware fp filtering
  - Gives you the average luminance for the scene



HDR Post-Processing Effects

- Glow / bloom / glare
  - Very popular
  - Bright parts of scene spill over neighbouring pixels
  - Softens overall image
  - Inspired by real effect seen in film photography and in human visual system
- Implemented using blur filter
  - Render scene to texture
  - Optionally, threshold image to get bright parts
  - Blur a copy of the scene texture
  - Final image is a mix of original and blurred image

Blur Tricks

- Down sample image first
  - More efficient for large blur filters
  - Instead of using 32 pixel blur filter, reduce image by 4x and use a 8 pixel blur instead
  - Reconstruct full size image using texture filtering
  - Very hard to see visual difference
- Use separable filters
  - Blur in X axis first, and then blur in Y
  - 2n texture look-ups rather than n*n
- Use fp16 texture filtering for blur
  - Can use half the number of filter taps
  - Space taps 2 pixels apart, offset by half a pixel
  - Bilinear filtering averages each group of 2x2 pixels

Floating-point Display

- Not quite there yet
  - Not currently supported by shipping hardware
  - But coming soon!
    - http://www.cs.ubc.ca/~heidrich/Projects/HDRDisplay

Demo!

HDR Tools

- HDRshop
  - Allows viewing and editing of .HDR format images
  - Diffuse and specular environment map convolutions
  - Available from www.debevec.org
- OpenEXR
  - exrdisplay
  - Photoshop plug-in
- Greg Ward's tools
  - Photosphere (MacOS)
  - Can construct HDR images from photographs taken at multiple exposures

HDR Practicalities – FP Specials

- In debugging a number of apps, we noticed many that came out "all black" or "all white"
  - Assumed a bug somewhere in our driver
- Turns out the problem stems from implementation of floating point specials
  - NaN, +Inf, -Inf, etc.
- Some competitor's hw does not handle these like IEEE
  - So problems cropping up on GeForce

FP Specials

- Where can you get a special?
  - In the shader
  - In the framebuffer
  - From constants, vertex attributes, or a texture
  - Due to blending!

- Any time you do a calculation where a division takes place
  - For example, ray -> plane intersection, accumulating fog through a volume, often can result in divide-by-zero when ray is parallel to plane
  - Result -> Inf

FP Specials - Inf

- If you get +Inf or –Inf it will
  - Look white on screen for +Inf, black for -Inf
  - Propagate like crazy
  - Inf times any non-zero value is Inf, so convolution propagates specials
  - Inf times zero is NaN, which looks like black
  - NaN propagates even more powerfully

FP Specials – Especially Bad Case

- One extremely sneaky Inf is caused by writing out a value out of range to an fp surface
- If you write a value greater than MAX_FLOAT, you will get Inf
  - Even though it wasn't Inf in the shader!
  - MAX_FLOAT in fp16 is only around 65505, very reasonable value
  - Be careful writing out world space (x,y,z) positions to fp, since overflow can easily happen
- Clamping is the only solution for values that can go out of range
  - Adds some overhead, unfortunately

Conclusion

- HDR lighting is finally here
  - Previous hardware either wasn’t fast enough or full featured enough
- Don’t let fp specials trip you up
  - Non-obvious and difficult to debug

Questions?

- Simon Green (sgreen@nvidia.com)
- Cem Cebenoyan (cem@nvidia.com)