- In the case of the visual sense, ΔE can be specified in more detail:

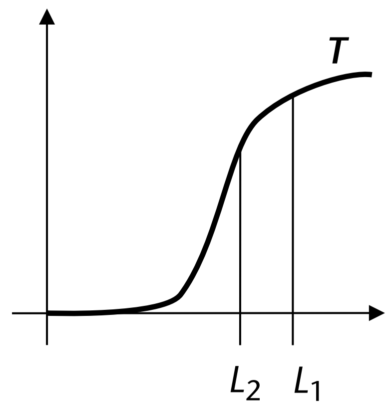
$$\Delta E = \begin{cases} -2.8 & , \log L < -3.9 \\ (0.4 \log L + 1.6)^{2.2} - 2.8 & , -3.9 \leq \log L < -1.4 \\ \log L - 0.4 & , -1.4 \leq \log L < -0.02 \\ (0.3 \log L + 0.7)^{2.7} - 0.7 & , -0.02 \leq \log L < 1.9 \\ \log L - 1.3 & , \log L \geq 1.9 \end{cases}$$

Application to Tone Mapping

- Assume two adjacent pixels in the original image have just a difference in intensity of the JND, i.e.

$$\Delta L = L_1 - L_2 = J(L_1)$$

(w.l.o.g. $L_1 > L_2$)



- Wanted is a transfer function T such that this condition is an invariant, i.e.

$$T(L_1) - T(L_2) \leq J(T(L_1))$$

- Transformation:

$$p(L_1) = T'(L_1) \approx \frac{T(L_1) - T(L_2)}{L_1 - L_2} \leq \frac{J(T(L_1))}{L_1 - L_2} = \frac{J(T(L_1))}{J(L_1)}$$

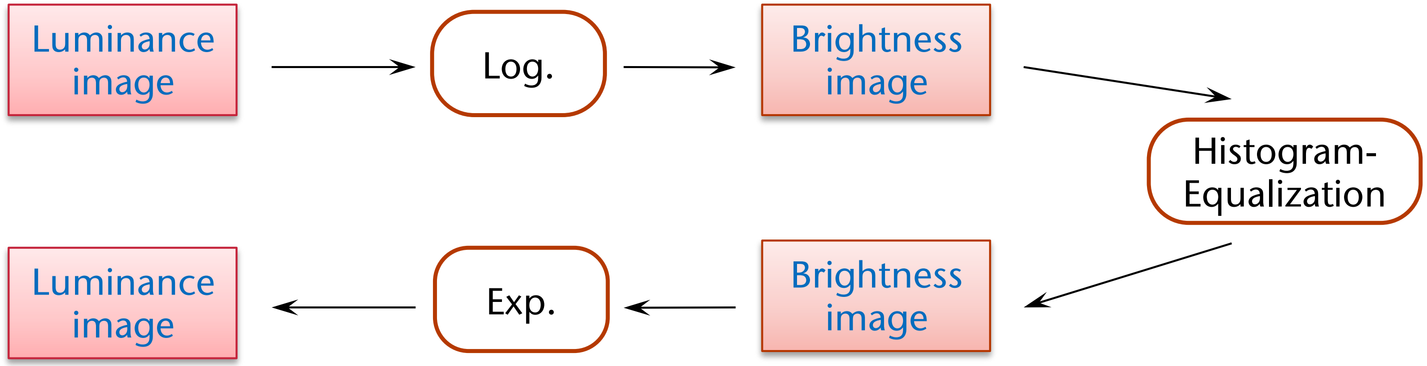
- Algorithm:
  1. Compute the histogram h
  2. Calculate the cumulative histogram $\longrightarrow$ transfer function T
  3. Clamp all bins of the original h, such that

$$h(i) \leq \frac{J(T(L_i))}{J(L_i)}$$

     where L_i is the intensity level of bin i
  4. Compute a new cumulative histogram $\longrightarrow$ new transfer function T
  5. Repeat a few times

Example

- Side note: The Weber-Fechner law is also the reason for performing the histogram equalization or tone mapping very often in so-called "log-space"



Further Ideas

- Problem: This method prevents $\Delta L > J(L)$ also between pixels, which are not adjacent
  - Idea: map each pixel taking into account *only* the neighboring pixels

    → Real local Tone-Mapping-Operator (local TMO)
  - Unfortunately leading again to other problems (i.e. "halos")
- Further limitations of the human visual systems:
  - Glare (Blendung): strong light sources in the peripheral vision reduce contrast sensitivity of the eye
  - Scotopic / mesopic vision: at low luminance, the color sensitivity decreases sharply
  - Similarly, spatial resolution decreases

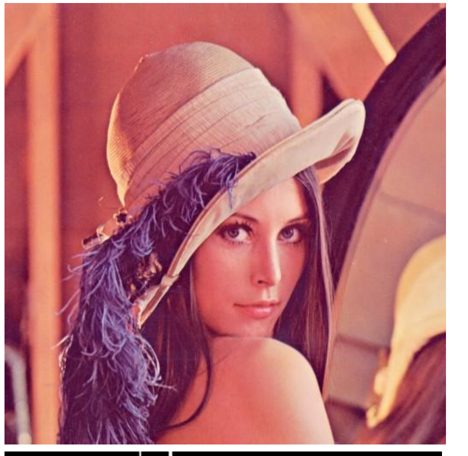
➢ Could take advantage of all that in the TMO

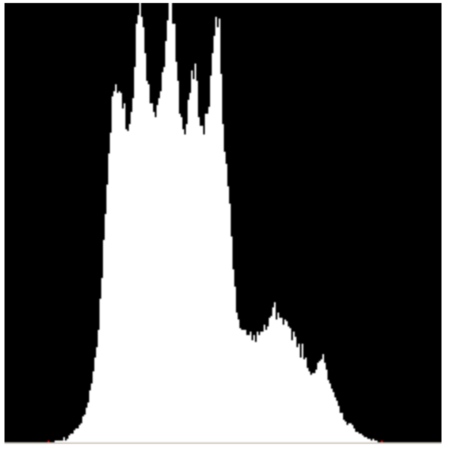
Generating a Histogram on the GPU

- Given: gray-scale image (= texture)
- Goal: histogram as 1D texture
  - Each texel = one bin
- Problem: "distribution" of pixels into the bins
  - Destination output address of a fragment shader is fixed
- First idea:
  - For each pixel in the original image, render one point (GL_POINT)
  - In the vertex shader, calculate the corresponding bin (instead of a transformation with MVP matrix)
  - Pass the "coordinate" of this bin as the coordinate of the point to the fragment shader
- Problem:
  - High data transfer volume CPU → GPU
  - Example: $1024^2 \times 2 \times 4$ Bytes = 8 MB in addition to 1024^2-image

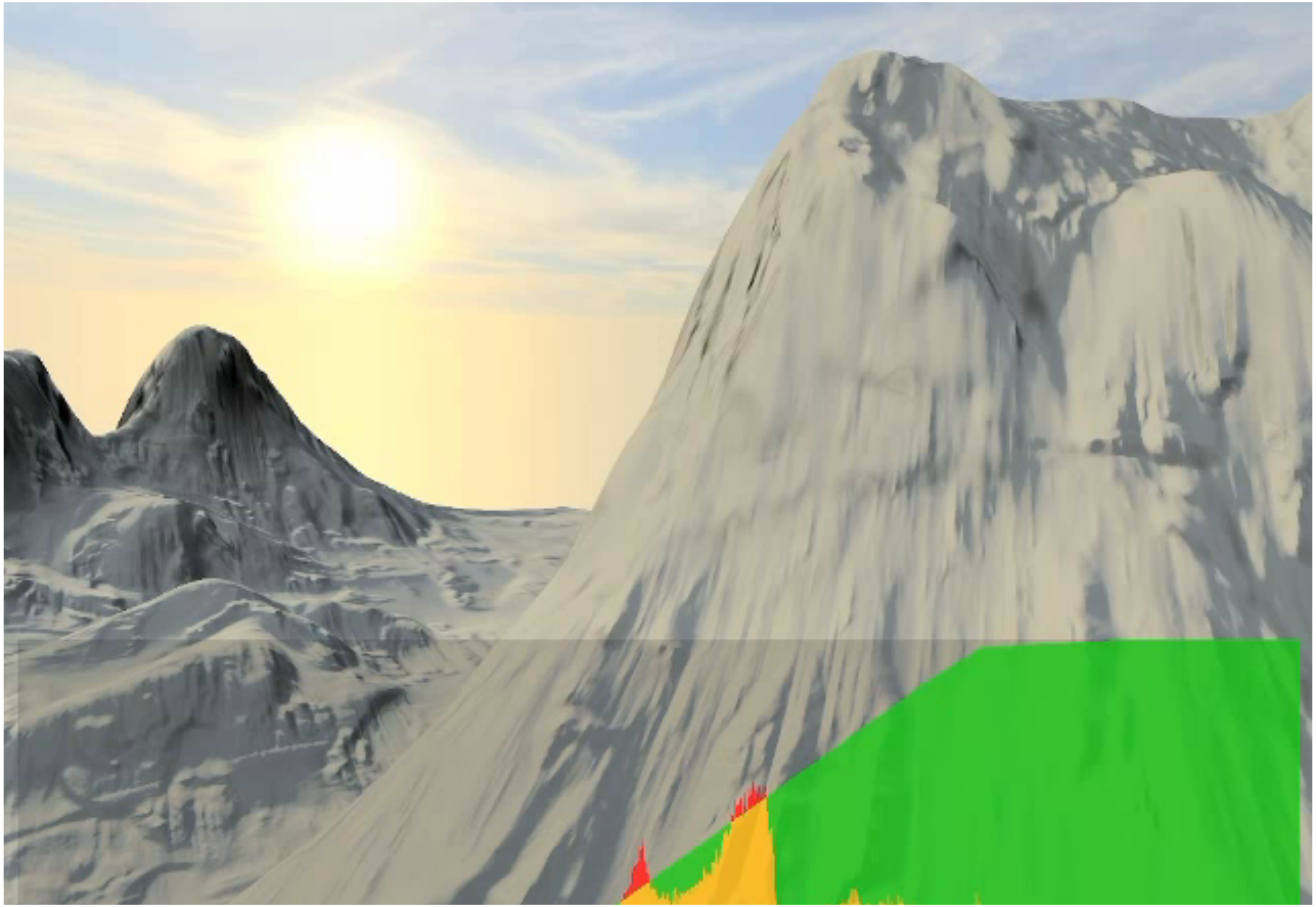
Generation of Histograms Using the Geometry Shader

- Render a quad in the application
- Vertex shader is just a pass-through
- The geometry shader ...
  - makes one loop over the image,
  - emits for each pixel a point primitive with x coordinate = bin , y=0
- The fragment shader ...
  - takes the points,
  - outputs color (1,0,0,0),
  - at position (x,0)
- The pixel operation ...
  - is set to blending with **`glBlendFunc(GL_ONE,GL_ONE)`** = accumulation (current cards can do that also with FP-FBOs)





Video



Thorsten Scheuermann, Justin Hensley; 2007.
Graphics Product Group, Advanced Micro Devices Inc.

Alternative: Use CUDA on the GPU

- Reminder for those of you who have attended my Massively Parallel Algorithms class:
  - Use CUDA's Graphics Interoperability to use image in CUDA
  - Compute the histogram using a massively parallel algorithm
  - Do a *parallel prefix sum* on the histogram
  - Switch back to OpenGL and transform the image (or do it in CUDA, too)

- For those of you who have *not* attended my Massively Parallel Algorithms class:
  - This might be an incentive to do so ☺

High-Dynamic Range Imaging in Photography

- Were there first [Charles Wyckoff, 1930-40]
- Meanwhile, HDRI is well integrated in Photoshop & Co.

Examples