Technische Hochschule Ingolstadt
Prof. Dr.-Ing. Richard Membarth

07.06.2023

GPU Programming
Assignment 7

Submission deadline for the exercises: 19.06.2023

HDR Pipeline: Light Bloom

In this assignment, we will be completing our HDR pipeline example by adding a convolution step to simulate *light bloom*, the effect of light from very bright regions in an image seemingly bleeding into adjacent darker regions.
The cause of this effect is due to the fact that any real imaging system has to work with a finite aperture. Thus, no real imaging system can produce a perfectly sharp image. The light that hits each point on the image sensor is actually the result of a convolution with an Airy Disk[1] pattern, that is, essentially a weighted average of the light coming in from a whole range of points rather than just a single point. In parts of the image that are in focus, the region around each point that has significant weight in its Airy Disk will be very small. However, even when weighed with a very small factor, an extremely bright point can still

[1] https://en.wikipedia.org/wiki/Airy_disk

have a significant effect on a very dark spot. Therefore, around areas where two regions of vastly different brightness meet, the imperfections in the optics of any real-world imaging system manifest in the generation of glaring halos around these bright regions.
To simulate this effect of light bloom, we will be using a simple Gaussian filter to blur regions of the image that are brighter than a certain threshold. We define this threshold in terms of the apparent intensity in the output image, that is, in terms of the output of our tone mapping function which we recall to be

$$\tau(v) = \frac{v \cdot (0.9036 \cdot v + 0.018)}{v \cdot (0.8748 \cdot v + 0.354) + 0.14}. \tag{1}$$

To avoid discontinuity artifacts caused by pixels barely above the threshold contributing all their light while pixels barely below the threshold contribute none, we don't use a hard cutoff but rather fade in the contribution of pixels starting at 80% of the threshold value, reaching 100% at the threshold. To avoid second-order discontinuities as a result of an abrupt change in contribution factor, we use a quadratic rather than a simple linear function to fade in the contribution. The factor to which an input exposure of v contributes to the light bloom effect is then given by:

$$\beta(v) = \left(\text{saturate}\left(\frac{\tau(v) - 0.8 \cdot \Xi}{0.2 \cdot \Xi}\right)\right)^2 \tag{2}$$

where Ξ denotes our threshold value. The function

$$\text{saturate}(x) = \min\left(\max\left(x, 0\right), 1\right) \tag{3}$$

is simply used to clamp the resulting contribution factor to the $[0, 1]$ range. For an input HDR image $I(x, y)$ at exposure e, the *bright pass* image $B(x, y)$ of each pixel's contribution to the light bloom effect can then be computed as

$$B(x, y) = \beta\left(eI(x, y)\right) \cdot I(x, y). \tag{4}$$

To obtain a blurred version of this image, we convolve it with an $N \times N$ filter kernel w_{ij}:

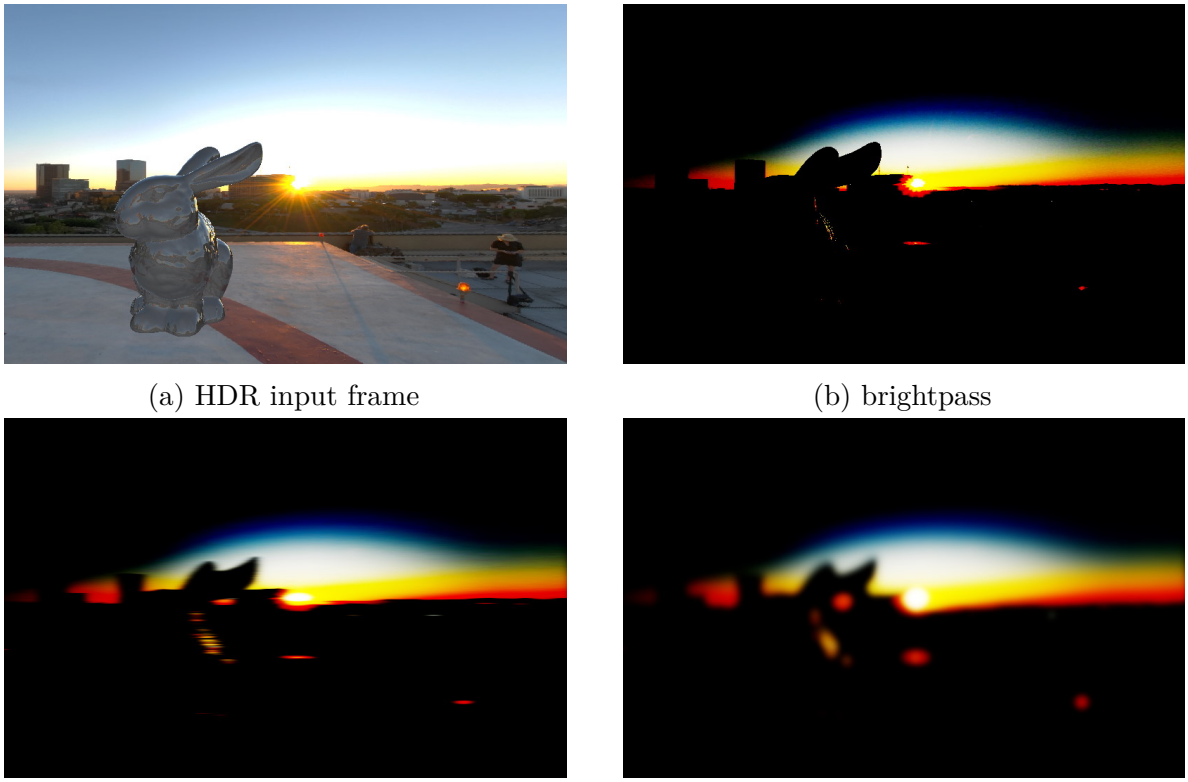
$$\overline{B}(x, y) = \sum_{j=-\frac{N}{2}}^{\frac{N}{2}} \sum_{i=-\frac{N}{2}}^{\frac{N}{2}} w_{ij} \cdot B(x + i, y + j). \tag{5}$$

For the Gaussian kernel we are going to be using, it holds that

$$w_{ij} = w_i w_j,$$

that is, the $N \times N$ matrix of 2D filter kernel weights is actually the dyadic product of a 1D Gaussian kernel of length N with itself. This gives rise of the property of *separability*:

$$\begin{aligned}
\overline{B}(x, y) &= \sum_{j=-\frac{N}{2}}^{\frac{N}{2}} \sum_{i=-\frac{N}{2}}^{\frac{N}{2}} w_i \cdot w_j \cdot B(x + i, y + j) \\
&= \sum_{j=-\frac{N}{2}}^{\frac{N}{2}} w_j \cdot \sum_{i=-\frac{N}{2}}^{\frac{N}{2}} w_i \cdot B(x + i, y + j) \\
&= \sum_{i=-\frac{N}{2}}^{\frac{N}{2}} w_i \cdot \sum_{j=-\frac{N}{2}}^{\frac{N}{2}} w_j \cdot B(x + i, y + j).
\end{aligned}$$



(a) HDR input frame (b) brightpass

(c) blurred along x (d) blurred along y

Figure 1: To simulate the effect of light bloom, we extract very bright parts of (a) the HDR input image into (b) a bright pass image. This bright pass is then blurred (c) along the x dimension and then (d) along the y dimension using a Gaussian kernel to obtain a blurred version of the bright pass to be composed on top of the input frame.

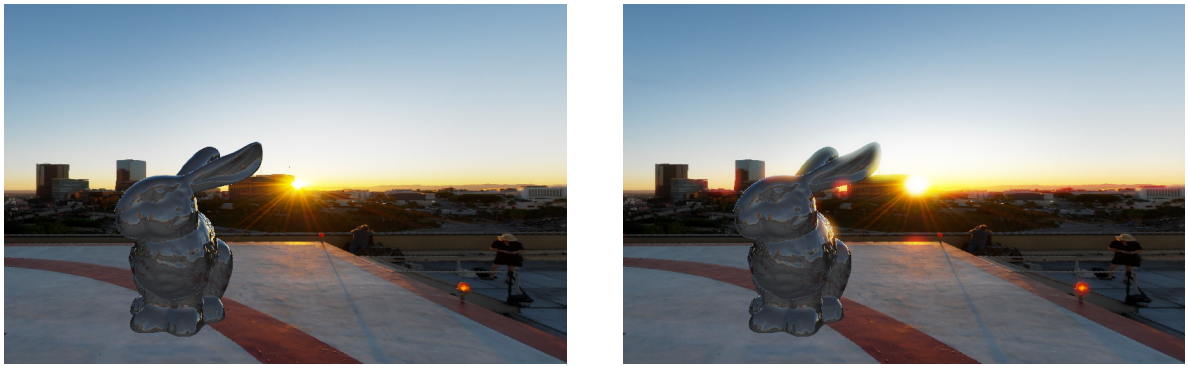
Given such a separable filter kernel, rather than computing an $N \times N$ 2D convolution for the cost of $O(N^2)$ samples to be taken per pixel, we can instead compute a sequence of two 1D convolutions of length N to obtain the same result at the cost of only $O(N)$ samples per pixel. The order in which these two convolutions are applied does not matter. For example, we could first blur the image along the x dimension:

$$\overline{B}_x(x,y) = \sum_{i=-\frac{N}{2}}^{\frac{N}{2}} w_i \cdot B(x+i,y) \tag{6}$$

and then blur the result along the y dimension

$$\overline{B}(x,y) = \sum_{j=-\frac{N}{2}}^{\frac{N}{2}} w_j \cdot \overline{B}_x(x,y+j). \tag{7}$$

Figure 1 illustrates this process of computing a blurred bright pass image.
Once we have computed our blurred bright pass image $\overline{B}(x,y)$, the last step is to compose this bright pass with the initial HDR frame before tone mapping is applied. This composition



(a) (b)

Figure 2: Comparison of the same HDR image rendered using (a) tone mapping only and (b) tone mapping with a light bloom effect applied. Note the glaring halos around the regions of bright sky bleeding into the much darker silhouette of the bunny. Further halos can be seen around some parts of the reflection of the sky in the bunny against the dark background.

is performed by adding the light bloom contributions from the blurred bright pass to the initial HDR frame. However, in order for energy to be conserved, we must make sure that pixels which did contribute to the bright pass contribute proportionally less from the initial HDR input frame. The contribution of each input pixel $I(x, y)$ to the bright pass was given by $\beta\left(eI(x, y)\right)$. Thus, the combination of input HDR frame and bright pass is obtained by

$$O(x, y) = \left(1 - \beta\left(eI(x, y)\right)\right) I(x, y) + \overline{B}(x, y). \tag{8}$$

The output LDR image is then obtained by applying tone mapping and encoding to 8-bit sRGB color like before. Figure 2 shows the difference that applying a light bloom effect can make to the result.

HDR Pipeline Task 3: Light Bloom

Extend your HDR pipeline such that a blurred bright pass is composed on top of the initial input frame to simulate the effect of light bloom using the approach described above. The threshold Ξ for the bright pass contribution in Equation (2) is given by the parameter `brightpass_threshold`. The set of $N = 63$ filter kernel coefficients to be used can be found in Appendix A as well as in the file «`bloom_kernel.h`». For the purpose of computing the convolution, assume that values $I(x, y)$ for x or y outside the image are zero.

A. Filter Coefficients

```
const float bloom_kernel[] = {
    0.000000046f, // [ 0]: -31
    0.000000108f, // [ 1]: -30
    0.000000249f, // [ 2]: -29
    0.000000559f, // [ 3]: -28
```

```
    0.000001227f, // [ 4]: -27
    0.000002629f, // [ 5]: -26
    0.000005499f, // [ 6]: -25
    0.000011221f, // [ 7]: -24
    0.000022331f, // [ 8]: -23
    0.000043323f, // [ 9]: -22
    0.000081900f, // [10]: -21
    0.000150817f, // [11]: -20
    0.000270421f, // [12]: -19
    0.000471941f, // [13]: -18
    0.000801355f, // [14]: -17
    0.001323381f, // [15]: -16
    0.002124736f, // [16]: -15
    0.003315321f, // [17]: -14
    0.005025641f, // [18]: -13
    0.007398654f, // [19]: -12
    0.010574632f, // [20]: -11
    0.014668714f, // [21]: -10
    0.019742578f, // [22]:  -9
    0.025773914f, // [23]:  -8
    0.032629535f, // [24]:  -7
    0.040049335f, // [25]:  -6
    0.047648035f, // [26]:  -5
    0.054939346f, // [27]:  -4
    0.061382872f, // [28]:  -3
    0.066448634f, // [29]:  -2
    0.069688951f, // [30]:  -1
    0.070804194f, // [31]:   0
    0.069688951f, // [32]:   1
    0.066448634f, // [33]:   2
    0.061382872f, // [34]:   3
    0.054939346f, // [35]:   4
    0.047648035f, // [36]:   5
    0.040049335f, // [37]:   6
    0.032629535f, // [38]:   7
    0.025773914f, // [39]:   8
    0.019742578f, // [40]:   9
    0.014668714f, // [41]:  10
    0.010574632f, // [42]:  11
    0.007398654f, // [43]:  12
    0.005025641f, // [44]:  13
    0.003315321f, // [45]:  14
    0.002124736f, // [46]:  15
    0.001323381f, // [47]:  16
    0.000801355f, // [48]:  17
    0.000471941f, // [49]:  18
    0.000270421f, // [50]:  19
    0.000150817f, // [51]:  20
```

```
    0.000081900f, // [52]:  21
    0.000043323f, // [53]:  22
    0.000022331f, // [54]:  23
    0.000011221f, // [55]:  24
    0.000005499f, // [56]:  25
    0.000002629f, // [57]:  26
    0.000001227f, // [58]:  27
    0.000000559f, // [59]:  28
    0.000000249f, // [60]:  29
    0.000000108f, // [61]:  30
    0.000000046f, // [62]:  31
};
```