Appendix A

This appendix provides additional analysis of our differentiable HDR synthesis layer with the radiometric calibration methods.

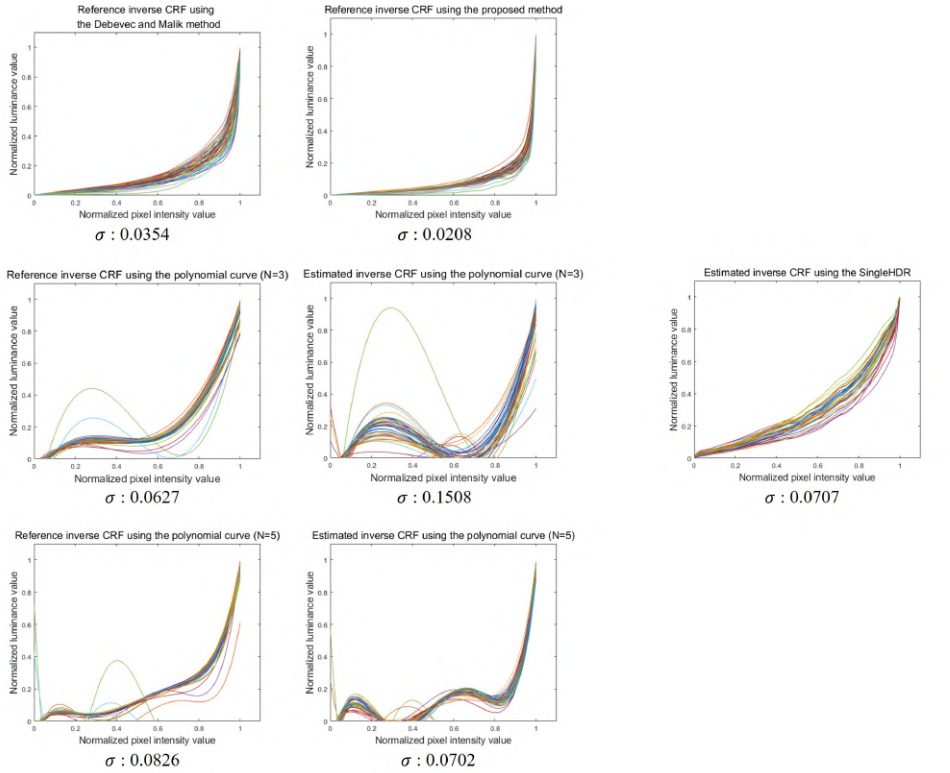

Figure 3: Comparison of the consistency of estimated inverse CRFs. N denotes the order of the polynomial curve, and the σ indicates the deviation value of inverse CRF taken from a single camera

Radiometric calibration comparison

To clarify the strength of our differentiable non-parametric radiometric calibration, we compared the radiometric calibration results between parametric and non-parametric methods. We implemented the polynomial curve fitting approach [17] as a parametric method. The method starts with using initial exposure ratios. After the iterative optimization, the coefficients of the inverse CRF and the exposure ratios are estimated by minimizing the objective function. In detail, the polynomial curve is formulated as follows:

$$\ln E = g(Z) = \sum_{k=0}^{K} c_k Z^k \tag{8}$$

where g denotes an inverse CRF, and Z denotes a pixel intensity value, E denotes a luminance value, and c_k denotes coefficients of k-th polynomial curve. Note that the Z is normalized in the range of [0,1]. The objective function to estimate the inverse CRF can be formulated as a least-square problem as follows:

$$\epsilon = \sum_{i}^{N} \sum_{j}^{P-1} \left[\sum_{k}^{K} c_k Z_{i,j}^k - R_{j,j+1} \sum_{k}^{K} c_k Z_{i,j+1}^k \right]^2 \tag{9}$$

where ϵ denotes an objective function, $Z_{i,j}$ denotes a pixel intensity value of i-th pixel with j-th exposure value. N and P is the total number of pixels and exposure values, respectively. $R_{j,j+1}$ denotes initial estimates of exposure ratio between j-th and $j+1$-th exposure values. The least-square optimization iteratively solves the objective function to optimize $R_{j,j+1}$ and the coefficients c_k, until the convergence.

With the recovered inverse CRF g, the differential coefficients can be obtained as follows :

$$\frac{\partial g}{\partial Z} = \sum_{k=1}^{N} k c_k Z^k \tag{10}$$

We conducted the comparison of estimated inverse CRFs on the VDS dataset [12], of which the images were captured with a Nikon D700 camera of 4256×2832 pixels resolutions. Since this dataset consists of images taken by the single camera, the estimated inverse CRFs from the images must have concentrated configuration. Therefore, we verified the consistency of the estimated inverse CRFs for each method. In the experiment, we have set the order of the polynomial curve to be 3 and 5. Fig. 3 shows the comparison result of different radiometric calibration approaches and SingleHDR [14], which is the state-of-the-art method. We appended ground truth results, which were generated with ground truth images with different approaches.

For the parametric approach, both ground truth and estimated results show non-consistent outputs, regardless of the order of the polynomial curve. In contrast, estimated results using our model has the least deviation even compared to the SingleHDR [14]. Furthermore, Chen *et al.*[3] analyzed that modern cameras do not have exotic structures. Based on these insights, we applied the differentiable non-parametric radiometric calibration approach. Note that we utilized the trained model as the retraining of the SingleHDR model using the VDS dataset [12] is limited.

Appendix B

This appendix describes the detailed structure of our networks and the training process.

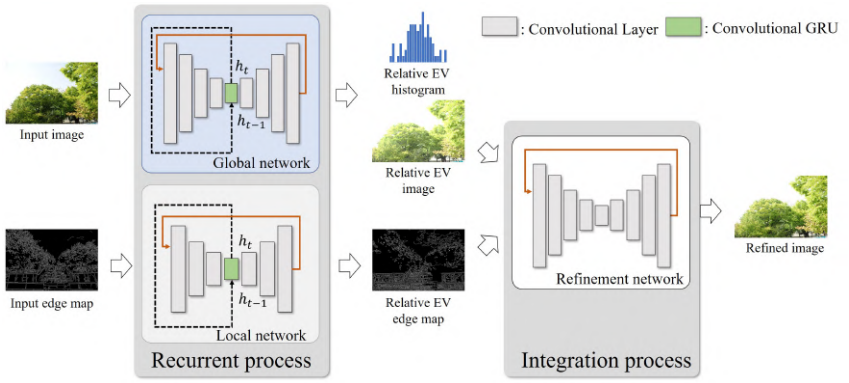

Figure 4: Sub-networks architecture. The global network focuses on minimizing the difference of histograms between the generated and target EV image, and the local network focuses on generating gradient-based edge structures. We facilitate the hidden state h_t of t-the recursion to feed into the bottleneck layers of the global and local networks for the recurrent process. We then concatenate the input image, relative EV image, and edge map to feed into the refinement network to focus on the integration process.

Our model is made up of the recurrent-up and recurrent-down networks, which share the same sub-network structures. We adopted 5-level and 4-level U-Net structure [21] to construct each sub-network as shown in Fig. 4. Each level consists of 2 convolutional layers with 2×2 average pooling layers for the encoding layers and 2×2 bilinear upsampling layers for the decoding layers. We implemented the $Tanh$ activation for the last layers of the global and refinement networks, and the sigmoid activation for that of the local network. All the other convolutional layers were followed by the Swish activation [19]. The conditional instance normalization (CIN) [6] was used in decoding layers of global and local networks. On the other hand, the instance normalization (IN) [24] was used for decoding layers of the refinement network. We further implemented the convolutional gated recurrent unit (Conv-GRU) [22] for the bottleneck layers of the global and local networks, which were preceded by the encoding layers. For the sub-networks, the global and local networks are trained in advance for 10k iterations, then we jointly trained the entire network, including the refinement network for additional 70k iterations.

Appendix C

This appendix gives extended experimental results.

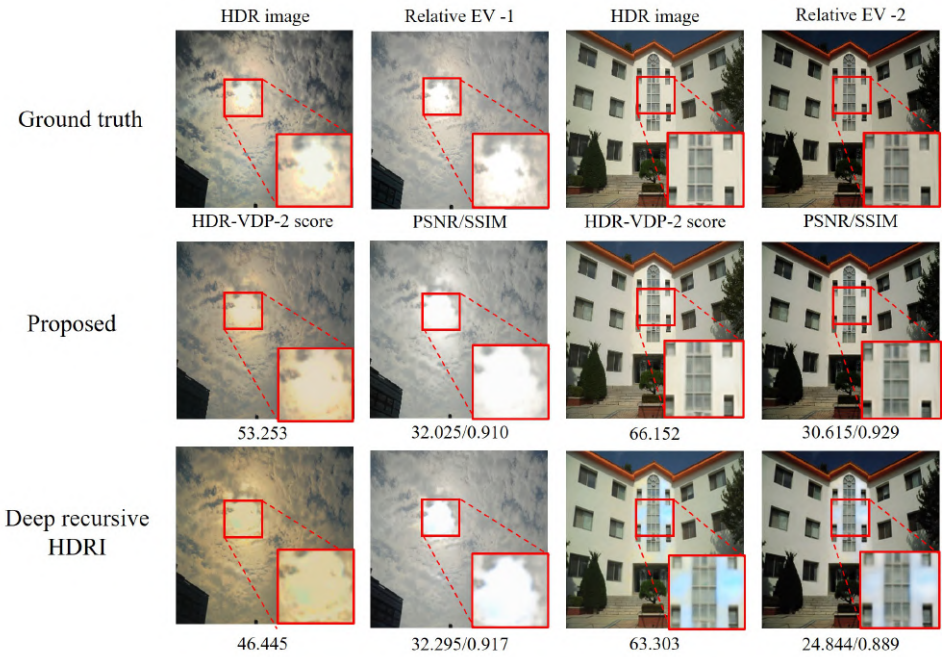

Figure 5: Case analysis of correlations between the multi-exposure stack reconstruction and the HDR reconstruction on the VDS dataset. The experiment was conducted with Lee *et al.*[13] and our method. The result shows that two factors (stack reconstruction accuracy, HDR reconstruction accuracy) have a weak correlation (suboptimal, optimal).

Table 2: Quantitative comparison of stack reconstruction results. Relative EV+1 indicates the average value of three recursive recurrent-up results and Relative EV-1 indicates the average value of three recurrent-down results.

	Method	PSNR (dB) $m \pm \sigma$	SSIM $m \pm \sigma$	MS-SSIM $m \pm \sigma$
Relative EV +1	**Proposed**	30.292±3.725	0.952±0.050	0.989±0.009
	Deep recursive HDRI [13]	30.142±2.873	0.955±0.036	0.986±0.010
Relative EV -1	**Proposed**	30.403±3.601	0.940±0.038	0.985±0.011
	Deep recursive HDRI [13]	30.483±3.836	0.936±0.044	0.982±0.014

Multi-exposure stack reconstruction

We verified the relations between the multi-exposure stack reconstruction and the HDR reconstruction. Specifically, we evaluated PSNR, SSIM, and MS-SSIM results of reconstructed stacks by our method and the previous stack-based method [13]. The previous approach [13] focused on reconstructing the multi-exposure stack, and hence, reproducing stacks with high PSNRs, SSIMs, and MS-SSIMs. However, with the results of Fig. 5 and Table 2, our method reproduced similar PSNR, SSIM, and MS-SSIM with the previous method, but achieved much higher HDR-VDP-2 scores. The results indicate that focusing on the exposure transfer task might lead to suboptimal generation performances. Furthermore, our method does not include any adversarial loss; however, as the direct relation between pixel values was imposed during the training, we achieved the result of the highest quality, thereby providing higher HDR-VDP-2 scores.

Table 3: Performance of various configurations on the VDS dataset [12]

Method	HDR-VDP-2	PSNR (dB)
Baseline	49.502±6.519	25.864±3.013
+ Recurrent network	52.344±6.852	27.652±3.189
+ Conditional instance normalization	53.020±5.110	27.996±2.779
+ Image decomposition	54.548±6.455	28.542±3.500
+ Differentiable HDR synthesis layer	57.813±5.185	29.592±3.596
+ Contextual bilateral loss	58.807±5.413	30.347±3.663

Ablation studies

We evaluated the effectiveness of the individual components in our model on the VDS dataset, as shown in Table 3. We added modules incrementally on the U-Net structure [21], which is a baseline of our model with 5-level and 2 convolutional layers for each level, and evaluated with the HDR-VDP-2 score. The overall results show that our method using all modules improved 9.305 and 4.483 with HDR-VDP-2 score and PSNR, respectively.

Recurrent network First, we added the recurrent module, the Conv-GRU [22], to be located in the bottleneck layer. We utilized the hidden state of each recurrent network to convey the important state variables, such as recursion numbers to the network. Table 3 shows that recurrent module could increase both the HDR reconstruction performance with the HDR-VDP-2 score and multi-exposure stack reconstruction with PSNR by 2.842 and 1.788, respectively.

Conditional instance normalization We demonstrated the effectiveness of the conditional instance normalization layer with a comparison experiment with the instance normalization layer [24]. We confirmed that the conditional instance normalization layer decreases the standard deviation of the reconstruction error.

Image decomposition We decomposed input images into global and local components. To verify the effectiveness of our structure, we compared the PSNR result of the decomposition network with that of the baseline network, as shown in Table 3. We trained both networks for the same iterations, and the quantitative result of PSNR shows that decomposition decreases the reconstruction error.

Differentiable HDR synthesis layer The proposed differentiable HDR synthesis layer could reconstruct the target HDR image without any learnable parameters in the layer. The mean of HDR-VDP-2 score was significantly increased by up to 3.265, and the standard deviation was decreased by up to 1.270. Hence, the differentiable HDR synthesis layer guided the network to generate a high-quality HDR image while stabilizing the training process.

Contextual bilateral loss To enhance the perceptual quality of the generated multi-exposure stack, we added contextual bilateral loss [27] to fine-tune our networks. This loss alleviated the limitations of using ghosting artifacts induced by applying L_1 loss on the misaligned image dataset. Table 3 shows that contextual bilateral loss fine-tunes the outputs of networks.

Additional qualitative results

In this section, we present qualitative results of reconstructed HDR images using the Reinhard tone mapping operator [20]. Fig. 6 shows qualitative results using the VDS dataset and HDR-eye dataset with six recent deep learning-based HDR reconstruction methods. We further evaluated comparison results with the state-of-the-art method of both direct reconstruction method (SingleHDR) and multi-exposure stack method (Deep recursive HDRI), as shown in Fig. 7. The results present that our method preserves details on the over-exposed and under-exposed region while compensating local inversion artifacts.

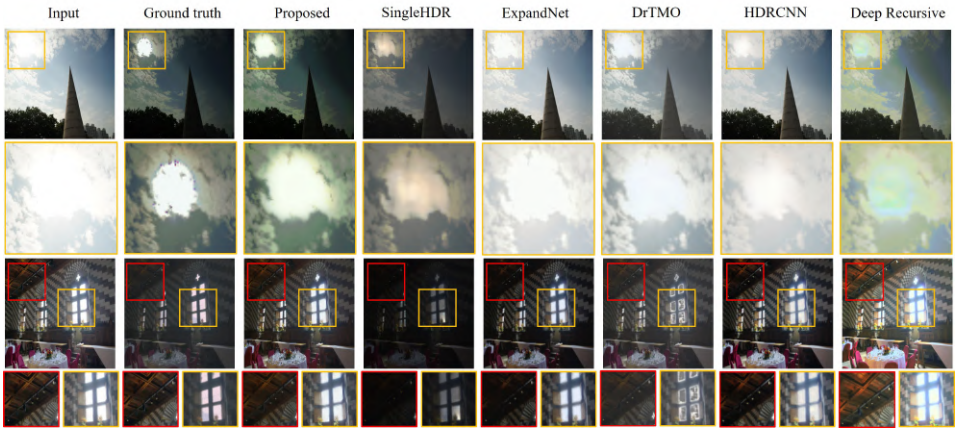

Figure 6: Comparison of tone-mapped HDR images from 6 different HDR reconstruction approaches. The loss of image details in over-exposed and under-exposed regions occurs with the SingleHDR, ExpandNet, and HDRCNN. The DrTMO and Deep recursive HDRI suffer from the local inversion artifacts. Nonetheless, our method reduces local inversion artifacts and preserves image details and contrasts in overexposed regions.

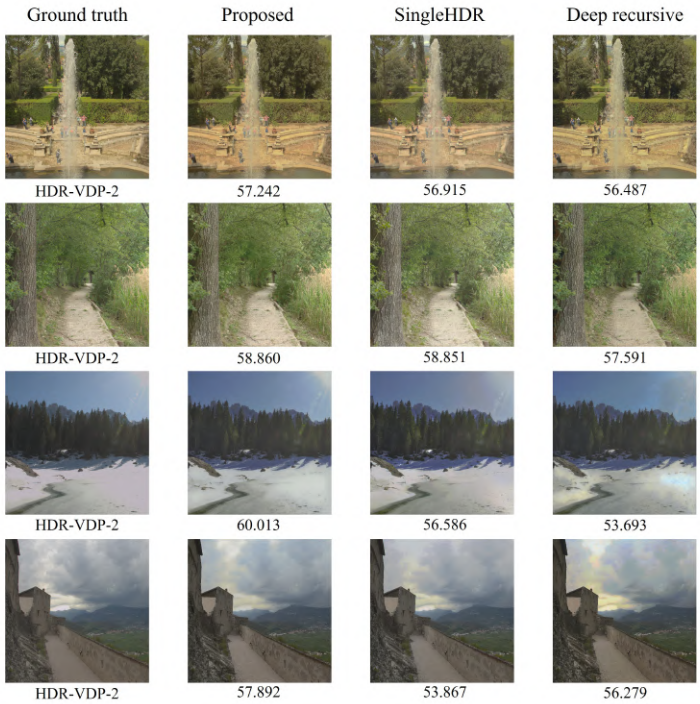

Figure 7: Qualitative results on the RAISE dataset. Scores below each image indicates corresponding HDR-VDP-2 scores.

Appendix D

In this section, we present details of our implementations with a differentiable HDR synthesis layer using both the parametric and non-parametric methods. We referenced MATLAB HDR toolbox [1] with the implementation and to generate matching results. We further provide the implementation of histogram loss used in the global network.

Autograd implementation of the differential HDR synthesis Layer

Non-parametric approach

```
import torch
from torch.autograd import Function

def tabledFunction(img, table):
    '''
    Returns remapped values regarding the table function
    Input :
        - img: an LDR image or stack with values in [0,2^nBit - 1]
        - table: three functions for remapping image pixels values
    Output:
        - img_out: a remapped image
    '''
    plf = Piecewise_Linear.apply
    img_out = torch.zeros_like(img)

    num_images, channels = img.shape[:2]
    for i in range(num_images):
        for j in range(channels):
            img_out[i,j,:,:] = plf(img[i,j,:,:], table[j,:])

    return img_out

class Piecewise_Linear(Function):
    @staticmethod
    def forward(ctx, x, w):
        '''
        Returns remapped values regarding the table function
        Input :
            - x: an LDR image / w: an tabled function [float type]
        Output:
            - result: an remapped image
        '''
        quantized_x = x.long()
        result = w[quantized_x]

        index_x = x.flatten().long()
        slope = torch.tensor(
            [index_x[0]] + [i-j for i, j in zip(w[1:], w)],
            dtype=torch.float)
        slope_x = slope[index_x].reshape(x.shape)

        ctx.save_for_backward(slope_x)
        return result

    @staticmethod
    def backward(ctx, grad_output):
        result, = ctx.saved_tensors
        return grad_output * result, None
```

Parametric approach

```
import torch
import numpy as np
from torch.autograd import Function

def PolyFunction(img, pp):
    '''
    Returns remapped values regarding the table function
    Input :
        - img: an LDR image or stack with values in [0,2^nBit - 1]
        - pp: coefficient values for the estimated polynomial function
    Output:
        - img_out: a remapped image
    '''
    poly = Polynomial.apply
    img_out = torch.zeros_like(img)

    num_images, channels = img.shape[:2]
    for i in range(num_images):
        for j in range(channels):
            img_out[i,j,:,:] = poly(img[i,j,:,:], pp[j,:])

    return img_out

class Polynomial(Function):
    @staticmethod
    def forward(ctx, x, pp):
        '''
        Returns remapped values regarding the table function
        Input :
            - x: an LDR image / pp: coefficients of the polynomial function
        Output:
            - result: an remapped image
        '''
        result = np.polyval(pp, x)
        coeff_length = pp.shape[0]

        der_poly = torch.tensor([pp[i]*i for i in range(coeff_length)]) # Differential coeffic
        slope = torch.tensor(np.polyval(der_poly, torch.linspace(0,1,256)))

        index_x = x.flatten().long()
        slope_x = slope[index_x].reshape(x.shape)

        ctx.save_for_backward(slope_x)
        return result

    @staticmethod
    def backward(ctx, grad_output):
        result, = ctx.saved_tensors
        return grad_output * result, None
```

Implementation of the histogram loss

```
import torch

def extract_histogram(x, min_value=1, max_value=255):
    '''
    Returns color histogram in the input image
    Input :
        - x: a quantized LDR image x which has values in [0, max_value]
    Output:
        - histogram: the color histogram of the input image
    '''
    assert min_value > 0
    assert type(min_value) == int
    assert type(max_value) == int

    batch_size, c, h, w = x.shape
    histogram = torch.zeros((batch_size, c, max_value+1))
    value = torch.arange(0, max_value+1).float()

    updates = x.view(batch_size, c, -1)
    indices = updates.long()
    histogram = histogram.scatter_add(-1, indices, updates)

    return histogram[..., min_value:] / value[min_value:]
```