Fast Monte Carlo Rendering via Multi-Resolution Sampling

Qiqi Hou†‡§ Zhan Li† Carl S Marshall‡ Selvakumar Panneer§ Feng Liu¶
Portland State University Intel Intel Portland State University

Abstract
Monte Carlo rendering algorithms are widely used to produce photorealistic computer graphics images. However, these algorithms need to sample a substantial amount of rays per pixel to enable proper global illumination and thus require an immense amount of computation. In this paper, we present a hybrid rendering method to speed up Monte Carlo rendering algorithms. Our method first generates two versions of a rendering: one at a low resolution with a high sample rate (LRHS) and the other at a high resolution with a low sample rate (HRLS). We then develop a deep convolutional neural network to fuse these two renderings into a high-quality image as if it were rendered at a high resolution with a high sample rate. Specifically, we formulate this fusion task as a super resolution problem that generates a high resolution rendering from a low resolution input (LRHS), assisted with the HRLS rendering. The HRLS rendering provides critical high frequency details which are difficult to recover from the LRHS for any super resolution methods. Our experiments show that our hybrid rendering algorithm is significantly faster than the state-of-the-art Monte Carlo denoising methods while rendering high-quality images when tested on both our own BCR dataset and the Gharbi dataset [14].

Index Terms: Computing methodologies—Computer graphics—Ray tracing

1 Introduction
Physically-based image synthesis has attracted considerable attention due to its wide applications in visual effects, video games, design visualization, and simulation [25]. Among them, ray tracing methods have achieved remarkable success as the most practical realistic image synthesis algorithms. For each pixel, they cast numerous rays that are bounced back from the environment to collect photons from light sources and integrate them to compute the color of that pixel. In this way, ray tracing methods are able to generate images with a very high degree of visual realism. However, obtaining visually satisfactory renderings with ray tracing algorithms often requires casting a large number of rays and thus takes a vast amount of computations. The extensive computational and memory requirements of ray tracing methods pose a challenge, especially when running these rendering algorithms on resource-constrained platforms and impede their applications that require high resolutions and refresh rates.

To speed up ray tracing, Monte Carlo rendering algorithms are used to reduce ray samples per pixel (spp) that a ray tracing method needs to cast [10]. For instance, adaptive reconstruction methods control sampling densities according to the reconstruction error estimation from existing ray samples [57]. However, when the ray sample rate is not sufficiently high, the rendering results from a Monte Carlo algorithm are often noisy. Therefore, the ray tracing results are usually post-processed to reduce the noise using algorithms like bilateral filtering and guided image filtering [27,42,44,45,51,56]. Recently, deep learning-based denoising approaches are developed to reduce the noise from Monte Carlo rendering algorithms [20,22,28]. These methods achieved high-quality results with impressive time reduction, and some of them are incorporated into commercial tools, such as VRay Renderer, Corona Renderer, and RenderMan, and open source renderers like Blender. However, real-time ray tracing is still a challenging problem, especially on devices with limited computing resources.

Our idea to speed up ray tracing is to reduce the number of pixels that we need to estimate color values. For instance, upsampling by 2 × 2 can reduce 75% of pixels that need ray tracing to estimate color. There are two main challenges in super-resolving a Monte Carlo rendering. First, it is still a fundamentally ill-posed problem to recover the high-frequency visual details that are missing from the low-resolution input. Second, a Monte Carlo rendering is subject to sampling noise, especially when it is produced at a low spp rate. Upsampling a noisy image will often amplify the noise level as well. To address these challenges, we propose to generate two versions of rendering: a low-resolution rendering but at a reasonable high spp rate (LRHS) and a high-resolution rendering but at a lower spp rate (HRLS). LRHS is less noisy while the more noisy HRLS can potentially provide high-frequency visual details that are inherently difficult to recover from the low resolution image.

Figure 1: Our fast hybrid rendering method takes a low resolution with a high sample rate rendering (LRHS) and a high resolution with a low sample rate rendering (HRLS) as inputs, and produces the high resolution high quality result.

https://github.com/hqzxyy/msspl
We accordingly develop a hybrid rendering method dedicated for images rendered by a Monte Carlo rendering algorithm. Our neural network takes both LRHS and HRLS renderings as input. We use a de-shuffle layer to downsample the HRLS rendering to make it the same size as LRHS and to reduce the computational cost. Then we concatenate the features from both LRHS and HRLS and feed them to the rest of the network to generate the high-quality high resolution rendering. Our experiments show that given the hybrid input, our method outperforms the state-of-the-art Monte Carlo rendering algorithms significantly.

To train our network, we collected a large Blender Cycles Ray-tracing dataset, which contains 2449 high-quality images rendered from 1463 models. The dataset consists of various factors that affect the Monte Carlo noise distribution, such as depth of field, motion blur, and reflections. We render the images at a range of spp rates, including 1-8, 12, 16, 32, 64, 250, 1000, and 4000 spp. All the images are rendered at the resolution of 1080p. Each image contains not only the final rendered result but also the intermediate render layers, including albedo, normal, diffuse, glossy, and so on.

This paper contributes to the research on photo-realistic image synthesis by integrating Monte Carlo rendering and image super resolution for efficient high-quality image rendering. First, we explore super resolution to reduce the number of pixels that need ray tracing. Second, we use multi-resolution sampling to both reduce noises and create visual details. Third, we develop a large ray-tracing image dataset, which will be made publicly available.

2 Related Work

Monte Carlo rendering is an important technology for photo-realistic rendering. It aims to reduce the number of rays that a ray tracing algorithm needs to cast and integrate while synthesizing a high-quality image [10][23]. Conventional Monte Carlo rendering algorithms investigate various ways to adaptively distribute ray samples [8][13][20][23][38][41][46][47]. When only a small number of rays are casted, the rendered images are often noisy. They are typically filtered using various algorithms [11][21][29][30][32][44][49]. Due to the space limit, we refer readers to a recent survey on Monte Carlo rendering [57].

Our research is more related to the recent deep learning approaches to Monte Carlo rendering denoising. Kalantari et al. trained a multilayer perceptron neural network to learn the parameters of filters before applying these filters to the noisy images [23]. Bako et al. extended this method by employing filters with spatially adaptive kernels to denoise Monte Carlo renderings [2]. They developed a convolutional neural network method to estimate spatially adaptive filter kernels. Chaitanya et al. developed an encoder-decoder network with recurrent connections to denoise a Monte Carlo image sequence [9]. Recently, Kuznetsov et al. developed a deep convolutional neural network approach that combines adaptive sampling and image denoising to optimize the rendering performance. Different from the above methods, Gharbi et al. argued that splatting samples to relevant pixels is more effective than gathering relevant samples for each pixel for denoising. Accordingly, they developed a novel kernel-splatting architecture that estimates the splatting kernel for each sample, which was shown particularly effective when only a small number of samples were used [14]. Compared to these methods, our method improves the speed of Monte Carlo rendering by reducing the number of pixels that we need to cast rays for.

Our work also builds upon the success of deep image super resolution methods [1][12][19][26][33][35][50][52][55]. Dong et al. developed the first deep learning approach to image super resolution [12]. They designed a three-layer fully convolutional neural network and showed that a neural network could be trained end to end for super resolution. Since that, a variety of neural network architectures, such as residual network [16], densely connected network [19], and squeeze-and-excitation network [17], are introduced to the task of image super resolution. For instance, Kim et al. developed a deep neural network that employs residual architectures and obtained promising results [26]. Lim et al. further improved super resolution results by removing batch norm layers and increasing the depth of networks [34]. Zhang et al. developed a residual densely connected network that is able to explore intermediate features via local dense connections for better image super resolution [54]. Zhang et al. recently reported that a channel-wise attention network which is able to learn attention as guidance to model channel-wise features could more effectively super resolve a low resolution image [53]. While these image super resolution methods achieved promising results, recovering visual details that do not exist in the input image is necessarily an ill-posed problem. Our method addresses this fundamentally challenging problem by leveraging a high-resolution image but rendered at a low ray sample rate. Such an auxiliary rendering can be quickly rendered and yet provide visual details that do not exist in the low resolution input rendered at a high sample rate.

3 The Blender Cycles Ray-tracing Dataset

We develop a Blender Cycles Ray-tracing dataset (BCR) that consists of a large number of high quality scenes together with the ray-tracing images and the intermediate rendering layers. We will share BCR with our community.

3.1 Source Scenes

Blender’s Cycles is a popular ray tracing engine that is capable of high-quality production rendering. It has an open and active community where thousands of artists share their work. Using the Blender community assets, we collected over 8000 scenes under Creative Commons Licenses, which allow us to share our dataset with the research community. We rendered these scenes at 4000 spp and manually checked the rendered images and all the rendering
layers. We eliminated scenes with missing materials, lack of high frequency information, or with noticeable rendering noises even rendered at 4000 spp. This culling process reduced the total number of source scenes to 1465. These remaining scenes produced 2449 images by rendering from 1 to 10 viewpoints per scene. We split the dataset into 3 subsets: 2126 images from 1283 scenes as the training set, 193 images from 76 scenes as the validation set, and 130 images from 104 scenes as the test set. There is no overlap scene among them. As shown in Figure 2, our dataset covers various optical phenomena, such as motion blur, depth of field, and complex light transport effects. It covers a variety of scene contents, including indoor scenes, buildings, landscapes, fruits, plants, vehicles, animals, glass, and so on.

3.2 Rendering Settings

To generate the high-quality “ground-truth” renderings, we rendered each scene at 4000 spp. As described previously, we noticed that the rendered images for some scenes still contain noticeable noises even when rendered at 4000 spp and we removed them through manual inspection. On average, it took around 20 minutes to render an image on an Nvidia Titan X Pascal GPU. We set the rendering resolution to 1920 × 1080 or 1080 × 1080 to cover the most content of scenes. For each image, we provide both the final rendered image and the render layers, which are essential for Monte Carlo rendering algorithms, including Albedo, Normal, Diffuse, Specular, Variance layer RGB channel, while Sub= ISubDir + ISubInd, Trans = ITransDir + ITransInd.

Besides rendering 4000spp images as ground truth, we rendered each scene at 1-8, 12, 16, 32, 64, 128, 250, and 1000 spp as input for Monte Carlo rendering enhancement algorithms, including ours. The rendered images and the auxiliary results in the scene were saved in the scene linear color space, which closely corresponds to natural colors [5]. These images were rendered with a high dynamic range. The pixel values were represented in Float32. As shown in Figure 3, 99.98% pixel values were in the range of [0, 10]. However, the pixel value distribution had a long tail. We also noticed that many of the very large values come from the firefly rendering artifacts. Therefore we removed these outliers by clipping at value 100. An image in the scene linear space can be converted to sRGB space for visualization in this paper as follows.

\[
s = \begin{cases} 
0 & \text{if } l \leq 0, \\
12.92 \cdot l & \text{if } 0 < l \leq 0.0031308, \\
1.055 \cdot \log_{255}(l) - 0.055 & \text{if } 0.0031308 < l < 1, \\
1 & \text{if } l \geq 1,
\end{cases}
\]

where \(l\) and \(s\) indicate the pixel value in scene linear color space and sRGB respectively [4].

3.3 Low Resolution Image Generation

A straightforward way to generate low resolution images is to change the output resolution in Cycles. However, directly rendering a low resolution image does not always work [7]. For example, some scenes are modelled using a subdivision technology and changing the rendering resolution will disrupt the inherent relationship among the material and geometry settings in the scene files and thus cause mismatch between images rendered at different resolutions. Therefore, we generate low resolution images by downsampling the corresponding high resolution rendered images via the nearest neighbour degradation. We did not use bilinear or bicubic sampling as the nearest neighbor degradation more accurately simulates a real-world rendering engine. That is, rays for low resolution renderings are sampled at a sparse grid compared with high resolution ones.

3.4 Monte Carlo Rendering Dataset Comparison

We compare our dataset with those used in recent deep learning-based Monte Carlo rendering denoising algorithms, including Kuznetsov [28], KPCN [2], and RDGs, we combine it with residual dense groups (RDG) [53, 54]. With the feature network firstly extracts the features \(I_{LRHS}\) and downsampling \(HRLS\) features to the same size of \(F_{HRHS}\) as input and aims to estimate a corresponding HR image \(I_{HRHS}\) that contains the RGB channel, while \(I_{LRHS}\) is composed of RGB channel and extra layers, including Albedo, Normal, Diffuse, Specular, Variance layer as these extra layers can provide high-frequency visual details.

As shown in Figure 4, we design a two-encoder-one-decoder network to estimate the HR image. Given \(I_{LRHS}\) and \(I_{LRHS}\), our network firstly extracts the features \(F_{LRHS}\) and \(F_{HRHS}\), respectively. We leverage a downscale module with deshuffle layers instead of pooling layers to downscale the feature maps as deshuffle layers can keep the high-frequency information. Compared with upsampling LRHS features, downsampling HRHS features to the same size of \(F_{LRHS}\) can reduce the computational complexity of the network significantly. It also enables the features to fuse in the earlier layer of the network. We obtain the fused feature \(F_0\) by combining \(F_{LRHS}\) through a fusion module and feed it to a sequence of residual dense groups (RDG) [53, 54]. With the feature \(F_0\) from RDGs, we combine it with \(F_{LRHS}\) by element-wise adding. Finally, we upscale the resulting dense feature \(F_{DF}\) and predict the final HR image \(I_{HRHS}\) by convolutional layer. Below we describe the network in detail.
we build a residual densely connected group (RDG) as follows, where:

\[ F_{RDG} = H_{RDG}(I_{LRHS}). \]  

**(HRLS shallow feature)** \( F_{LRHS} \). Following [34, 52, 54], we adopt a convolutional layer to get the shallow feature \( F_{LRHS} \):

\[ F_{LRHS} = H_{LRHS}(I_{LRHS}). \]  

where \( H(\cdot) \) indicates the convolution operation.

**(HRLS shallow feature)** \( F_{LRHS} \). We first extract the shallow feature from \( I_{LRHS} \) with a convolutional layer,

\[ F_{0} = H_{LRHS}(I_{LRHS}). \]  

Inspired by ESPCN [45], we design a deshuffle layer to downscale the features. As shown in Figure 5, we downscale the feature map with a stride of \( \alpha \). In our network, we set \( \alpha = 2 \). To downscale the feature map, we stack deshuffle layers together. Supposing our network has \( D \) deshuffle layers, we can get the output \( F_{HRLS} \)

\[ F_{HRLS} = DSF^{-1}(DSF^{-1}(\cdots DSF^{-1}(F_{0})))) \]  

where \( DSF(\cdot) \) indicates the operation of the deshuffle layer. By downscaling auxiliary features, our network can work in the size of the LRHS image, which can significantly reduce the computational complexity of the overall network.

We concatenate \( F_{LRHS} \) from LRHS image and \( F_{HRLS} \) from HRLS into a combined feature map \( F_{0} \).

**Residual densely connected block.** We employ the densely connected network and residual groups to build the backbone of our neural network as they are shown effective for image super resolution [53, 54]. In our network, we use 4 convolutional layers in each residual densely connected block (RDB). By stacking \( B = 5 \) RDBs, we build a residual densely connected group (RDG) as follows,

\[ F_{g} = RDB_{g}(RDB_{g-1}(\cdots RDB_{1}(F_{g-1})))) + F_{g-1} \]  

We predict the dense feature \( F_{DF} \) with \( G = 3 \) RDGs as follows,

\[ F_{DF} = RDB_{G}(RDB_{G-1}(\cdots RDB_{1}(F_{0})))) + F_{0} \]  

**Upscale.** In our network, we adopt the shuffle layer from ESPCN [45] to upscale the features and estimate the high resolution prediction \( I_{SR} \):

\[ I_{SR} = H_{Rec}(UP(F_{DF})), \]  

where \( UP(\cdot) \) indicates the operation of upscale [45].

**Loss function.** The BCR dataset is in the scene linear color space. As shown in Figure 5, the pixel value distribution of this BCR dataset has a long tail. \( \ell_{1} \) loss cannot handle it well because it might be biased to the extremely large pixel values. To handle this problem, we adopt the following robust loss

\[ \ell_{r} = \frac{1}{N} \sum_{p \in F_{SR}} |F_{SR}^{p} - F_{LR}^{p}|^{\beta}, \]  

where \( \beta \) indicates the robust factor. For the small difference, \( \ell_{r} \) works quite similarly to \( \ell_{1} \). For the extremely large difference, \( \ell_{r} \) will be close to but always below 1. This will prevent our network from the bias towards rare but extremely large pixel values. We set \( \beta = 0.1 \) in our experiments.

**Implement details.** We set the kernel size of all convolutional layers to \( 3 \times 3 \), except for the fuse convolutional layer, whose kernel size is \( 1 \times 1 \). Every convolutional layer is followed by a RELU layer, except for the last convolutional layer. The shallow features, fusion features, and dense features have 64 channels. During each iteration of the training, we randomly select the spp of \( I_{LRHS} \) from the set of \{1-8, 12, 16, 32\} and the spp of \( I_{HRLS} \) from the set of \{2-8, 12, 16, 32, 64, 128, 250, 500, 1000, 4000\} while making sure that the spp of \( I_{HRLS} \) is smaller than that of \( I_{LRHS} \).

We use PyTorch to implement our network. We use a mini-batch size of 16 and train the network for 500 epochs. It takes about one week on one Nvidia Titan Xp for training. We use the SGD optimizer with the learning rate of \( 10^{-4} \). We also perform data augmentation on-the-fly by randomly cropping patches. In order to save data loading time, we pre-crop training HR images into \( 300 \times 300 \) large patches. During training, we further crop smaller patches on those large patches. The final patch size of HR is set to \( 96 \times 2 \), 192 for \( \times 4 \), and 256 for \( \times 8 \). We select the model that works best on the validation set.

**5 Experiments**

We evaluate our method by comparing it with representative state-of-the-art denoising methods for Monte Carlo rendering and image super resolution algorithms. We also conduct ablation studies to further examine our method. We use two metrics to evaluate our results. First, we adopt RelMSE (Relative Mean Square Error) to report the results in the scene linear color space, which is defined as

\[ \text{RelMSE} = \lambda_{1} \frac{(I_{SR} - I_{HR})^{2}}{I_{SR} + \lambda_{2}}, \]  

where \( \lambda_{1} = 0.5 \) and \( \lambda_{2} = 0.01 \) when experimenting on our BCR dataset following KPCN [2]. For the Gharbi dataset, we use the evaluation code from its authors [14] where \( \lambda_{1} = 1 \) and \( \lambda_{2} = 10^{-4} \).

We also use PSNR to evaluate the results in the sRGB space. For our BCR dataset, we convert images to sRGB to calculate PSNR use Equation [5]. For the Gharbi dataset, we convert images to the sRGB space using codes provided by its authors as follows [14].

\[ s = \min(1, \max(0, l)), \]  

Figure 4: The architecture of our network. Our network takes a low-resolution-high- spp rendering (LRGB) and its corresponding high-resolution-low-spp rendering (HRLS) as input and predicts the final high-resolution-high-quality image.
where \( l \) indicates images in the scene linear space, \( s \) indicates images in the sRGB space.

### 5.1 Comparison with Denoising Methods

We compare our method to both state-of-the-art traditional denoising methods, including Sen et al. [44], Rousselle et al. [41], Kalantari et al. [23], Bitterli et al. [3], and recent representative deep learning based methods, including KPCN [2] and Gharbi et al. [14]. Unlike other methods, our method takes both a LRHS rendering and a HRLS rendering as input. Therefore, we compute the average spp for our input as \( spp_{avg} = spp_{LRHS}/s^2 + spp_{HRLS} \), where \( s \) indicates the super resolution scale. For instance, in Table 3 "Ours \( \times 2 \)" indicates that our method performs \( \times 2 \) super resolution and \( (4 - 1) \) indicates that our method takes 4 spp LRHS and 1 spp HRLS as input, which is effectively \( 2 \) spp on average. We conducted on the comparisons on both the Gharbi dataset and our BCR dataset.

Table 2 compares our method to Bitterli et al. [3], KPCN [2], and Gharbi et al. [14]. We used the code / model shared by their authors in this experiment. For KPCN [2], we provide another version of the results produced by their neural network but fine-tuned on our BCR dataset. This experiment shows that our method, especially ours \( \times 2 \), outperform the state-of-the-art methods by a large margin. For KPCN [2], we provide another version of the results produced by their neural network but fine-tuned on our BCR dataset. This experiment shows that our method, especially ours \( \times 2 \), outperform the state-of-the-art methods by a large margin.

Table 2: Comparison on our BCR dataset. Ours \( \times 2 \) indicates that our method performs \( x2 \) super resolution and \( (4 - 1) \) indicates that our method takes 4 spp LRHS and 1 spp HRLS as input, which is effectively \( 2 \) spp on average for all the pixels.

| Method       | 2 spp | 4 spp | 8 spp | 16 spp |
|--------------|-------|-------|-------|--------|
|               | PSNR  | RelMSE| PSNR  | RelMSE|
| Input        |       |       |       |        |
| KPCN [2]     | 18.12 | 0.2953| 21.51 | 0.1400|
|               | 25.87 | 0.0390| 27.31 | 0.0299|
| KPCN-H [4]   | 31.03 | 0.0078| 33.69 | 0.0043|
| Bitterli [5] | 26.67 | 0.0293| 27.22 | 0.0252|
| Gharbi [14]  | 30.73 | 0.0068| 31.61 | 0.0057|
| Ours \( \times 2 \) | (4 - 1) | (3 - 2) | (2 - 1) | (1 - 1) |
|               | 33.27 | 0.0044| 35.15 | 0.0027|
| Ours \( \times 4 \) | (6 - 4) | (9 - 6) | (12 - 8) | (16 - 10) |
|               | 33.94 | 0.0039| 35.21 | 0.0028|
| Ours \( \times 8 \) | (12 - 8) | (24 - 16) | (36 - 24) | (48 - 32) |
|               | 31.37 | 0.0075| 32.35 | 0.0057|
|               |       |       |       |        |

Table 3: Comparison on the Gharbi dataset [14].

| Method       | 4 spp | 8 spp | 16 spp |
|--------------|-------|-------|--------|
|               | PSNR  | RelMSE| PSNR  | RelMSE|
| Input        |       |       |       |        |
| Sen [44]     | 19.58 | 17.5358| 21.91 | 7.5682|
| Rousselle [41]| 28.23 | 1.0484| 30.92 | 0.5747|
| Kalantari [23]| 30.01 | 1.9407| 31.66 | 0.8616|
| Bitterli [3] | 28.98 | 1.9407| 31.66 | 0.8616|
| KPCN [2]     | 29.75 | 1.9407| 31.66 | 0.8616|
| KPCN-H [4]   | 29.86 | 1.9407| 31.66 | 0.8616|
| Gharbi [14]  | 33.11 | 0.0486| 34.45 | 0.0385|
| Ours \( \times 2 \) | (8 - 2) | (16 - 4) | (32 - 8) |
|               | 34.02 | 1.5025| 35.30 | 1.4902|
| Ours \( \times 4 \) | (16 - 8) | (32 - 16) | (64 - 32) |
|               | 33.94 | 5.5586| 35.22 | 5.6781|
| Ours \( \times 8 \) | (32 - 16) | (64 - 32) | (128 - 64) |
|               | 31.56 | 3.7228| 32.60 | 4.2300|
|               |       |       |       |        |

Table 4: Comparison of runtime cost (second) to denoise a 1024 × 1024 image. The data is from Gharbi [14]. If the runtime is constant, we report it in the last column. Our method outperforms all the other methods in terms of PSNR.

We removed one image from the Gharbi testing set as its source model is also included in our BCR training set. We report the speeds of the above methods in Table 4. Ours \( \times 2 \) indicates that our method performs \( \times 2 \) super resolution and \( (4 - 1) \) indicates that our method takes 4 spp LRHS and 1 spp HRLS as input, which is effectively \( 2 \) spp on average.

Table 4: Comparison of runtime cost (second) to denoise a 1024 × 1024 image. The data is from Gharbi [14]. If the runtime is constant, we report it in the last column. Our method outperforms all the other methods in terms of PSNR.

**Figure 6:** Error map visualization.
Figure 7: Visual comparison on the BCR dataset.

Figure 8: Visual comparison on the Gharbi dataset.
We now examine several key components of our method. We also compare our method with several baseline methods that use denoising methods \[2, 14\].

For the model with “None”, we remove this branch. As shown in Table 6, the upsampling scale is set to \(\times 4\). The input channels of \(I_{LRHS}\) and \(I_{HRLS}\) are set to RGB. Figure 10 shows that the robust loss \(\ell_1\) with \(\beta = 0.1\) outperforms \(\ell_1\) by a large margin as it can avoid the bias towards a very small number of pixels with extremely large pixel values. We also find that using a too large or too small \(\beta\) will be harmful to the results. This is because a very large \(\beta\) value reduces the robust loss to the \(\ell_1\) loss while a very small beta value makes the loss always close to 1 without regard to the error between the output and the ground truth.

**SPP of \(I_{LRHS}\) and \(I_{HRLS}\)**. We examine how our method works with different spp values used to render \(I_{LRHS}\) and \(I_{HRLS}\). In the experiment, we set the upsampling scale to \(\times 4\). Table 6 shows rendering at high spp values consistently leads to better final results.

### 5.2 Comparisons with Super Resolution Methods

We also compare our method with several baseline methods that use super resolution to upsample the low-resolution-high-spp renderings to the target size. In this experiment, we used the trained models shared by the authors of these super resolution methods \[31, 53\] and fine-tuned them on our BCR dataset. As reported in Table 5, our method generates significantly better results than these super resolution methods. While this comparison is unfair to these baseline methods, it indeed shows the benefits of taking an extra high-resolution-low-spp rendering as input. As shown in Figure 11, our results contain more fine details that are missing from the super resolution results.

### 5.3 Ablation study

We now examine several key components of our method. **Input layers of \(I_{LRHS}\) and \(I_{HRLS}\)**. We examine how the rendering layers affect the final results. In this experiment, we use \(1\) spp for \(I_{LRHS}\) and 4000 spp for \(I_{HRLS}\). The upsampling scale is set to \(\times 4\). Our neural network contains two input branches, one for \(I_{LRHS}\) and the other for \(I_{HRLS}\). In this experiment, we fix the input layer of one branch to RGB while changing the input layers of the other. For the model with “None”, we remove this branch. As shown in Figure 9, compared with no inputs, \(I_{HRLS}\) can greatly improve the results. Among various input layers, RGB improves the results by a large margin. The result can be further improved if the \(I_{LRHS}\) takes all rendering layers. We believe that these improvements come from the high frequency information in the \(I_{LRHS}\). For the \(I_{LRHS}\), while all the input layers still help, the RGB result alone can achieve the best result. We conjecture that since \(I_{LRHS}\) is rendered with a high spp, its RGB layer is already of very high quality, and the other intermediate layers do not further contribute. On the other hand, the intermediate layers for \(I_{HRLS}\) provide useful information for denoising, which is consistent with the findings of the previous denoising methods \[2, 14\].

**Robust loss \(\ell_1\)**. We examine the effect of the parameter \(\beta\) in our robust loss. We also compare it to the standard \(\ell_1\) loss. In this experiment, we use 4000 spp for \(I_{LRHS}\) and 1 spp for \(I_{HRLS}\). The
Ground Truth
×4

EDSR [54] 31.83/0.0045
RCAN [53] 31.79/0.0046
Ours 34.39/0.0025

EDSR [54] 25.41/0.0231
RCAN [53] 25.41/0.0226
Ours 29.07/0.0089

EDSR [54] 29.08/0.0111
RCAN [53] 28.99/0.0115
Ours 33.01/0.0033

Figure 11: Visual comparison with super-resolution methods on the BCR dataset.

networks for denoising monte carlo renderings. ACM Transactions on Graphics, 36(4):97, 2017.

[3] B. Bitterli, F. Rousselle, B. Moon, J. A. Iglesias-Guitián, D. Adler, K. Mitchell, W. Jarosz, and J. Novák. Nonlinearly weighted first-order regression for denoising monte carlo renderings. In Computer Graphics Forum, vol. 35, pp. 107–117. Wiley Online Library, 2016.

[4] Blender. Blender color linear to srgb. https://github.com/blender/blender/blob/6c9178b183f5267e87abc55497bb6496e468a709/intern/cycles/util/util_color.h#L77.

[5] Blender. Blender color management. https://docs.blender.org/manual/en/dev/render/color_management.html 2020.

[6] Blender. Blender passes. https://docs.blender.org/manual/en/latest/render/layers/passes.html 2020.

[7] Blender. Cycles design goals. https://wiki.blender.org/wiki/Source/Render/Cycles/DesignGoals 2020.

[8] M. R. Bolin and G. W. Meyer. A perceptually based adaptive sampling algorithm. Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 299–309, 1998.

[9] C. R. A. Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai, and T. Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. ACM Transactions on Graphics, 36(4):98, 2017.

[10] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In ACM SIGGRAPH computer graphics, vol. 18, pp. 137–145, 1984.

[11] H. Dammertz, D. Sewtz, J. Hanika, and H. Lensch. Edge-avoiding `a-trous wavelet transform for fast global illumination filtering. In Proceedings of the Conference on High Performance Graphics, pp. 67–75. Eurographics Association, 2010.

[12] C. Dong, C. C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In European conference on computer vision, pp. 184–199. Springer, 2014.

[13] K. Egan, Y.-T. Tseng, N. Holzschuch, F. Durand, and R. Ramamoorthi. Frequency analysis and sheared reconstruction for rendering motion blur. In ACM Transactions on Graphics, vol. 28, p. 93, 2009.

[14] M. Gharbi, T.-M. Li, M. Aittala, J. Lehtinen, and F. Durand. Sample-
based monte carlo denoising using a kernel-splatting network. *ACM Transactions on Graphics*, 38(4):1–12, 2019.

[15] M. Haris, G. Shakhnarovich, and N. Ukita. Recurrent back-projection network for video super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3897–3906, 2019.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[17] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

[18] G. Huang, Z. Liu, L.-W. Wang, C.-T. Li, and W.-C. Siu. Hierarchical back projection network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.

[19] Z. Hui, X. Wang, and X. Gao. Fast and accurate single image super-resolution via information distillation network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 723–731, 2018.

[20] H. W. Jensen. *Realistic image synthesis using photon mapping*. AK Peters/CRC Press, 2001.

[21] H. W. Jensen and N. J. Christensen. Optimizing path tracing using noise reduction filters. 1995.

[22] J. T. Kajiya. The rendering equation. *ACM SIGGRAPH computer graphics*, 20(4):143–150, 1986.

[23] N. K. Kalantari, S. Bako, and P. Sen. A machine learning approach for filtering monte carlo noise. *ACM Trans. Graph.*, 34(4):122–1, 2015.

[24] S. Kallweit, T. Müller, B. Mcwilliams, M. Gross, and J. Novák. Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Transactions on Graphics*, 36(6):1–11, 2017.

[25] A. Keller, L. Fascione, M. Fajardo, I. Georgiev, P. H. Christensen, S. Kallweit, T. Müller, and B. Mcwilliams. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[26] J. Kim, J. Kwon Lee, and K. Mu Lee. Deeply-recursive convolutional network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1637–1645, 2016.

[27] M. Koskela, K. Immonen, M. Mäkitalo, A. Foi, T. Viitanen, P. Jääskeläinen, H. Kultala, and J. Takala. Blockwise multi-order feature regression for real-time path-tracing reconstruction. *ACM Transactions on Graphics*, 38(5):138, 2019.

[28] A. Kuznetsov, N. K. Kalantari, and R. Ramamoorthi. Deep adaptive sampling for low sample count rendering. In *SIGGRAPH Courses*, pp. 24–1, 2015.

[29] J. Kim, J. Kwon Lee, and G. Iehl, C. Eisenacher, and G. Nichols. The path tracing revolution in the movie industry. In *SIGGRAPH Courses*, pp. 24–1, 2015.

[30] J. Kuznetsov, N. K. Kalantari, and R. Ramamoorthi. Deep adaptive sampling for low sample count rendering. In *Computers Graphics Forum*, vol. 37, pp. 35–44, 2018.

[31] S. Laine, H. Saransaari, J. Kontkanen, J. Lehtinen, and T. Aila. Incremental instant radiosity for real-time indirect illumination. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pp. 277–286. Eurographics Association, 2007.

[32] M. E. Lee and R. A. Redner. A note on the use of nonlinear filtering in computer graphics. *IEEE Computer Graphics and Applications*, 10(3):23–29, 1990.

[33] T. Leinkühler, H.-P. Seidel, and T. Ritschel. Laplacian kernel splatting for efficient depth-of-field and motion blur synthesis or reconstruction. *ACM Transactions on Graphics*, 37(4), 2018.

[34] T.-M. Li, Y.-T. Wu, and Y.-Y. Chuang. Sure-based optimization for adaptive sampling and reconstruction. *ACM Transactions on Graphics*, 31(6):194, 2012.

[35] Y. Li, V. Tsiminaki, R. Timofte, M. Pollefeys, and L. V. Gool. 3d appearance super-resolution with deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9671–9680, 2019.

[36] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 136–144, 2017.

[37] Z.-S. Liu, L.-W. Wang, C.-T. Li, and W.-C. Siu. Hierarchical back projection network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.

[38] M. D. McCool. Anisotropic diffusion for monte carlo noise reduction. *ACM Transactions on Graphics*, 18(2):171–194, 1999.

[39] S. U. Mehta, J. Yao, R. Ramamoorthi, and F. Durand. Factored axis-aligned filtering for rendering multiple distribution effects. *ACM Transactions on Graphics*, 33(4):57, 2014.

[40] M. Meyer and J. Anderson. Statistical acceleration for animated global illumination. *ACM Transactions on Graphics*, 25(3):1075–1080, 2006.

[41] J. Moom, S. McDonagh, K. Mitchell, and M. Gross. Adaptive polynomial rendering. *ACM Transactions on Graphics*, 35(4):40, 2016.

[42] R. S. Overbeck, G. Iehl, C. Eisenacher, and B. Péroche. Non-interleaved deferred shading of interleaved sample patterns. In *Graphics Hardware*, pp. 53–60, 2006.

[43] P. Sen and S. Darabi. On filtering the noise from the random parameters in monte carlo rendering. *ACM Trans. Graph.*, 31(3):18–1, 2012.

[44] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1874–1883, 2016.

[45] B. Walter, A. Arbree, K. Bala, and D. P. Greenberg. Multidimensional lightcuts. *ACM Transactions on graphics*, 25(3):1081–1088, 2006.

[46] G. J. Ward, F. M. Rubinstein, and R. D. Clear. A ray tracing solution for diffuse interreflection. *ACM SIGGRAPH Computer Graphics*, 22(4):85–92, 1988.

[47] L. Wu, L.-Q. Yan, A. Kuznetsov, and R. Ramamoorthi. Multiple axis-aligned filters for rendering of combined distribution effects. In *Computer Graphics Forum*, vol. 36, pp. 155–166. 2017.

[48] R. Xu and S. N. Pattanaik. A novel monte carlo noise reduction operator. *IEEE Computer Graphics and Applications*, 25(2):31–35, 2005.

[49] X. Xu, Y. Ma, and W. Sun. Towards real scene super-resolution with raw images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1723–1731, 2019.

[50] L.-Q. Yan, S. U. Mehta, R. Ramamoorthi, and F. Durand. Fast 4d sheared filtering for interactive rendering of distribution effects. *ACM Transactions on Graphics*, 35(1):7, 2015.

[51] K. Zhang, W. Zuo, and L. Zhang. Learning a single convolutional super-resolution network for multiple degradations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3262–3271, 2018.

[52] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu. Image super-resolution using very deep residual channel attention networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 286–301, 2018.

[53] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu. Residual dense network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2472–2481, 2018.

[54] Z. Zhang, Z. Wang, Z. Lin, and H. Qi. Image super-resolution by neural texture transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7982–7991, 2019.

[55] H. Zimmer, F. Rousselle, W. Jakob, O. Wang, D. Adler, W. Jarosz, O. Sorkine-Hornung, and A. Sorkine-Hornung. Path-space motion estimation and decomposition for robust animation filtering. In *Computer Graphics Forum*, vol. 33, pp. 131–142, 2015.

[56] M. Zwicker, W. Jarosz, J. Lehtinen, B. Moon, R. Ramamoorthi, F. Rousselle, P. Sen, C. Soler, and S.-E. Yoon. Recent advances in adaptive sampling and reconstruction for monte carlo rendering. In *Computer Graphics Forum*, vol. 34, pp. 667–681, 2015.