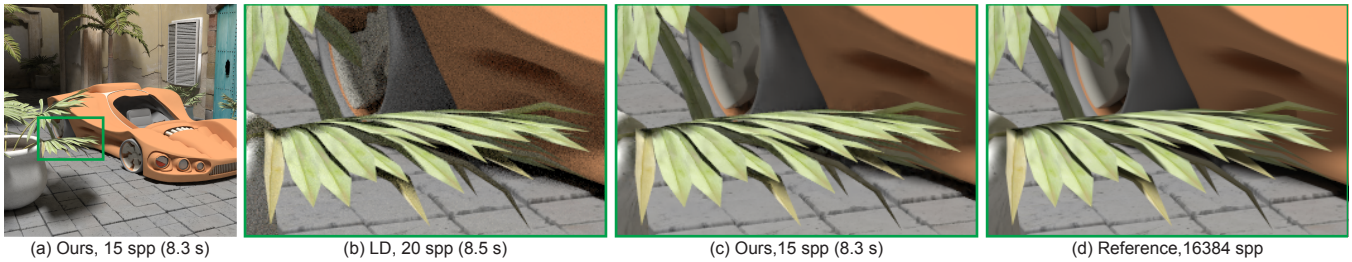
Adaptive Rendering with Linear Predictions

Bochang Moon[1], Jose A. Iglesias-Guitian[1], Sung-Eui Yoon[2], Kenny Mitchell[1]
[1]Disney Research Zürich, [2]KAIST



(a) Ours, 15 spp (8.3 s) rMSE 0.00199 | (b) LD, 20 spp (8.5 s) rMSE 0.01003 | (c) Ours,15 spp (8.3 s) rMSE 0.00199 | (d) Reference,16384 spp

Figure 1: *Our adaptive rendering result in the Courtyard scene. Our method with 15 samples per pixel (spp) produces a high-quality reconstruction result and drastically reduces a relative mean squared error (rMSE) compared to a straight-forward method utilizing low discrepancy (LD) sampling patterns uniformly.*

Abstract

We propose a new adaptive rendering algorithm that enhances the performance of Monte Carlo ray tracing by reducing the noise, i.e., variance, while preserving a variety of high-frequency edges in rendered images through a novel prediction based reconstruction. To achieve our goal, we iteratively build multiple, but sparse linear models. Each linear model has its prediction window, where the linear model predicts the unknown ground truth image that can be generated with an infinite number of samples. Our method recursively estimates prediction errors introduced by linear predictions performed with different prediction windows, and selects an optimal prediction window minimizing the error for each linear model. Since each linear model predicts multiple pixels within its optimal prediction interval, we can construct our linear models only at a sparse set of pixels in the image screen. Predicting multiple pixels with a single linear model poses technical challenges, related to deriving error analysis for regions rather than pixels, and has not been addressed in the field. We address these technical challenges, and our method with robust error analysis leads to a drastically reduced reconstruction time even with higher rendering quality, compared to state-of-the-art adaptive methods. We have demonstrated that our method outperforms previous methods numerically and visually with high performance ray tracing kernels such as OptiX and Embree.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

Keywords: Adaptive rendering, image-space reconstruction, Monte Carlo ray tracing

1 Introduction

Monte Carlo (MC) ray tracing [Kajiya 1986] has received extensive attention for synthesizing realistic rendering results, but generally requires a huge number of ray samples (e.g., more than ten thousands samples per pixel) until a converged or even visually pleasing image is generated. Unfortunately, its slow convergence directly leads to prohibitive rendering time (e.g., hours), which is often proportional to the number of ray samples generated. When a relatively small number of ray samples (e.g., less than one hundred) per pixel are allocated, images are typically corrupted by MC noise, i.e., variance.

Adaptive rendering that adjusts sampling density non-uniformly and applies smoothing locally has been actively studied recently, as the approach significantly boosts MC ray tracing by reducing the required number of ray samples drastically [Hachisuka et al. 2008; Overbeck et al. 2009]. These methods can be classified into two categories, multi-dimensional and image space adaptive rendering in terms of the dimensionality of MC samples [Moon et al. 2014]. The multi-dimensional methods [Hachisuka et al. 2008; Lehtinen et al. 2012] allocate samples and reconstruct them in a high dimensional space, where each coordinate corresponds to a random parameter in the MC integration [Kajiya 1986]. These methods can produce a high quality image even with a small number of samples (e.g., 8 samples per pixel), but managing individual samples unfortunately requires high computational and memory overhead.

On the other hand, image space methods [Rousselle et al. 2012; Li et al. 2012; Moon et al. 2014] utilize per-pixel information (e.g., colors, variances, and G-buffers) that can be easily obtained in rendering, and thus these techniques can be easily applied into existing rendering frameworks. The state-of-the-art methods (e.g., [Rousselle et al. 2012; Li et al. 2012; Moon et al. 2014]) have been shown to improve the performance of MC ray tracing by an order of magnitude. Their main target applications, however, are often limited to offline rendering frameworks, since its computational overhead is relatively large. For example, the reconstruction times of the previuos methods [Rousselle et al. 2012; Li et al. 2012; Moon et al. 2014] are more than 3 s given the Courtyard scene (Fig. 1) due to their expensive reconstructions (e.g., non-local means and local regression). Especially, the recent local linear approximation [Moon et al. 2014] shows a superior reconstruction performance when a reference image has a strong linear correlation with given features (e.g., textures), but it has very expensive reconstruction time (e.g.,

18 s in the scene of Fig. 1), since it utilizes a complex optimization process (least-squares fitting).

To address this problem, we propose a new adaptive rendering method, which performs expensive model reconstruction and optimization only at a small number of pixels and predicts filtered values in other pixels by using estimated linear models. The key difference between our work and the previous methods is that our method estimates optimal colors in a region (i.e., multiple pixels) by performing a novel linear prediction based reconstruction. Specifically, our major technical contributions are summarized as the following:

- We construct multiple linear models iteratively by using a recursive least squares. Our method estimates coefficients of linear models recursively given prediction windows with different sizes, where we predict multiple pixels from the linear models (Sec. 4.1).
- We design a recursive error analysis to estimate the prediction error introduced by our linear prediction, and select an optimal prediction size by using the error analysis (Sec. 4.2).
- We provide an adaptive sampling approach which allocates more ray samples on high error regions based on our estimated prediction errors (Sec. 5.1).

We have demonstrated our method with high-performance ray tracing kernels such as Embree [Wald et al. 2014] and OptiX [Parker et al. 2010], and our result shows higher rendering quality compared to the state-of-the-art methods [Rousselle et al. 2012; Li et al. 2012; Moon et al. 2014] in equal-time comparisons thanks to its accurate error analysis and lower computational overhead. Our method uses a sophisticated optimization (e.g., least squares fitting), but we drastically reduce the optimization overhead (e.g., 28$\times$ lower than the previous method [Moon et al. 2014] in the Fig. 1), by running our optimization only at a sparse number of pixels thanks to our prediction, while preserving its high reconstruction quality.

2 Previous Work

Multi-dimensional adaptive rendering. As an early work, Kajiya [1986] proposed a high-level idea to allocate high-dimensional samples in a hierarchical manner and to reconstruct outputs based on the samples stored in a tree structure (e.g., kd-trees) by using a Riemann sum. In a similar research line, Hachisuka et al. [2008] refined the idea and demonstrated that this approach significantly reduces the number of samples required for synthesizing a variety of rendering effects. Frequency analysis based anisotropic reconstruction often focused on simulating specific rendering effects such as depth-of-field [Soler et al. 2009], motion blur [Egan et al. 2009], soft shadows [Egan et al. 2011b], and ambient occlusions [Egan et al. 2011a]. Lehtinen et al. [2011] presented a new reprojection method to reuse samples among multiple pixels in order to reduce noise introduced by distributed effects, and Lehtinen et al. [2012] extended the idea to support indirect illumination. These methods demonstrated that high quality reconstruction can be achieved even with a small number of samples. These techniques, unfortunately, support a limited set of rendering effects.

Image-space adaptive rendering. Image-space approaches generally take per-pixel information (e.g., colors and variances) as an input, and then apply well-known image filters such as wavelet thresholding [Overbeck et al. 2009] and Gaussian filter [Rousselle et al. 2011] to fully utilize rendering-specific information (e.g., variances). Recently, sophisticated error estimation has been developed for supporting superior filtering methods. Kalantari et al. [2013] proposed a robust error metric based on the median absolute deviation. Rousselle et al. [2012] divided input samples into two buffers (i.e., dual buffer) and estimated the error introduced by the non-local means, and Delbracio et al. [2014] proposed a new similarity measure between two patches by using a histogram constructed with samples. Li et al. [2012] introduced a general unbiased error metric (i.e., Stein's unbiased risk estimator) to improve reconstruction quality of non-linear filtering methods (e.g., cross-bilateral filter), and the error metric was also utilized to decide how to combine the filter weights computed from an input color buffer and a feature buffer (e.g., geometries) in the non-local means filter [Rousselle et al. 2013]. Although these methods employ different error analysis and filters, their common behaviour for high-quality filtering is to apply a filter with estimated optimal parameters, e.g., bandwidths, at each pixel. Fortunately, these methods can be easily parallelized thanks to their image-space nature. Nonetheless, these approaches become computationally heavy and may take a number of seconds, since they require sophisticated error analysis as well as expensive filtering for high-quality or error-guaranteeing results. As a result, these methods have not been adopted to recent interactive rendering systems such as Embree [Wald et al. 2014] and OptiX [Parker et al. 2010], which use high-performance kernels. Computationally efficient filters such as A-trous [Dammertz et al. 2010] and guided filter [Bauszat et al. 2011] have been developed for real-time rendering. The usage of the real-time filters, however, has been limited to a preview, since its filtering quality can be sub-optimal due to the lack of robust error analysis.

Most recently, Moon et al. [2014] approximated image functions with linear models locally using features (e.g., G-buffers). In addition, they developed an error estimation of a local linear approximation by decomposing its reconstruction error into bias and variance, and estimated optimal filter bandwidths for different features to minimize the error. They demonstrated that the local regression framework can produce high-quality rendering results for a variety of rendering effects. However this method, like other high-quality adaptive techniques, suffers from a high computational overhead, since expensive optimization for estimating the optimal bandwidths is performed at every single pixel. Our method also utilizes a local linear approximation using G-buffers, but the key difference is that our approach reconstructs multiple pixels simultaneously from a single linear model so that expensive error estimation can be performed only at a sparse number of pixels, resulting in a drastically reduced computational overhead.

Table 1: *Notation used throughout this paper*

Symbol	Description
y	input image generated by Monte Carlo ray tracing
$\mathbf{x}$	feature vector for a pixel, which includes its pixel position and geometries (e.g., normal, texture and depth)
$f(\mathbf{x})$	ground truth image as a function of $\mathbf{x}$
$\nabla f(\mathbf{x})$	ground truth gradient of $f(\mathbf{x})$
Ω_c^F	filtering window with a fixed size (e.g., 19 $\times$ 19) defined at center pixel c
$\Omega_c^P(k)$	prediction window with a variable size k centered at pixel c
$\hat{\beta}_c(k)$	estimated coefficients of a linear model defined within $\Omega_c^P(k)$
$\hat{f}(\mathbf{x}_i)$	predicted value at a neighboring pixel i within a prediction window $\Omega_c^P(k)$

3 Reconstruction using Local Linear Models

The ultimate goal of image reconstruction methods is to restore the ground truth image, $f(\mathbf{x})$, from an input image, y, generated by

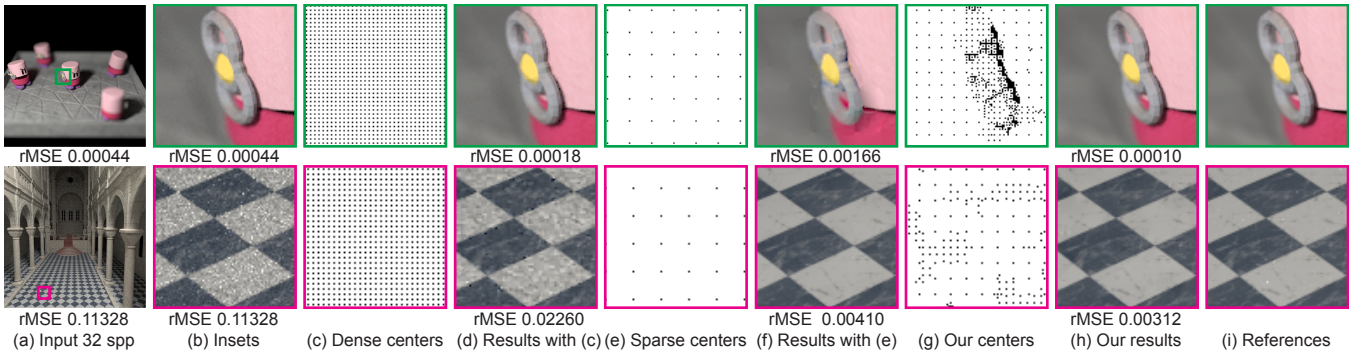


Figure 2: *Visualization of center pixels and their effects. Given uniformly sampled, i.e., 32 samples per pixel (spp), input images (a) and (b), we create our linear models at center pixels. The black dots in the image (c) indicate regularly, but densely selected center pixels with a fixed small prediction window. On the other hand, in (e), we choose regularly populated center pixels with a large fixed prediction window. When we use the small fixed prediction interval, we can preserve the detailed occlusion and features, but leave a lot of high-frequency noise (d). When using the large fixed interval, we can reduce noise well, but remove the high-frequency occlusion and features (f). On the other hand, our method adaptively selects the center pixels (g) and shows high-quality reconstruction results with lowest numerical errors.*

MC ray tracing, which is corrupted by MC noise, i.e., variance. Throughout the paper, we will use a subscript to represent the value of a function at a specific pixel. For example, $f(\mathbf{x}_i)$ and y_i denote a ground truth and input pixel value at pixel i, respectively. We summarize our notation in Table 1. To design an efficient, yet high-quality filtering technique, we propose a novel prediction based reconstruction method, which estimates the ground truth image $f(\mathbf{x})$ based on a sparse number of linear models.

Let us define a filtering window, Ω_c^F, centered at a center pixel, c. The window is considered as a set including all the pixels within the window. We also define a prediction window, $\Omega_c^P(k) \subseteq \Omega_c^F$, which has $k \equiv |\Omega_c^P(k)|$ pixels as its elements. Note that the filtering window Ω_c^F has a globally fixed size (e.g., 19×19). The prediction window $\Omega_c^P(k)$ where we predict the ground truth image $f(\mathbf{x})$ by a linear model, however, has variable size k. Within the prediction window, we define our linear model by using the first-order Taylor polynomial from the center pixel c:

$$f(\mathbf{x}_i) \approx f(\mathbf{x}_c) + \nabla f(\mathbf{x}_c)^T(\mathbf{x}_i - \mathbf{x}_c), \tag{1}$$

where $\mathbf{x}_i$ denotes a feature vector at pixel i. The ground truth value $f(\mathbf{x}_c)$ and its gradient $\nabla f(\mathbf{x}_c)$ are unknown, but can be estimated in the least squares sense, which minimizes the sum of squared residuals between the filtered image $\hat{f}(\mathbf{x})$ reconstructed by least squares and input image y. Once the estimated gradient, $\nabla \hat{f}(\mathbf{x}_c)$, is computed, we utilize it to predict the ground truth function $f(\mathbf{x}_i)$ at i pixels within the prediction window $\Omega_c^P(k)$, where i can be the center pixel c and even other pixels, i.e., $i \neq c$, instead of fitting expensive linear models separately at other pixels within the window. Fig. 2 visualizes center pixels c where we build our linear models (black dots), and our method linearly predicts all the color values of other pixels i from a sparse number of linear models by utilizing the Taylor polynomial (Eq. 1). Even with a sparse number of linear models, we can appropriately reconstruct high-frequency details (e.g., noisy textures in the bottom row), especially when the details have a linear correlation with a rendering-specific feature (e.g., textures).

Our high-level idea of reconstructing multiple pixels within a prediction window by using a single linear model may be considered intuitive given Eq. 1. It introduces, however, novel technical challenges, since we should estimate an optimal, local prediction window, $\Omega_c^P(k_{opt})$, which minimizes our prediction error, $\xi_c(k) = \frac{1}{k}\sum_{i \in \Omega_c^P(k)}(\hat{f}(\mathbf{x}_i) - f(\mathbf{x}_i))^2$. In Fig. 2, we show the results computed by fixed prediction windows (e.g., small and large one), and these results show a noisy result or over-smoothed result. It motivates us to develop adaptive prediction sizes for high-quality reconstruction.

Technically, the prediction window $\Omega_c^P(k)$ corresponds to the interval, where we approximate the unknown functions $f(\mathbf{x}_i)$ by using the first-order Taylor polynomial (Eq. 1), and thus the predicted value $\hat{f}(\mathbf{x}_i)$ varies in terms of the size of the interval (i.e., prediction size $k \equiv |\Omega_c^P(k)|$). As a result, our first technical challenge is to efficiently construct multiple linear models with different k values, i.e., different intervals (Sec. 4.1). We then need to estimate their prediction errors, i.e., $\xi_c(k)$, depending on the size of k in order to select the optimal size k_{opt} that minimizes the error. The second technical challenge is that the optimal prediction size k_{opt} is still unknown even after computing multiple linear models, since the error $\xi_c(k)$ depends on the unknown term $f(\mathbf{x}_i)$. Therefore, we should find an estimated error $\hat{\xi}_c(k)$ and its corresponding estimated optimal prediction size $\hat{k}_{opt}$ (Sec. 4.2). To realize our high-level idea while tackling these challenges, we propose a novel algorithm to estimate the optimal prediction window $\Omega_c^P(k_{opt})$ in the subsequent section.

Linear Approximation using geometries. The linear approximation based on rendering-specific features $\mathbf{x}$ (e.g., data in the G-buffer) in Eq. 1 was previously studied for filtering Monte Carlo noise [Bauszat et al. 2011; Moon et al. 2014], but the previous methods do not fully utilize $\nabla f(\mathbf{x}_c)$ for reconstructing multiple pixels of $\Omega_c^P(k)$. Our method, however, reconstructs all pixels within the prediction window Ω_c^P from a linear model simultaneously, instead of performing a filtering at every pixel.

4 Linear Model Estimation

Our optimization goal is to estimate the optimal model defined as a linear model (e.g., first-order Taylor polynomial) computed within the optimal prediction size k_{opt}, which has a minimal prediction error $\xi_c(k_{opt})$. The optimization to calculate the prediction size k_{opt} can be formulated as follows:

$$k_{opt} = \underset{k}{\operatorname{argmin}}\, \xi_c(k) = \underset{k}{\operatorname{argmin}}\, \frac{1}{k} \sum_{i \in \Omega_c^P(k)} \left(\hat{f}(\mathbf{x}_i) - f(\mathbf{x}_i)\right)^2. \tag{2}$$

Note that our goal is to minimize the averaged squared difference between predicted values $\hat{f}(\mathbf{x}_i)$ and ground truth values $f(\mathbf{x}_i)$ over the pixels defined in the prediction window $\Omega_c^P(k)$, since we plan to predict all the pixels i in $\Omega_c^P(k)$ from a single linear model. We propose an iterative estimation process for efficiently computing linear models as a function of k in Sec. 4.1 and a recursive error analysis for computing $\hat{k}_{opt}$ in Sec. 4.2.

4.1 Recursive Reconstruction of Linear Models

Our linear model construction is the process of estimating the coefficients (i.e., intercept and gradient) of a linear function, the first-order Taylor polynomial, which correspond to the ground truth $f(\mathbf{x}_c)$ and its gradient $\nabla f(\mathbf{x}_c)$ in Eq. 1, given a prediction size (i.e., an interval of the first-order Taylor polynomial). To this end, we utilize the least squares problem to compute the optimal coefficients, which minimize the sum of squared residuals between observed noisy function y and filtered image $\hat{f}(\mathbf{x})$, i.e., $\sum_{i\in\Omega_c^P(k)}(\hat{f}(\mathbf{x}_i)-y_i)^2$. We define the estimated coefficients as a vector $\hat{\beta}_c(k) \equiv (\hat{f}(\mathbf{x}_c), \nabla\hat{f}(\mathbf{x}_c))$, which is the least squares estimator for the unknown vector $\beta_c(k) \equiv (f(\mathbf{x}_c), \nabla f(\mathbf{x}_c))$.

Given this least squares problem, we propose a recursive algorithm for computing the least squares solution $\hat{\beta}_c(k)$ within a prediction size k. To compute multiple linear models, each of which is constructed within a different prediction size k, one can apply the *normal equation*, $\hat{\beta}_c(k) = (X_k^T X_k)^{-1} X_k^T Y_k$, where X_k is $k\times(d+1)$ design matrix whose i-th row is set as $(1, (\mathbf{x}_i - \mathbf{x}_c)^T)$ and d is the length of the feature vector $\mathbf{x}_i$. The design matrix is filled with the features from k pixels. Analogously, each element in the vector $Y_k = (y_1, ..., y_k)^T$ is set with the intensities of k different pixels from the Monte Carlo input image y. We consider y_i as a 1D value, since we can apply our method to each channel independently for color images.

Unfortunately, the normal equation used in least squares based methods [Moon et al. 2014] commonly requires a matrix inversion, i.e., $(X_k^T X_k)^{-1}$, for each prediction size k. Furthermore, when we consider $|\Omega_c^F|$ candidates, which are computed by adding pixels one-by-one from the prediction window to the least-squared based reconstruction, we need to solve the normal equations $|\Omega_c^F|$ times. This is impractical, since this approach would require a prohibitive computational cost. Our main idea to avoid the expensive matrix inversion is to use the recursive least squares [Ljung and Söderström 1987], which updates the inverse covariance matrix, $P_c(k) \equiv (X_k^T X_k)^{-1}$, incrementally without performing the matrix inversion.

Specifically, we update the inverse covariance matrix $P_c(k)$ and the corresponding linear model $\hat{\beta}_c(k)$ by using both $\mathbf{x}_k$ and y_k at the k-th pixel from the ones computed using prior $k-1$ pixels as follows:

$$
\begin{aligned}
G_c(k) &= \frac{P_c(k-1)\mathbf{z}_k}{1+\mathbf{z}_k^T P_c(k-1)\mathbf{z}_k},\\
P_c(k) &= P_c(k-1) - G_c(k)\mathbf{z}_k^T P_c(k-1),\\
\hat{\beta}_c(k) &= \hat{\beta}_c(k-1) + G_c(k)\left(y_k - \hat{\beta}^T(k-1)\mathbf{z}_k\right), \qquad (3)
\end{aligned}
$$

where $\mathbf{z}_k^T = (1, (\mathbf{x}_k - \mathbf{x}_c)^T)$ corresponds to the k-th row in the design matrix of the normal equation. The vector $G_c(k)$ can be considered as a weight allocated to a new sample pair, $\mathbf{x}_k$ and y_k, and the linear model $\hat{\beta}_c(k)$ is updated by considering a weighted a priori error $(y_k - \hat{\beta}^T(k-1)\mathbf{z}_k)$. Analogously, the inverse covariance matrix $P_c(k)$ is also updated by using the weight [1]. Technically, the computed linear model $\hat{\beta}_c(k)$ up to k pixels is the least squares solution, which minimizes the least squares cost function $\sum_{i\in\Omega_c^P(k)}(\hat{f}(\mathbf{x}_i)-y_i)^2$.

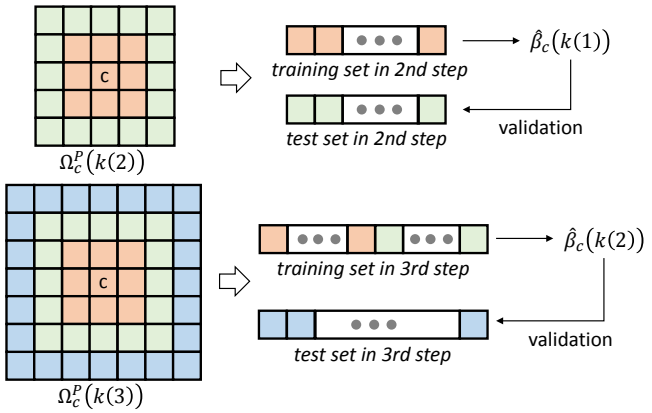


Figure 3: *We visualize how our method estimates prediction errors of linear models $\hat{\beta}_c(k(r))$, given r, (the floor integer of) the half of the width of the prediction window; $r = 2$ in the top and $r = 3$ in the bottom row. When increasing r of a square window from a prior step, we split pixels into two disjoint sets (i.e., training and test sets). We then estimate the error of a new r value and its new prediction window, by testing the prior model $\hat{\beta}_c(k(r-1))$ computed from its training set against newly included samples (i.e., the test set). This process is recursively performed for estimating the optimal prediction size k_{opt} that minimizes the prediction error.*

Note that the equation above is not an approximation of the batch solution, i.e., the normal equation, but is a recursive formula that is exactly derived based on the matrix inversion lemma (i.e., Woodbury matrix identity) from the batch version [Ljung and Söderström 1987]. Its computational complexity for updating the matrix $P_c(k)$ and $\hat{\beta}_c(k)$ with k-th pixel is $\mathcal{O}(d^2)$, where d is the length of the feature vector $\mathbf{x}_i$ containing a pixel position, normal, texture, and depth. Therefore, the complexity for computing all the linear models, each of which is constructed within a different prediction size k, is $\mathcal{O}(|\Omega_c^F|d^2)$. We explain how to find an estimated optimal model $\hat{\beta}_c(k_{opt})$ in the next section.

4.2 Recursive Estimation of Prediction Error

In this section, we explain our process of choosing the optimal prediction size $\hat{k}_{opt}$ and its corresponding optimal linear model $\hat{\beta}_c(\hat{k}_{opt})$ among possible candidates. Given our recursive reconstruction (Sec. 4.1), we have iteratively computed multiple linear models $\hat{\beta}_c(k)$ as we grow its prediction size k. To select the optimal prediction size, we should estimate the prediction error $\xi_c(k)$ introduced when we predict k pixels by the linear model $\hat{\beta}_c(k)$. To choose the optimal prediction size in an efficient and robust manner, we propose a novel, iterative technique of estimating the prediction error $\xi_c(k)$ as a function of k.

A few techniques exist for estimating reconstruction errors such as Stein's unbiased risk estimator [Li et al. 2012] and estimated mean squared error based on the asymptotic expressions of weighted local regression [Moon et al. 2014]. Unfortunately, these prior techniques utilize the general error estimation tools developed in

[1] The recursive equations have a similar structure to the Kalman filter.

statistics for reducing the point error only at the center pixel, i.e., $(\hat{f}(\mathbf{x}_c) - f(\mathbf{x}_c))^2$. As a result, for our optimization goal (Eq. 2), we take a more aggressive approach and attempt to estimate the prediction error $\xi_c(k)$ defined in multiple pixels, since we plan to predict values of the k pixels in the prediction window $\Omega_c^P(k)$ based on a single linear model $\hat{\beta}_c(k)$.

Our high-level idea for addressing both accuracy and efficiency of evaluating the prediction error is to fully utilize the recursive least squares (Sec. 4.1), where we can naturally predict subsequent samples from previous samples. For example, given a linear model $\hat{\beta}_c^T(k-t)$ computed with $k-t$ pixels, we can estimate its prediction error in the next t steps as $(\hat{\beta}_c^T(k-t)\mathbf{z}_i - y_i)^2$ with newly added t samples before updating the linear model with those t samples. In this context, let us call the t pixels *a test set* and $k-t$ pixels *a training set*. Based on this idea, we propose a new iterative validation process, which estimates prediction errors by splitting pixels into the two disjoint sets. Furthermore, we design it to have a recursive form for high efficiency when considering k different linear models.

Given a $(2R+1) \times (2R+1)$ square filtering window Ω_c^F, we estimate the prediction error, when $k \in \{1^2, 3^2, 5^2, ..., (2R+1)^2\}$; such k values are chosen by increasing half of the width (or height), r, of our prediction window, and thus are computed based on a function of r. For computational efficiency, we consider only a subset of possible prediction size k, which is defined using the half of the width (or height), r, instead of taking all positive integers. As a result, we parameterize k by $k(r)$ and formulate our iterative validation approach into the following recursion:

$$\hat{\xi}_c(k(r)) = \frac{\hat{\xi}_c^{acc}(k(r))}{(2r+1)^2} = \frac{\hat{\xi}_c^{acc}(k(r-1)) + \Delta\hat{\xi}_c^{acc}(k(r))}{(2r+1)^2}, \quad (4)$$

where $\hat{\xi}_c^{acc}(k(r))$ is the accumulated prediction error, which needs to be normalized by its pixel count, $k(r) \equiv (2r+1)^2$. We decompose the error into two terms, accumulated error from $k(0)$ and to $k(r-1)$, $\hat{\xi}_c^{acc}(k(r-1))$, and the newly added error at the current r-th step, $\Delta\hat{\xi}_c^{acc}(k(r))$.

Given this recursion, we estimate the newly added error at r-th step $\Delta\hat{\xi}_c^{acc}(k(r))$ introduced when we increase the prediction size from $k(r-1)$ to $k(r)$, by using the following equation:

$$\Delta\hat{\xi}_c^{acc}(k(r)) = \sum_{i=1}^{8r} \left(\hat{\beta}_c^T\left(k(r-1)\right)\mathbf{z}_i - y_i\right)^2, \quad (5)$$

where $\hat{\beta}_c\left(k(r-1)\right)$ is the estimated linear model from $k(r-1)$ samples and these samples are defined as the training set of the r-th step in order to test newly included $8r \equiv k(r) - k(r-1)$ samples, i.e., the test set of the step. Fig. 3 illustrates how we iteratively split samples into training and test sets. We substitute our estimated prediction error $\hat{\xi}_c(k(r))$ (Eq. 4) for the unknown error $\xi_c(k(r))$ (Eq. 2), and then select the optimal prediction size $\hat{k}_{opt}$ and its corresponding linear model $\hat{\beta}_c(\hat{k}_{opt})$.

In Fig. 4, we compare our estimated error $\hat{\xi}_c(k(r))$ with its reference error $\xi_c(k(r))$. Also, we visualize our estimated optimal prediction size $\hat{k}_{opt}$ by using our estimated error $\hat{\xi}_c(k(r))$, with a reference prediction size k_{opt} computed from the reference error $\xi_c(k(r))$. For our visualization purpose, we compare our estimations with references for all pixels although we run our reconstruction on a sparse set of image pixels. We use a reference image generated by 8K ray samples per pixel, and then plug the reference values into Eq. 2. As a result, the reference optimal size is computed by minimizing the actual L^2 error between predicted images and its reference, which cannot be achieved in practice. Our prediction size shows a similar pattern with its reference and a very high correlation (e.g., 0.87). Around the key-spindles of the toasters in Fig. 4, our method with 19×19 prediction windows has relatively high prediction errors. In these regions, we need to use smaller prediction windows and allocate more ray samples to achieve high-quality results. This is addressed in the next section.

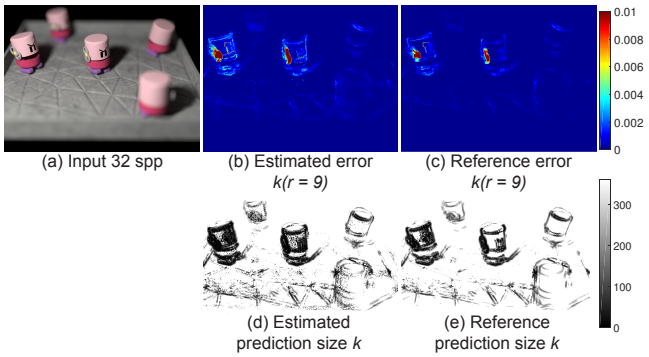


Figure 4: *We compare our estimated error and estimated optimal prediction size with a reference error and reference prediction size, respectively. Given a uniformly sampled input image (a), our estimated error (b) with a prediction size $k(r=9)$ shows a similar pattern compared to its reference (c). Also, our estimated optimal prediction size (d) has a strong correlation (e.g., 0.87) with the ground truth (e).*

5 Linear Model Construction and Adaptive Sampling

In this section, we introduce an algorithm to determine positions, i.e., center pixels c, of linear models (Sec. 5.1) and adaptive sampling process to guide additional ray samples on high error regions of our filtered image (Sec. 5.2).

5.1 Iterative Construction of Linear Models

We present a simple iterative algorithm to find center pixels c, where our local linear models (Sec. 3) are created by our recursive estimation process (Sec. 4). The computational complexity of a linear model estimation is $\mathcal{O}(|\Omega_c^F|d^2)$, and thus our overall complexity for reconstructing the values of all pixels is $\mathcal{O}(L|\Omega_c^F|d^2)$, where L is the number of linear models, i.e., the number of center pixels c. Ideally, L needs to be much smaller than the total pixel count of an input image y, while maintaining a high quality reconstruction.

On the first pass, we regularly select center pixels c by using a granularity factor g, which is initialized to a large one (e.g., width of filtering window Ω_c^F) along the X and Y directions in the screen space; for example, we choose a pixel as the center pixel c, whose x and y positions are multiples of the factor g. After we decide the center pixels, we estimate an optimal linear model within its optimal prediction size k_{opt} per each center pixel c, and then predict k_{opt} pixels from each model.

In the second pass, we reduce the global granularity factor (e.g., $g/2$), and test the pixels whose positions are multiples of $g/2$ to see whether or not each newly tested pixel is predicted by existing linear models constructed in the prior pass. If it is not reconstructed by prior prediction, mainly because of small $\hat{k}_{opt}$ values caused by drastic illumination changes, we create a new center pixel c on

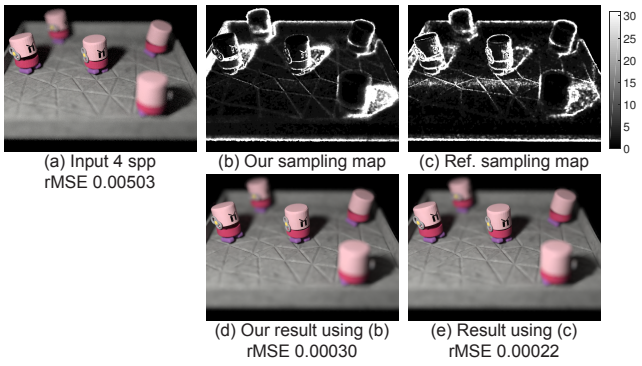


Figure 5: *Given an input image (a) generated with 4 spp, we adaptively allocate additional ray samples until the average sample count becomes 8. We visualize our sampling map (b) and its reference map (c). Our result (d) based on our map shows only 36 % higher rMSE compared to the reference result (e) computed by using the reference map (c). We achieve such a high quality result, mainly because our estimated map has a high correlation, 0.80, with its reference map (c).*

the pixel and estimate a new linear model. We repeat this process with smaller granularity factors (e.g., $g/4$ and $g/8$ for the third and fourth passes, respectively) until all the pixels are predicted from at least a single linear model. The proposed, simple iterative scheme for the model construction guarantees that every pixel is covered by at least a single linear model. On the other hand, a pixel can be predicted by multiple linear models, because prediction windows of center pixels near the pixel can overlap. In this case, we simply average multiple predictions from linear models of the respective center pixels.

Our iterative refinement is driven by a global granularity factor, but the density of linear models is automatically adjusted by locally chosen prediction sizes from each local model. The adaptive placement of linear models is conceptually similar to the lazy evaluation that iteratively caches shading values to compute indirect illumination by interpolating these values [Ward et al. 1988; Křivánek et al. 2006]. Fig. 2 visualizes selected center pixels, and shows that our method preserves fine detailed features (e.g., noisy texture) even with a small number of linear models (e.g., 6.4 % compared to the number of all pixels). As shown in the figure, our method preserves the high-frequency occlusion around the key-spindle of the toaster thanks to a high density of linear models in the area.

5.2 Adaptive Sampling

The main advantage of error analysis based reconstruction methods [Rousselle et al. 2012; Li et al. 2012; Moon et al. 2014] including our method is that we can allocate additional ray budgets adaptively on high error regions. For our adaptive sampling, we choose an iterative approach [Mitchell 1987] commonly adopted in the state-of-the-art methods. In the first pass, we uniformly allocate a small number of ray samples, e.g., 4 samples per pixel. In the following passes, we adaptively allocate additional rays based on the estimated error $\hat{\xi}_c(\hat{k}_{opt})$. Specifically, we allocate additional rays to each pixel i within the prediction window, by using the reduction of the error, $\Delta_i\hat{\xi}_c(\hat{k}_{opt})$, when a new sample is added. We adopt an estimated reduction of the error $n_i^{-4/(d+4)}$ in terms of local dimensionality d and the number of samples n_i at pixel i, which is derived in [Moon et al. 2014]. As a result, the reduction of our error at pixel i can be estimated as $\Delta_i\hat{\xi}_c(\hat{k}_{opt}) = \hat{\xi}_c(\hat{k}_{opt})n_i^{-4/(d+4)}$.

In addition, we employ a relative error metric [Rousselle et al. 2011] used in the previous methods [Rousselle et al. 2012; Li et al. 2012; Moon et al. 2014], and thus our relative error, $\rho(\Delta_i\hat{\xi}_c(\hat{k}_{opt}))$, at pixel i is set as $\Delta_i\hat{\xi}_c(\hat{k}_{opt})/(\hat{f}(\mathbf{x}_i)^2 + \epsilon)$. In practice, ϵ is set as a small constant (e.g., 0.001). Given the relative error, the number of ray samples, Δn_i, allocated to each pixel i is chosen proportionally as $\Delta n_i = T\rho(\Delta_i\hat{\xi}_c(\hat{k}_{opt}))/\sum_j \rho(\Delta_j\hat{\xi}_c(\hat{k}_{opt}))$, where T is the total number of ray samples in an iteration step.

The number of ray samples Δn_i is an estimated value, since its computation depends on the estimated error map $\hat{\xi}_c(\hat{k}_{opt})$. To verify the robustness of our sampling map, we compare our sampling map with the reference map computed by using its reference error, $\xi_c(k_{opt})$ (Fig. 5). For this test, we render the reference image $f(\mathbf{x})$ with a large number of samples in order to compute $\xi_c(k_{opt})$. To compute the reference sampling map, we use the reference error $\xi_c(k_{opt})$ based on the actual L^2 error between the predicted and reference image generated by 8K ray samples per pixel. Also, we use the reference value, $f(\mathbf{x}_i)$, when we compute the reference relative error, i.e., $\rho(\Delta_i\xi_c(k_{opt})) = \Delta_i\xi_c(k_{opt})/(f(\mathbf{x}_i)^2 + \epsilon)$. We then use this error to compute the reference number of ray samples. Our sampling map shows a similar pattern (i.e., very high correlation) with its reference even if the input image is generated with only 4 ray samples. In addition, our result based on our map shows a slightly higher error (e.g., 36 %) compared to the result using the reference map, which cannot be achieved in practice.

6 Temporal Extension

In this section, we extend our method to effectively utilize temporal coherence from past frames up to the current frame t given a scenario where users can interact with scenes (e.g., navigate a scene). At the current frame t, we first apply our prediction based reconstruction technique for a static image in order to utilize spatial coherence between pixels, and then compute the filtered image $\hat{f}(t)$. We use the filtered image as an input $y(t)$ of our temporal filtering. We create a linear model per each pixel i, and then update its coefficients at each frame.

Specifically, given the filtered image $y(t)$ generated by our method for static images, we update each linear model $\hat{\beta}_i(t)$ at pixel i in frame t by using the extended recursive least squares [Ljung and Söderström 1987]:

$$
\begin{aligned}
G_i(t) &= \frac{P_i(t-1)\mathbf{z}_i(t)}{\lambda + \mathbf{z}_i^T(t)P_i(t-1)\mathbf{z}_i(t)}, \\
P_i(t) &= \lambda^{-1}\left(P_i(t-1) - G_i(t)\mathbf{z}_i^T(t)P_i(t-1)\right), \\
\hat{\beta}_i(t) &= \hat{\beta}_i(t-1) + G_i(t)\left(y_i(t) - \hat{\beta}_i^T(t-1)\mathbf{z}_i(t)\right),
\end{aligned} \tag{6}
$$

where λ is a weight to gradually down-weight previous frames and is typically fixed to a value near one (e.g., 0.98) in practice [Ljung and Söderström 1987]. We also use a simple feature vector $\mathbf{z}_t$ defined as $\mathbf{z}_t \equiv (1, t)^T$; intuitively, we approximate a time sequence of the ground truth value as a linear function locally based on a single feature, the frame number t. Technically, the extended equation for considering temporal coherence (Eq. 6) is a least squares solution that minimizes the cost function $\sum_{j=1}^{t}\lambda^{t-j}(\hat{\beta}_i^T(j)\mathbf{z}_i(j) - y_i(j))^2$, which aims to minimize the sum of a weighted squared residual. Note that the residuals from previous frames ($j < t$) are exponentially down-weighted by λ^{t-j}.

The extended equation utilizing temporal coherence generates a temporally filtered output $\hat{\beta}_i^T(t)\mathbf{z}_i(t)$ at each pixel i for the current frame t by updating its error, $(y_i(t) - \hat{\beta}_i^T(t-1)\mathbf{z}_i(t))$, but it is more

desirable to track illumination changes by using world positions. For example, a pixel i at frame t can have a totally different illumination compared to one at frame $t-1$, when an object or the camera moves. To tackle this problem, we utilize the pixels that share similar world positions and have small prediction errors for achieving better accuracy. Specifically, given a pixel i at frame t we find a corresponding pixel o at frame $t-1$, which has the largest value of $\left(\|W_j(t-1) - W_i(t)\|^2 \times \|y_i(t) - \hat{\beta}_j^T(t-1)\mathbf{z}_i(t)\|^2\right)^{-1}$ by checking the value within a search window. W is the 3D object position that primary rays intersect. We then assign linear models $\hat{\beta}_o^T(t-1)$ and its covariance $P_o(t-1)$ stored at pixel o to $\hat{\beta}_i^T(t-1)$ and $P_i(t-1)$, and update the matrix and vector through the extended recursive least squares (Eq. 6).

We have tested our method with high-performance ray tracing kernels such as Embree [Wald et al. 2014] and OptiX [Parker et al. 2010]. In the accompanying video, we test the movement of a point light in the Toasters scene as it creates sharp illumination changes on our linear model, caused by moving shadow boundaries. Our results preserve the hard boundaries well since we reproject our models based on its prediction error as well as world positions. We have also verified our method for the Courtyard scene (Fig. 1). When the camera moves, our method still exhibits some flickering given input images generated with 4 samples per pixel. Nonetheless, we achieve 3 frames per second on average and our result shows reasonable quality for the preview purpose, especially when we consider that we use a challenging configuration with an area light and the small sample count (i.e., 4). We also provide our results with a relatively large sample count (e.g., 64) for high-quality results. In this case, the rendering time for computing our input image and our filtering time are 1.9 s and 0.3 s, respectively, on average per frame.

The proposed extension for utilizing temporal coherence can be considered as a low-pass filter, because it approximates illumination changes between frames as a low order polynomial (i.e., linear). When the illumination changes follow non-linear functions (e.g., moving highlights on glossy materials), our method can produce over-blurred results. As an interesting future work, we would like to investigate an adaptive technique to adjust the weight, λ, locally to better preserve non-linear functions. In addition, it would be interesting to automatically set the order of our temporal model locally (e.g., quadratic or cubic) instead of using a fixed order (e.g., linear) to preserve non-linear changes of illumination well.

7 Implementation Details

We have implemented our estimation and reconstruction methods using CUDA and used a 19×19 filtering window Ω_c^F for all tested benchmarks. We pre-filter input features $\mathbf{x}$ since some features can be noisy due to depth-of-field effects or motion blur. Precisely, we apply the truncated singular value decomposition (SVD) [Moon et al. 2014] to original input features within Ω_c^F in order to reduce the noise and remove dependency among features.

Initialization for recursive reconstruction of models. As the initial condition of the recursion (Eq. 3), we set $\hat{\beta}_c(0)$ as a vector (y_1,0,...,0), where y_1 is the intensity of the first pixel (i.e., center pixel c) that we include when $k = 1$. We also set $P_c(0)$ as $\delta^{-1}I$ where I is the identity matrix. The δ indicates a confidence on the $\hat{\beta}_c(0)$. For example, if we have a small δ, the recursive least squares solution changes quickly once it includes new samples. As a result, $\hat{\beta}_c(0)$ is not important when we have a very small δ. We set δ to be a small constant (e.g., 0.001) to avoid numerical instability and to give less priority on the initial $\hat{\beta}_c(0)$. For the recursion of temporal extension (Eq. 6), we initialize the matrix $P_i(0)$ and the coefficients $\hat{\beta}_i(0)$ in the same way that we apply for our static image (Eq. 3).

Initialization for recursive estimation of prediction error. To initiate the recursion (Eq. 4), we compute two initial values, $\hat{\xi}_c^{acc}(k(0))$ and $\hat{\xi}_c^{acc}(k(1))$. When $k(0) = 1^2$ we have only one sample, i.e., the center pixel c, and thus the reconstructed value $\hat{f}(\mathbf{x}_c)$ is $\hat{\beta}_c^T(1)\mathbf{z}_c = y_c$. One can easily verify this by computing $\hat{\beta}_c(1)$ (Eq. 3). In this case, our filtering does not introduce any bias, and thus we set $\hat{\xi}_c^{acc}(k(0))$ as the variance of the sample mean at the center pixel c, i.e., the error of Monte Carlo input. The variance goes to zero as the number of samples goes to infinity, and thus our method is a consistent method, since we select the smallest prediction size $k = 0$ in that case by our optimization (Eq. 2).

When $k(1) = 3^2$, we have only a single training sample and eight test samples. In this case, the least squares solution can be unstable (e.g., under-determined), since the number of samples are smaller than the local dimensionality d. To alleviate this small sample problem, we apply the leave-one-out cross validation [Kohavi 1995], where $k-1$ pixels are used to test the remaining 1 pixel. This test is repeated k times in order to test all the k pixels. Specifically, we employ the closed-form solution of the leave-one-out cross validation [Allen 1974] defined as the following:

$$\hat{\xi}_c^{acc}(k(1)) = \sum_{i=1}^{9} \left(\frac{\hat{\beta}_c^T(k(1))\,\mathbf{z}_i - y_i}{1 - \mathbf{z}_i^T P_c(k(1))\mathbf{z}_i} \right)^2. \tag{7}$$

The leave-one-out cross validation is computationally expensive, since it requires $\mathcal{O}(kd^2)$ operations per each k. Precisely, the complexity of the vector-matrix multiplication $\mathbf{z}_i^T P_c(k(1))\mathbf{z}_i$ is $\mathcal{O}(d^2)$, and it is performed k times in the summation. Therefore, we apply this approach only for the initialization of our recursion when $k = k(1) = 3^2$. Our iterative validation has a computationally preferable property that all the pixels are used as a test sample only once. Therefore, if the number of pixels in a filtering window size is large enough (e.g., $19 \times 19 = 361$) compared to $k = 3^2$, the total computational burden of our estimation has a time complexity $\mathcal{O}(|\Omega_c^F|d)$, since our method requires a dot product of two vectors (Eq. 5) and it is performed at most once per each pixel.

Alternative approaches to error estimation. As an alternative of our recursive estimation (Sec. 4.2), one may estimate the prediction error $\xi_c(k)$ using the sum of squared residuals, i.e., $\xi_c(k) = \frac{1}{k}\sum_{i \in \Omega_c^P(k)} (\hat{\beta}_c^T(k)\mathbf{z}_i - y_i)^2$ where $\hat{\beta}_c^T(k)\mathbf{z}_i = \hat{f}(\mathbf{x}_i)$. Unfortunately, it typically leads to over-fitting since its equation is very similar to the least squares cost function $\sum_{i \in \Omega_c^P(k)} (\hat{f}(\mathbf{x}_i) - y_i)^2$; the difference is just a normalization term $1/k$. Specifically, the linear model $\hat{\beta}_c(k)$ is trained to minimize its cost function based on k pixels, and thus it is undesirable to use the k pixels again for testing the linear model, i.e., measuring its prediction error, resulting in over-fitting.

For estimating the prediction error introduced by different r, one may think that we can directly use the simple cross validation approach instead of our iterative estimation. In the Courtyard scene (Fig. 1), we have generated an input image with 16 samples per pixel uniformly, and we have tested our proposed error estimation and the cross validation. Our estimation time (e.g., 16 ms) is $4\times$ lower than the cross validation time (e.g., 67 ms) due to the time complexity difference. In addition, we have compared reconstruction errors between the two approaches using the input image. Our reconstruction error (e.g., 0.00253) is similar to that (e.g., 0.00318) of the cross validation. As a result, we have chosen our proposed method because of its faster performance with similar or improved accuracy in practice.

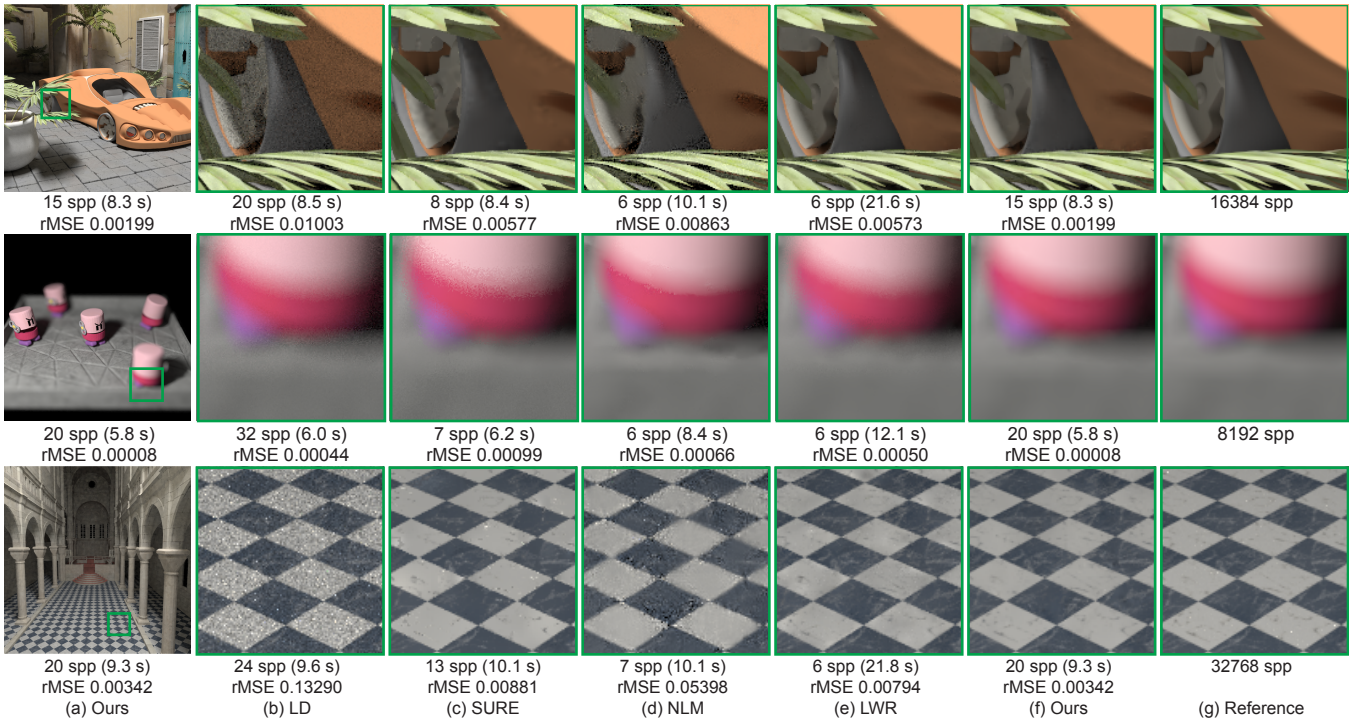


Figure 6: *Equal-time comparisons of the Courtyard (first row), Toasters (second row), and Sibenik scene (third row). The previous methods show sub-optimal results (e.g., under- or over-blurred results), since they can generate only small numbers of samples due to their relatively large filtering overheads. Our method, however, shows high-quality rendering results in terms of numerical accuracy and visual quality. Our method performs our filtering at only 11.6 %, 4.9 % and 6.0 % center pixels for the Courtyard, Toasters, and Sibenik scene, respectively. All the tested methods were integrated with Embree, a high-performance ray tracing system.*

8 Results and Discussions

We have demonstrated our method on top of the Embree example renderer, which fully utilizes high-performance ray tracing kernel [Wald et al. 2014]. Specifically, we use a path tracing implementation using the Embree kernel. For our tests, we use a 3.40 GHz processor and an NVIDIA GTX 980 GPU. To conduct comparisons between our method and previous work, we have implemented Stein's unbiased risk estimator (SURE) [Li et al. 2012], non-local means (NLM) [Rousselle et al. 2012], and locally weighted regression (LWR) [Moon et al. 2014] into the Embree renderer. To test NLM and LWR we used their CUDA implementations, provided by the respective authors and also converted CPU codes of SURE into a CUDA implementation in order to perform fair equal-time comparison. Furthermore, we have tested low discrepancy sampling (LD) that allocates ray samples uniformly and reconstructs images by a pixel filter (e.g., a small box filter), without performing any filtering. As a quantitative measure, we use the relative Mean Squared Error (rMSE) [Rousselle et al. 2011] that the state-of-the-art methods utilize.

Benchmarks. We have verified our method with the following scenes: 1) Courtyard, 2) Toasters, 3) Sibenik, 4) San Miguel, 5) Crown, and 6) Kitchen. We use 1K $\times$ 1K image resolutions for all benchmarks. The Courtyard scene (Fig. 1 and the first row in Fig. 6) has different types of shadows, which are introduced by a large area light source. For example, the noisy textured floor below the car has sharp shadow boundaries, which are non-linear functions that our linear prediction cannot have a large prediction size. In the Toasters scene (second row in Fig. 6), we show a depth-of-field effect on noisy textures, and this effect introduces a challenging scenario, since it is difficult to discern noisy textures from the noise introduced by the effect. In the Sibenik scene (third row in Fig. 6), we use a large area light source that introduces severe noise on the textured floor. The San Miguel benchmark (Fig. 9) contains a lot of high-frequency edges introduced by complex geometries, and strong depth-of-field effects are tested. As challenging scenarios for the reconstruction methods that use geometric features (including our method), we test the Crown and Kitchen benchmark (Fig. 11), where a lot of high-frequency edges (e.g., glossy reflections) that are not captured by the geometries are introduced.

Equal-time comparisons. Using the Toasters scene (second row in Fig. 6), we conduct equal-time comparisons. Surprisingly, the state-of-the-art methods, i.e., SURE, NLM, and LWR, do not introduce benefit over a baseline rendering method, a uniform rendering method using LD. This is mainly because their filtering and error analysis time are too large to generate enough samples for reducing errors. While the previous methods can be an excellent choice for accelerating offline rendering systems such as pbrt [Pharr and Humphreys 2010], their benefits are reduced or even disappear as rendering systems become more efficient. Our method, however, addresses this issue by using more aggressive reconstruction based on our novel prediction method. We use 4.9 % center pixels of the image resolution and predict all other pixels, but our result shows a similar quality with the reference image.

In the Sibenik benchmarks (third row in Fig. 6), NLM and SURE provide sub-optimal results on the noisy-textured floor, while the local linear approximation methods, i.e., LWR and our method, preserve the detailed features well. However, LWR suffers from an expensive filtering overhead. For example, LWR spends 6.5 s for computing its adaptive sampling map and 11.3 s for its final reconstruction. As a result, it was not possible to make the equal-time

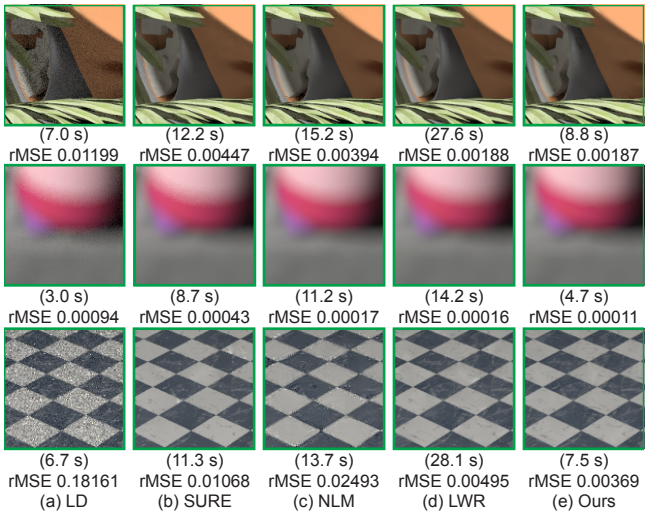


Figure 7: *Equal-sample comparisons with offline methods for the Courtyard (top), Toasters (middle), and Sibenik (bottom). All the methods use 16 spp on average. We show zoomed-in views shown in Fig. 6. Even with an equal-count sample, our method outperforms all the tested state-of-the-art methods in terms of numerical errors. Furthermore, our method has smaller adaptive rendering time over previous methods thanks to our prediction based reconstruction.*

comparison under 10 s for LWR. Our method tackles this problem by using sparse filtering performed at 6.0 % center pixels of the image resolution. We also observe similar results for the Courtyard scene (first row in Fig. 6).

Equal-sample comparisons. We conduct equal-sample comparisons with the prior offline methods (Fig. 7). Among all the tested scenes, our method shows numerically better results compared to other tested methods. In the Sibenik, LWR and our method show much better accuracy compared to SURE and NLM, by preserving the noisy textures on the floor well. However, the rendering time (28.1 s) of LWR is much higher than our rendering time (7.5 s), because of its expensive filtering overhead. Our method uses 11.6 %, 4.7 %, and 6.3 % linear models for the Courtyard, Toasters, and Sibenik scene respectively, compared to the number of linear models (i.e., total pixel count) used in LWR. Nevertheless, our approach using the sparse numbers of linear models performs high-quality adaptive rendering with a significantly reduced overhead.

Moreover, we measure the convergence of the relative MSE in terms of different ray counts (Fig. 8) given the Toasters and Sibenik scenes. Our method shows consistently better results compared to other methods, even if we apply our filtering to only a small number of center pixels. This indicates that our prediction based reconstruction can produce high quality reconstruction results in a reduced time without sacrificing a noticeable quality loss thanks to our adaptive prediction by using our error estimation.

Comparisons using offline ray tracers. We test our method and others with an offline rendering system, pbrt [Pharr and Humphreys 2010]. Given the San Miguel scene (Fig. 9), we compare our method with SURE, NLM, LWR, and a recent filtering method using the ray histogram fusion (RHF) [Delbracio et al. 2014]. To generate the results of RHF, we used the code provided by the authors. RHF tends to produce an over-blurred result in the focused area since it is difficult to discern detailed edges from high-frequency noise without utilizing G-buffers. The linear approximation methods based on G-buffers, LWR and our method, produce much better results compared to other techniques. However, our result produces high-quality rendering results using a sparse number of linear models (9.4 % center pixels). We also compare our method with a wavelet based adaptive method using the block-matching and 3D filtering (BM3D) [Kalantari and Sen 2013], and this result is in our supplementary report.

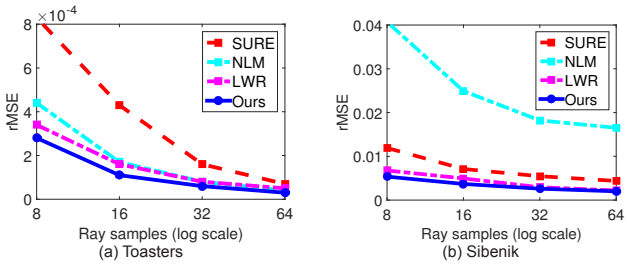


Figure 8: *Convergence of rMSE. We measure rMSE of different adaptive rendering methods for the Toasters and Sibenik scenes. Our method shows consistently better results numerically over other tested methods.*

Comparisons with real-time filtering. We compare our method with the edge-avoiding A-Trous [Dammertz et al. 2010] and guided filter [Bauszat et al. 2011], which are real-time filtering methods. The guided filter separates illumination into direct and indirect parts, and applies filtering only to indirect illumination. However it cannot remove noise (e.g., noise on soft shadows) from direct illumination, and thus we extend the original method to apply their filtering to a final input which includes both direct and indirect parts. Specifically, we additionally include textures as well as normals and depths which the original paper uses as features. To conduct a fair comparison, we use uniform sampling since the previous methods do not have adaptive sampling functionality. The real-time methods have reduced filtering time (e.g., 55 ms of the A-Trous) compared to our method (e.g., 562 ms), but our filtering errors (0.00536 and 0.00147) are significantly smaller than those of real-time filters (0.04927 and 0.01106 of the A-Trous and 0.01242 and 0.04633 of the guided filter). In addition, the real-time filters produce higher errors (0.01106 of the A-Trous and 0.04633 of the guided filter) compared even to the error of its input (0.00883) in the Courtyard scene; this is caused by the lack of robust error analysis.

Computational overhead. We measure time of individual tasks in our reconstruction given the Sibenik scene (first row in Fig. 10). We summarize the timing information in Table 2. Given accumulated buffers for input colors, normals, textures, and depths generated by MC ray tracing, we compute sample means and variances per pixel for each buffer (preprocessing step in Table 2) before we apply our reconstruction. Pre-filtering features (e.g., normals, textures, and depths) by SVD uses 24 % of the total reconstruction time, and our recursive reconstruction of linear models using error analysis spends a major portion (e.g., 64 %) of the total time.

Limitations and future work. A counterexample of filtering methods that utilize geometries is typically referred as a scenario where high-frequency details are mostly introduced by illumination changes (e.g., glossy reflections), as pointed out in [Rousselle et al. 2012]. Fig. 11 features this characteristic, and the geometry based filtering methods, i.e., SURE, LWR, and our method, show under- or over-blurred results. NLM preserves detailed edges, but leaves high-frequency noise since it is difficult to find enough neighboring patches on the detailed edges. Nonetheless, our method generates improved results compared to all the tested methods in terms of rMSE by increasing the number of linear models automatically

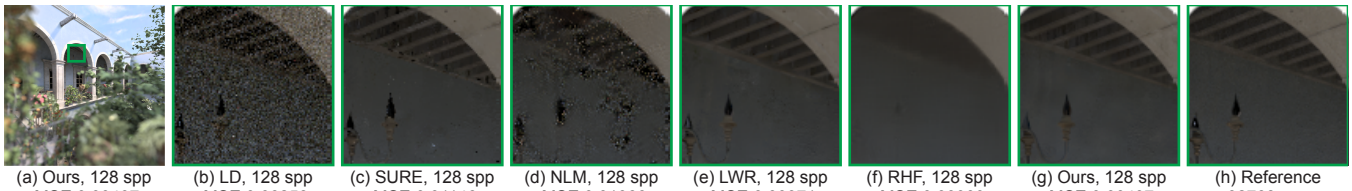


Figure 9: *Equal-sample comparisons for the San Miguel scene where complex geometries are tested with depth-of-field effects. The local linear approximation approaches, LWR and our method, show much better results compared to other methods. Nonetheless, our method achieves the high-quality rendering result with only 9.4 % center pixels compared to LWR.*

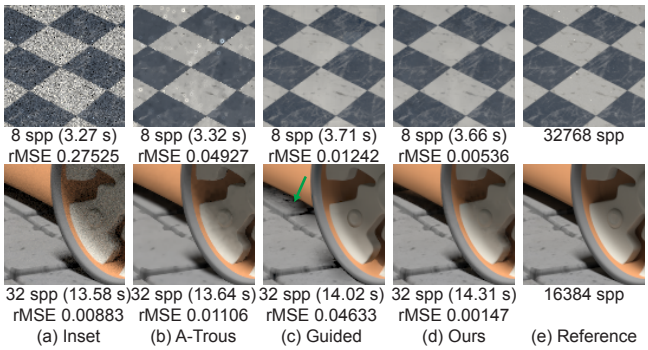


Figure 10: *Comparisons with real-time filtering methods. The A-Trous filter (b) leaves noises out (top and bottom rows), while the guided filter (c) removes the non-linear edges (i.e., shadow boundaries). The real-time filtering has a low filtering overhead (e.g., 55 ms on average for the A-Trous filter), but does not effectively reduce rMSE since it does not perform filtering based on robust error analysis. Our method (562 ms on average) is slower than the filter, but it shows much better results compared to the real-time filters.*

Table 2: *Timing breakdown for the Sibenik (Fig. 10)*

Task	Time (ms)	Percentage (%)
Preprocess input data	49	12
Pre-filter features by SVD	94	24
Compute linear models	252	64
Total reconstruction time	395	100

thanks to our adaptive prediction; the numbers of center pixels are 13.3 % and 18.9 % of the image resolution for the Crown (first row) and Kitchen scene (second row), respectively.

As an interesting future work, we would like to design robust rendering-specific features (e.g., virtual flash images [Moon et al. 2013]) that have a strong linear correlation with the reference image even for the aforementioned failure case. Our method, like other image-based solutions, uses brute-force sampling for high-dimensional space (e.g., lens) and does not utilize high-dimensional information for reconstruction. It would be interesting to extend our work to support the high-dimensional space, while maintaining a high efficiency. Our reconstruction treats each feature equally in the feature vector $\mathbf{x}$ for computational efficiency, but it would be desirable to use a non-uniform weighting function for considering the importance of different types of features [Moon et al. 2014].

9 Conclusions

We have proposed a novel prediction based reconstruction technique for high-quality and efficient MC ray tracing. An iterative computation of multiple linear models with varying prediction size is introduced, and its recursive error analysis for estimating the prediction error has been proposed. Our method selects an optimal linear model that is the least squares solution computed within an optimal prediction interval, and predicts all the other pixels within the interval without performing expensive optimization. We have tested our method with various scenes and recent high-performance ray tracing systems such as OptiX and Embree, and demonstrated that our method generates high quality results in a reduced rendering time compared to the state-of-the-art techniques.

Acknowledgements

We are thankful to Steven McDonagh, Philippe Le Prince, Mark Meyer, Markus Gross, and the anonymous reviewers for their insightful feedback. The Courtyard, Toasters, Sibenik, and San Miguel scenes are courtesy of Malgorzata Kosek, the Utah 3D Animation Repository, Marko Dabrovic (downloaded from McGuire's Graphics Archive), and Guillermo M. Leal Llaguno, respectively. The Crown model courtesy of Martin Lubich (www.loramel.net), HDR light courtesy of Lightmap Ltd (www.lightmap.co.uk). The Kitchen scene (designed by Nodexis) was purchased from TurboSquid (www.turbosquid.com). This work was funded by InnovateUK project #101858. Yoon is partly supported by the StarLab project of MSIP/IITP [R0126-15-1108] and NRF (2013R1A1A2058052).

References

ALLEN, D. M. 1974. The relationship between variable selection and data agumentation and a method for prediction. *Technometrics 16*, 1, 125–127.

BAUSZAT, P., EISEMANN, M., AND MAGNOR, M. 2011. Guided image filtering for interactive high-quality global illumination. *Computer Graphics Forum 30*, 4, 1361–1368.

DAMMERTZ, H., SEWTZ, D., HANIKA, J., AND LENSCH, H. P. A. 2010. Edge-avoiding A-Trous wavelet transform for fast global illumination filtering. In *High Performance Graphics*, 67–75.

DELBRACIO, M., MUSÉ, P., BUADES, A., CHAUVIER, J., PHELPS, N., AND MOREL, J.-M. 2014. Boosting Monte Carlo rendering by ray histogram fusion. *ACM Trans. Graph. 33*, 1, 8:1–8:15.

EGAN, K., TSENG, Y.-T., HOLZSCHUCH, N., DURAND, F., AND RAMAMOORTHI, R. 2009. Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Trans. Graph. 28*, 3, 93:1–93:13.

EGAN, K., DURAND, F., AND RAMAMOORTHI, R. 2011. Practical filtering for efficient ray-traced directional occlusion. *ACM Trans. Graph. 30*, 6, 180:1–180:10.

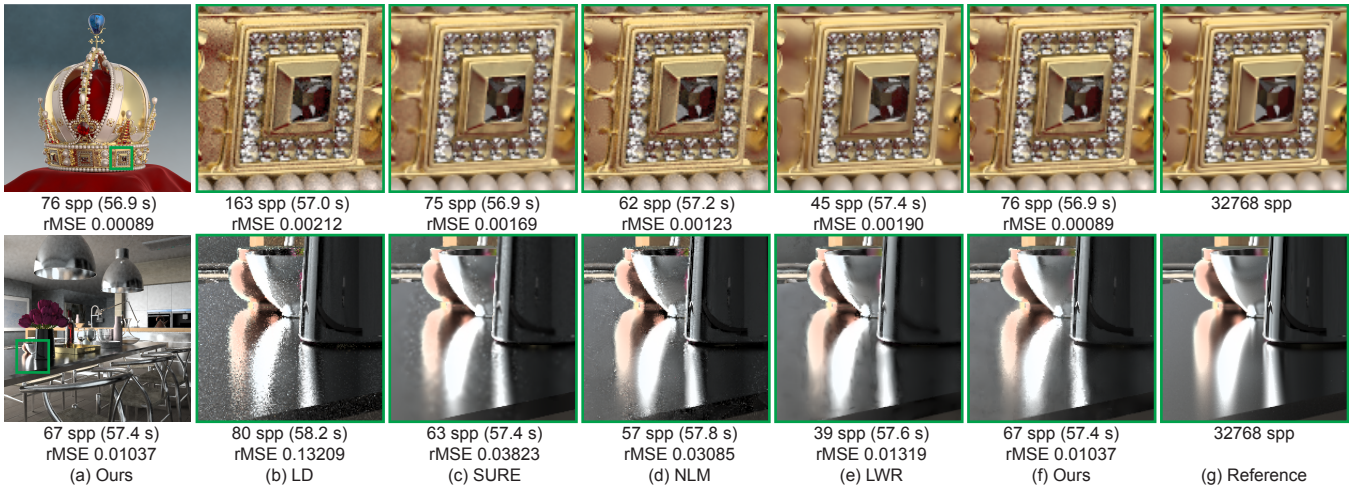


Figure 11: *Failure cases with the Crown (first row) and Kitchen scenes (second row). Most of the high-frequency edges are introduced by specular and glossy reflections, and thus the filtering methods utilizing geometries such as normals, textures and depths (i.e., SURE, LWR, and our method) can show sub-optimal results, since the features are not much helpful for detecting edges. Our method achieves the lowest error and preserves the detailed edges by adjusting prediction windows, but has some under-blurred results.*

EGAN, K., HECHT, F., DURAND, F., AND RAMAMOORTHI, R. 2011. Frequency analysis and sheared filtering for shadow light fields of complex occluders. *ACM Trans. Graph. 30*, 2, 9:1–9:13.

HACHISUKA, T., JAROSZ, W., WEISTROFFER, R. P., DALE, K., HUMPHREYS, G., ZWICKER, M., AND JENSEN, H. W. 2008. Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27*, 3, 33:1–33:10.

KAJIYA, J. T. 1986. The rendering equation. In *ACM SIGGRAPH '86*, 143–150.

KALANTARI, N. K., AND SEN, P. 2013. Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Computer Graphics Forum 32*, 2pt1, 93–102.

KOHAVI, R. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *14th International Joint Conference on Artificial Intelligence*, 1137–1143.

KŘIVÁNEK, J., BOUATOUCH, K., PATTANAIK, S. N., AND ŽÁRA, J. 2006. Making radiance and irradiance caching practical: Adaptive caching and neighbor clamping. In *Eurographics Symposium on Rendering*, 127–138.

LEHTINEN, J., AILA, T., CHEN, J., LAINE, S., AND DURAND, F. 2011. Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. 30*, 4, 55:1–55:12.

LEHTINEN, J., AILA, T., LAINE, S., AND DURAND, F. 2012. Reconstructing the indirect light field for global illumination. *ACM Trans. Graph. 31*, 4, 51:1–51:10.

LI, T.-M., WU, Y.-T., AND CHUANG, Y.-Y. 2012. Sure-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph. 31*, 6, 194:1–194:9.

LJUNG, L., AND SÖDERSTRÖM, T. 1987. *Theory and practice of recursive identification*. MIT Press.

MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. In *ACM SIGGRAPH '87*, vol. 21, 65–72.

MOON, B., JUN, J. Y., LEE, J., KIM, K., HACHISUKA, T., AND YOON, S.-E. 2013. Robust image denoising using a virtual flash image for Monte Carlo ray tracing. *Computer Graphics Forum 32*, 1, 139–151.

MOON, B., CARR, N., AND YOON, S.-E. 2014. Adaptive rendering based on weighted local regression. *ACM Trans. Graph. 33*, 5, 170.

OVERBECK, R. S., DONNER, C., AND RAMAMOORTHI, R. 2009. Adaptive wavelet rendering. *ACM Trans. Graph. 28*, 5, 140:1–140:12.

PARKER, S. G., BIGLER, J., DIETRICH, A., FRIEDRICH, H., HOBEROCK, J., LUEBKE, D., MCALLISTER, D., MCGUIRE, M., MORLEY, K., ROBISON, A., AND STICH, M. 2010. Optix: A general purpose ray tracing engine. *ACM Trans. Graph. 29*, 4, 66:1–66:13.

PHARR, M., AND HUMPHREYS, G. 2010. *Physically Based Rendering: From Theory to Implementation 2nd*. Morgan Kaufmann Publishers Inc.

ROUSSELLE, F., KNAUS, C., AND ZWICKER, M. 2011. Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph. 30*, 6, 159:1–159:12.

ROUSSELLE, F., KNAUS, C., AND ZWICKER, M. 2012. Adaptive rendering with non-local means filtering. *ACM Trans. Graph. 31*, 6, 195:1–195:11.

ROUSSELLE, F., MANZI, M., AND ZWICKER, M. 2013. Robust denoising using feature and color information. *Computer Graphics Forum 32*, 7, 121–130.

SOLER, C., SUBR, K., DURAND, F., HOLZSCHUCH, N., AND SILLION, F. 2009. Fourier depth of field. *ACM Trans. Graph. 28*, 2, 18:1–18:12.

WALD, I., WOOP, S., BENTHIN, C., JOHNSON, G. S., AND ERNST, M. 2014. Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph. 33*, 4, 143:1–143:8.

WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. 1988. A ray tracing solution for diffuse interreflection. In *ACM SIGGRAPH '88*, vol. 22, 85–92.